# OpenOfficeGeneration

## HOW TO USE OPENOFFICE FILES IN A GENERATOR IN COCOON.

Author: yves.vindevogel@implements.be

Date: 2003-03-14

---

Hi all,

I think I got a solution to use **OpenOffice (Writer) files as a generator in Cocoon**.

First of all, thanks to Con and Upayavira for pointing me at some very important details.

Okay, what I did ....

### files and folders

My cocoon folder looks like this:

```
cocoon/resources/entities
cocoon/implements/sxw
cocoon/implements/xsl
cocoon/implements/....
```

The sitemap for the Implements site is mounted as a submap of the sitemap in the /cocoon directory.

I've got the file "test.sxw" in my directory /implements/sxw

I've got the file "html.oowriter.xsl" in my directory /implements/xsl

I've got the files from, in my case opt/OpenOffice.org1.0/share/dtd/officedocument/1_0

(the DTDs from OpenOffice) copied to the folder /cocoon/resources/entities

I've copy the files 'common.xsl style_inlined.xsl table_cells.xsl table_rows.xsl global_document.xsl main_html.xsl style_header.xsl style_mapping.xsl table_columns.xsl table.xsl' from $OO_HOME/share/xslt/xhtml in my directory /implements/xsl

And edit the 'main_html.xsl':

1. set the following variables

<xsl:variable name="office:meta-file" select="/office:document/office:document-meta"/>

<xsl:variable name="office:styles-file" select="/office:document/office:document-styles"/>

<xsl:variable name="office:font-decls" select="$office:styles-file/office:font-decls"/>

<xsl:variable name="office:styles" select="$office:styles-file/office:styles"/>

b. edit all '/*/office:body' entries to '/office:document/*/office:body',

all '$office:meta-file/*' to '$office:meta-file'

in all the .xsl files, edit:

all '/*/office:body' to '/office:document/*/office:body'

Edit the style_headers.xsl:

edit the

<xsl:template name='create-css-styleheader'>

<xsl:comment>

<xsl:text>The CSS style header method for setting styles</xsl:text>

</xsl:comment>

<xsl:element name="style">

<xsl:attribute name="type">text/css</xsl:attribute>

<xsl:comment>

<xsl:text>

</xsl:text>

...

to

<xsl:template name='create-css-styleheader'>

<xsl:comment>

<xsl:text>The CSS style header method for setting styles</xsl:text>

</xsl:comment>

<xsl:element name="style">

<xsl:attribute name="type">text/css</xsl:attribute>

<xsl:comment>

<xsl:text>

BODY { background-repeat: no-repeat }

</xsl:text>

...

## Sitemap

This is my sitemap (well, the part for OO)

```
<map:match pattern="sxw/*.html">
  <map:aggregate element="office:document">
    <map:part src="jar:http://web/implements/{1}.sxw!/content.xml"/>
    <map:part src="jar:http://web/implements/{1}.sxw!/meta.xml"/>
<!-- additional for styles -->
    <map:part src="jar:http://web/implements/sxw/{1}.sxw!/styles.xml"/>
  </map:aggregate>
<!-- for using the standard OO transformation -->
     <map:transform src="xsl/main_html.xsl">
      <map:parameter name="metaFileURL" value="http://web/implements/sxw/{1}/meta.xml"/>
      <map:parameter name="stylesFileURL" value="http://web/implements/sxw/{1}/styles.xml"/>
      <map:parameter name="absoluteSourceDirRef" value="http://web/implements/sxw/{1}.sxw"/>
      <map:parameter name="jaredRootURL" value="http://web/implements/sxw/{1}"/>
     </map:transform>

<!--  Uncomment this, and comment previous if you want to use html.oowriter.xsl instead of standard OO
transformation
   <map:transform src="xsl/html.oowriter.xsl"/> -->
  <map:serialize type="html"/>
</map:match>
<map:match pattern="*.sxw">
  <map:read src="{1}.sxw" mime-type="application/zip"/>
</map:match>
<!-- additional for xml reading -->
    <map:match pattern="sxw/*/**.xml">
     <map:read src="jar:http://web/implements/sxw/{1}.sxw!/{2}.xml" mime-type="text/plain"/>
    </map:match>
<!-- additional for images -->
    <map:match pattern="sxw/*/Pictures/*.png">
     <map:read src="jar:http://web/implements/sxw/{1}.sxw!/Pictures/{2}.png" mime-type="image/png"/>
    </map:match>

    <map:match pattern="sxw/*/**.xml">
     <map:read src="jar:http://web/implements/sxw/{1}.sxw!/{2}.xml" mime-type="text/plain"/>
    </map:match>

    <map:match pattern="sxw/*.sxw">
     <map:read src="sxw/{1}.sxw" mime-type="application/zip"/>
    </map:match>
```

