DotNetOdf

User Documentation

Version 0.6

www.notreallyorm.com

1. Introduction

During the last years both "Big Players" for office software developed open document standards basing on XML: The OASIS format for OpenOffice / LibreOffice (I will abbreviate it with OLOffice) and the Office Open XML format for Microsoft Office. Both formats are zipped archives with several XML files and additional data (for pictures and so on). For a deeper explanation have a look into Wikipedia (for German speaking people: the German version provides more detailed information).

There are also a set of SDK's to build those formats from scratch or to manipulate them, some of them with very sophisticated features and corresponding learning curve (look at the Microsoft SDK for the Office Open XML).

The DotNetOdf library focuses mainly on easy usage. You should be able to create reports with lists or serial letters from templates with just a few lines of code. The user should be able to build templates without the help of a programmer. Some experience in using OLOffice or MS Office is be enough. Important for business users: At runtime you do not need OLOffice or MS Office to create the documents (so you also won't need a license for them).

Some years ago a published a library named PyOpenOffice that did the same for the old OpenOffice SXW format (and much more, even PDF generation...). Have a look at: http://www.bezirksreiter.de/PyOpenOffice.htm. It depended on Python, several external Python libraries and an XSLT processor (if you wanted the PDF generation).

The DotNetOdf library is written in C# and requires a .NET framework version 3.5 (or higher). It has less functionality, but it requires no setup, no entries in your Windows Registry and it integrates seamlessly into your C# or VB.NET projects. It supports not only the OLOffice formats, but also the MS Word DOCX. And: It works without any access to the file system (see below).

Thanks go to the developers of the DotNetZip library (http://dotnetzip.codeplex.com/). DotNetOdf uses it to unpack and repack the document archives. I use this library also for other (commercial) project and can warmly recommend it. Have a look at the DotNetZip homepage: They ask you to give a donation for a youth project if you want.

2. License

Like the NotReallyORM framework the DotNetOdf library is distributed under the *Apache 2.0 license*. You are encouraged to use it in both free or commercial applications.

3. Installation

For usage you should have installed at least version 3.5 of the .NET framework. Unzip the DotNetOdf.zip package into your path. After unzipping you will also find the IonicZip.dll (DotNetZip) that is needed by DotNetOdf and has to be in the same path at runtime. In your project set a reference only to DotNetOdf.dll. Make sure that the DLL is found by your application at runtime (for example in your setup program).

4. Principles

Technically spoken what you do is this: You create a template document with some special markers to be replaced by data from a database query for example. DotNetOdf unzips the archive and loads the content.xml resp. document.xml into a DOM tree (represented by the .NET XmlDocument class). Now it looks for the marked text nodes in the DOM using XPath and so on. It replaces the markers and packs back the manipulated XML to the archive. *It is important to know that*

DotNetOdf does not use any temporary files - everything "file like" is done with in-memory streams or bytearrays (this is an important difference to my good old PyOpenOffice library). For every manipulation you have to instantiate a new Doctools object. The information in it is destroyed together with the object during the garbage collection. So it should be no problem to use DotNetOdf in environments with restricted access to the file system, for security critical tasks or for multi-threaded usage on a web server.

5. A Simple Example

Imagine you have a template with:

@@-FirstName-@@
@@-Surname-@@
@@-City-@@

City in the Table	Surname in the Table	FirstName in the Table
Multi:@@-City-@@	Multi:@@-Surname-@@	Multi:@@-FirstName-@@

After filling in the data you will have:

Martin
Simon
Oldenburg

City in the Table	Surname in the Table	FirstName in the Table
Oldenburg	Simon	Martin
Seattle	Doe	John

I will show you how to perform this task with just a few lines of code:

5.1. Single Replacement

Example: Fill address data into a letterhead.

Copy the "test_2.odt" or "test_2.docx" template that ships with the zip package into your application path. Load it:

```
Byte[] myDocumentBytes = System.IO.File.ReadAllBytes("test_2.odt");
```

Create a new DocTool for the odt archive:

```
myDoctool = new Doctools(myDocumentBytes, Doctools.mode.ODT, "@@-",
"-@@);
```

The markers are the same you use in your template (for example "@@-city-@@"). The left and right markers always must be different (otherwise you will get an exception).

Create a dictionary and fill it with the values you want to use to replace the template fields:

```
Dictionary<String, String> myDict = new Dictionary<String, String>();
myDict.Add("FirstName", "Martin");
myDict.Add("Surname", "Simon");
myDict.Add("City", "Oldenburg");
```

If you have a LazyListRow from the NotReallyORM framework and your template shares the replacement fields with the database fieldnames this will be shorter:

```
Dictionary<String, String> myDict = myLazyListRow.toDictionary();
```

Do the replacement:

```
myDoctool.replaceFieldVariables(myDict);
```

Restore the modified content.xml to the odt archive that is in stored your DocTool:

```
myDoctool.restoreContentXMLToArchive();
```

Retrieve the modified archive:

```
Byte[] myNewArchive = myDoctool.getArchive();
```

Now you can send back the modified new created odt file with your webserver. Or you write it back to your file system:

```
System.IO.File.WriteAllBytes(modifiedDocument, myNewArchive);
```

Remember: After myDoctool is deleted by the garbage collection no data will be left in your file system. DocTool has done its work entirely in-memory.

5.2. Table Replacement

Example: Create an address list from a database.

Most things will work similar to the single replacement. *Mind the "Multi:" in the replacement fields of your template.* When you get your DocTool by:

```
myDoctool = new Doctools(myDocumentBytes, Doctools.mode.ODT, "@@-",
"-@@);
```

Then the table replacement fields in your template have to be like: *Multi:@@-MyField-@@*.

Instead of a dictionary you now create a list of dictionaries and fill it with the values:

```
Dictionary<String, String> myDict1 = new Dictionary<String, String>();
myDict1.Add("FirstName", "Martin");
myDict1.Add("Surname", "Simon");
myDict1.Add("City", "Oldenburg");

Dictionary<String, String> myDict2 = new Dictionary<String, String>();
myDict2.Add("FirstName", "John");
myDict2.Add("Surname", "Doe");
myDict2.Add("City", "Seattle");

List<Dictionary<String, String>> myList = new List<Dictionary<String,
String>>();
```

If you have a LazyList from the NotReallyORM framework (Version 0.7 and higher) and your template shares the replacement fields with the database fieldnames this will be shorter:

```
List<Dictionary<String, String>> myList =
myLazyList.toListOfDictionaries();
```

Do the replacement:

```
myDoctool.multiReplaceFieldVariables(myDict);
```

Caution: When you use single replacement and table replacement together, always do it in this order:

1. **Table Replacement**
2. **Single Replacement**
3. **Restore Content**

6. Important Hints for your Templates

The word processors do not always produce "streamlined" XML. So here are miscellaneous hints and warnings when you design your templates:

1. In DOCX-Templates you must register your marker fields as correct spelled words. Check for the red line under it. MS Word marks misspelled words in the document itself (not only in the editor at runtime), so the recognition of the markers will fail.
2. Sometimes you may want to use formatted marker fields like. In these cases always first write your marker field without any formatting: @@-City-@@. Then mark it in your word processor: @@-City-@@. Finally choose the formatting: ***@@-City-@@***.

##

To learn more have a look at the DotNetOdf_Winforms project that ships with the sourcecode. There you find some more features of the DocTool object.

More documentation to be expected...