The basic element is this one is the "jar:http://" thing. Thanks to Conal for pointing me to this. OpenOffices uses a kind of zip file to store its files in. The Jar: protocol is able to read those zips straight away. Conal made a wiki page for this:

JarProtocolExample

The only problem is the need for a "http://web/....".

You need to specify the full path, the Jar: protocol does not work with the cocoon sitemap.

His wiki page shows the <map:read>, which works fine. You can read whatever you want from the zip file.

## Generator

I went further: I wanted to use the content.xml in that zip in a generator.

So I changed the <map:read> into <map:generate>

This results in an error:

```
message File "jar:http://web/implements/test.sxw!/office.dtd" not found.
description org.apache.cocoon.ProcessingException:
Failed to execute pipeline.: org.xml.sax.SAXParseException:
File "jar:http://web/implements/test.sxw!/office.dtd" not found.
```

This is correct, the dtd is not in the zipped file. It's only a reference.

## Fixing problem 1: modify the catalog

I modified the /resources/entities/catalog file. I added two lines

(Thanks Upayavira for showing me this thing)

```
-- Open Office DTDs --
PUBLIC "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "office.dtd"
PUBLIC "-//OpenOffice.org//DTD Manifest 1.0//EN" "Manifest.dtd"
```

That corrects the error, but gives a new one ....

```
org.apache.cocoon.ProcessingException: Failed to execute pipeline.: org.xml.sax.SAXParseException:
A '(' character or an element type is required in the declaration of element type "draw:text-box".
```

## Fixing problem 2: modify the DTD

Appearantly, there's something in that file. I looked at it, but I could not find a mistake. I'm not an expert on those things, so, maybe someone can look into it for me ....

I used a workaround however. You can modify the
/resources/entities/office.dtd (from OO) file. There's entries like this:

```
<!ENTITY % dtypes-mod SYSTEM "dtypes.mod">
%dtypes-mod;
<!ENTITY % nmspace-mod SYSTEM "nmspace.mod">
%nmspace-mod;
<!ENTITY % office-mod SYSTEM "office.mod">
%office-mod;
<!ENTITY % style-mod SYSTEM "style.mod">
%style-mod;
<!ENTITY % meta-mod SYSTEM "meta.mod">
%meta-mod;
<!ENTITY % script-mod SYSTEM "script.mod">
%script-mod;
<!ENTITY % drawing-mod SYSTEM "drawing.mod">
%drawing-mod;
<!ENTITY % text-mod SYSTEM "text.mod">
%text-mod;
<!ENTITY % table-mod SYSTEM "table.mod">
%table-mod;
<!ENTITY % chart-mod SYSTEM "chart.mod">
%chart-mod;
<!ENTITY % datastyl-mod SYSTEM "datastyl.mod">
%datastyl-mod;
<!ENTITY % form-mod SYSTEM "form.mod">
%form-mod;
<!ENTITY % settings-mod SYSTEM "settings.mod">
%settings-mod;
```

When you remove them all, the generator should work (Sax does not check a lot, this is a workaround, not a solution 😉)

## Aggregation

Next thing I did, was to use the <map:aggregate> to combine the files from the zipped file into one. I enclose the original xml from all the files in a new <document> tag. At first, I used the <office:document> tag, but this gave problems in my XSL.

After that, it outputs both files as a combined xml file, like we needed !

## Sample XSL

I then reused an xsl file I wrote before, when I extracted the files to xml with a little perl too. This xsl file generates a very simple page in html, based on the document ...

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:dc="http://purl.org/dc/elements/1.1/"
                              xmlns:office="http://openoffice.org/2000/office"
```

```
                              xmlns:style="http://openoffice.org/2000/style"
                              xmlns:text="http://openoffice.org/2000/text"
                              xmlns:table="http://openoffice.org/2000/table"
                              xmlns:draw="http://openoffice.org/2000/drawing"
                              xmlns:fo="http://www.w3.org/1999/XSL/Format"
                              xmlns:xlink="http://www.w3.org/1999/xlink"
                              xmlns:number="http://openoffice.org/2000/datastyle"
                              xmlns:svg="http://www.w3.org/2000/svg"
                              xmlns:chart="http://openoffice.org/2000/chart"
                              xmlns:dr="http://openoffice.org/2000/dr"
                              xmlns:math="http://www.w3.org/1998/Math/MathML"
                              xmlns:form="http://openoffice.org/2000/form"
                              xmlns:script="http://openoffice.org/2000/script"
                              xmlns:config="http://openoffice.org/2001/config"
                              xmlns:meta="http://openoffice.org/2000/meta"
                              xmlns:manifest="http://openoffice.org/2001/manifest"
                              office:class="text" office:version="1.0">

<xsl:param name="relpath"/>
<xsl:param name="content"/>

<xsl:template match="/document">
  <html>
    <head>
      <link type="text/css" rel="stylesheet" href="{$relpath}/css/general.css"/>
    </head>
    <body>
      <xsl:if test="$content='only'">
        <xsl:attribute name="class">content</xsl:attribute>
        <xsl:apply-templates select="office:document-content/office:body/*"/>
      </xsl:if>
      <xsl:if test="$content='  '">
        <div id="divTitle" class="title">
          <xsl:value-of select="$content"/>
          <xsl:value-of select="office:document-meta/office:meta/dc:title"/>
        </div>
        <div id="divContent" class="content">
          <iframe src="?content=only" style="width:100%; height:100%" frameborder="0"/>
        </div>
      </xsl:if>
    </body>
  </html>
</xsl:template>

<xsl:template match="text:p">
  <p>
    <xsl:for-each select="node()">
      <xsl:choose>
        <xsl:when test="self::text()">
          <xsl:value-of select="."/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="."/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </p>
</xsl:template>

<xsl:template match="text:p" mode="table">
  <xsl:for-each select="node()">
    <xsl:choose>
      <xsl:when test="self::text()">
        <xsl:value-of select="."/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>
```

```
<xsl:template match="text:a">
  <a href="{@xlink:href}">
    <xsl:value-of select="."/>
  </a>
</xsl:template>

<xsl:template match="table:table">
  <table>
    <xsl:for-each select="table:table-header-rows/table:table-row">
      <tr>
        <xsl:for-each select="table:table-cell">
          <th>
            <xsl:value-of select="text:p"/>
          </th>
        </xsl:for-each>
      </tr>
    </xsl:for-each>

    <xsl:for-each select="table:table-row">
      <tr>
        <xsl:choose>
          <xsl:when test="position() mod 2 = 1">
            <xsl:attribute name="class">odd</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute name="class">even</xsl:attribute>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:for-each select="table:table-cell">
          <td>
            <xsl:apply-templates select="." mode="table"/>
          </td>
        </xsl:for-each>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>

<xsl:template match="text:line-break">
  <br/>
</xsl:template>

<xsl:template match="text:h">
  <xsl:element name="h{@text:level}">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:template>

<xsl:template match="text()">
  <!-- ignore unwanted text nodes -->
</xsl:template>

</xsl:stylesheet>
```

Et voila !!!! That's it ....

Once again, thanks to all who helped.

Could somebody please check this to see if he/she could reproduce my work on his/her machine ??

Yves Vindevogel

Implements

Mail: yves.vindevogel@implements.be – http://www.implements.be

Quote: The winner never says participating is more important than winning.