SCRIPTING THE ODF TOOLKIT

(ODFDOM)

Seminar Paper

Günter Mayer 8952223

LV 581 Projektseminar aus Wirtschaftsinformatik SS 2011

ao.Univ.Prof.Dr.Rony G. Flatscher

Institute for Management Information Services

Vienna University of Economics and Business Administration

Abstract

This paper is about using the ODFDOM Toolkit via OpenObjectRexx.

The paper shows possibilities to produce and manipulate office documents (text, spreadsheet, presentation). At first, a short guidance on how to install all the essential components, to enable the use of the presented examples or to develop own examples, is shown. A short view on the structure of an ODF document and the structure of the ODFDOM should make it clear to the reader, how it is possible to use a scripting language to deal with ODF documents. One important part of this paper contains small examples, which demonstrate to the reader how the scripting of the ODFDOM Toolkit can be handled.

Contents

List of Figures

1 Introduction

This chapter is trying to give the reader an overview of the structure and the main assignment of this work.

ODFDOM is a Java library to enable the access to ODF documents. Java is a very powerful language, but for beginners not easy to work with. With OORexx, an easy to use scripting language, using the BeanScripting Framework BSF4ooRexx it is possible to get access to ODF documents. However it is not necessary to learn Java but it is very helpful to have a basic understanding of how to read Java documentations. A rich source for information about Java is the documentation of the Java 2 SDK[1] [JaSd01]. All the necessary prerequisites to get the examples to work will be shown. OORexx can be written in every text editor. Using a text editor with syntax highlighting like VIM will be helpful because it makes the work much easier [ViEd01].

The main Part of the work shows some examples where different options to deal with ODF documents are demonstrated. The examples show only a small fraction of the possibilities that ODFDOM offers. They should facilitate the introduction to this interesting matter.

1.1 Keywords

Open Object Rexx

Bean Scripting Framework for Open Object Rexx

ODFDOM

Xerces

ODF

Open Office

1 *Software Development Kit*

2 Installation Guide

A few basic prerequisites to use the techniques are described in the following chapters.
The following instructions are tested on a Windows Vista /32bit system with ServicePack 2.

2.1 Java

A working Java runtime environment is needed. From http://www.java.com the latest version of the Java runtime environment can be downloaded. Normally the website changes automatically to the current language of the operating system (e.g. http://www.java.com/de/ for a German system).

2.2 Open Office

The next step is to download the newest version of OpenOffice from the OpenOffice.org homepage. A version of the installer with an included Java runtime is also available on the website. The examples in chapter 5 will also work without an OpenOffice installation. The shown methods, will create documents in the ODF format. So all other programs, that are able to view the ODF format, can be used to have a look at the created documents.

2.3 OoRexx[2]

Open Object Rexx the freely available version of the scripting language Object Rexx is downloadable from http://www.oorexx.org/. All the examples in this paper are tested with version 4.1.0.

2.4 BSF4ooRexx[3]

BSF4ooRexx offers the link between Java and ooRexx [BsRx01].
The latest version of BSF4ooRexx can be retrieved from the home at sourceforge.net.
http://sourceforge.net/projects/bsf4oorexx/

2 OpenObjectRexx
3 BeanScriptingFramework for OORexx

Take care that previous versions have been completely uninstalled prior to the installing process.

2.5 ODFDOM[4]

Furthermore the ODFDOM library is needed. The current version 0.8.7 released on Feb 16th 2011 can be downloaded from http://odftoolkit.org/projects/odfdom/pages/Home . After unpacking, the odfdom.jar file has to be added to the classpath, to enable java to find the appropriate classes.

The author uses the path: `C:\ProgramFiles\Java` to store the "`odfdom.jar`". Figure 1 shows the appropriate entry to the environment variable.

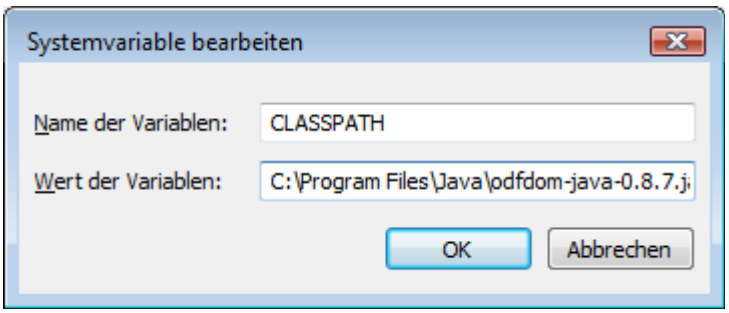


Figure 1: Environment Variable

In Windows Vista the classpath variable can be found as follows:

`"Control Panel" / "System" / "Advanced System Settings" / "Environment Variables" / "System Variables" (for all users) / "CLASSPATH"/ "Edit" (for modifying the existing variable).`

2.6 Xerces

Another tool that is needed is the Java XML processor "Xerces" from the Apache Software foundation [ApaSof]. It can be downloaded from

http://tweedo.com/mirror/apache//xerces/j/Xerces-J-src.2.11.0-xml-schema-1.1-beta.zip

The `xercesImpl.jar` and the `xml-apis.jar` file have to be added to the environment variable to enable java to find the appropriate classes. Both files are located in the Xerces package.

4 OpenDoumentformat Document Object Model

3 ODF Definition

ODF - Open Document Format is an international standard for office documents. When taking a look at today's office environment it can be seen, that many documents are stored in a proprietary format. This presents no problem as long as they are handled in an organization that uses the same proprietary office solution. But as soon as word processing documents, spreadsheets, etc. are being exchanged with other organizations, which use a different proprietary office solution, some problems may occur. There are some converters but it's possibly that they lead in loss of formatting information. Another problem may arise when the provider of the proprietary software decides to change its data format. The access to existing documents may no longer be possible.

OASIS Open Document Format for Office Applications (short form: ODF) is an internationally standardized open-source standard for file formats for office documents such as text, spreadsheets, presentations, drawings, pictures and diagrams. It is application- and vendor-neutral. OpenDocument uses an XML[5]-based markup language for the document which elements are based on the HTML standard. For mathematical formulas a subset of MathML[6] is used. For formatting information an own XML-based language is used. OpenDocument can be supplemented with any other XML language. It was originally developed by Sun Microsystems [SuMi01], and specified by the OASIS standards organization [OaSi01].

3.1 Inside an OpenDocument File

An OpenDocument file is a collection of `XML` files and other objects (such as embedded images) that are combined into a single file in `ZIP` format. These archive files contain special items that are geared to the structure of the `JAR` (Java archive format) [JaAr01]. Examples for file extensions of OpenDocument files:

- text: .odt
- spreadsheet: .ods
- presentations: .odp

5 Extensible Markup Language
6 Mathematical Markup Language

The type of the file can also be determined independent of the file extension and without decompression, because the first entry in the archive is the MIME[7] type of the file content.

The entire XML file is very comprehensive but human readable. To save space the JAR format is used. A `JAR` file is a packed zip file with a "manifest" file that lists the content of the archive. The manifest.xml file is plain text and uncompressed.

Figure 2 shows a short text document which got saved with the name "`text.odt`".

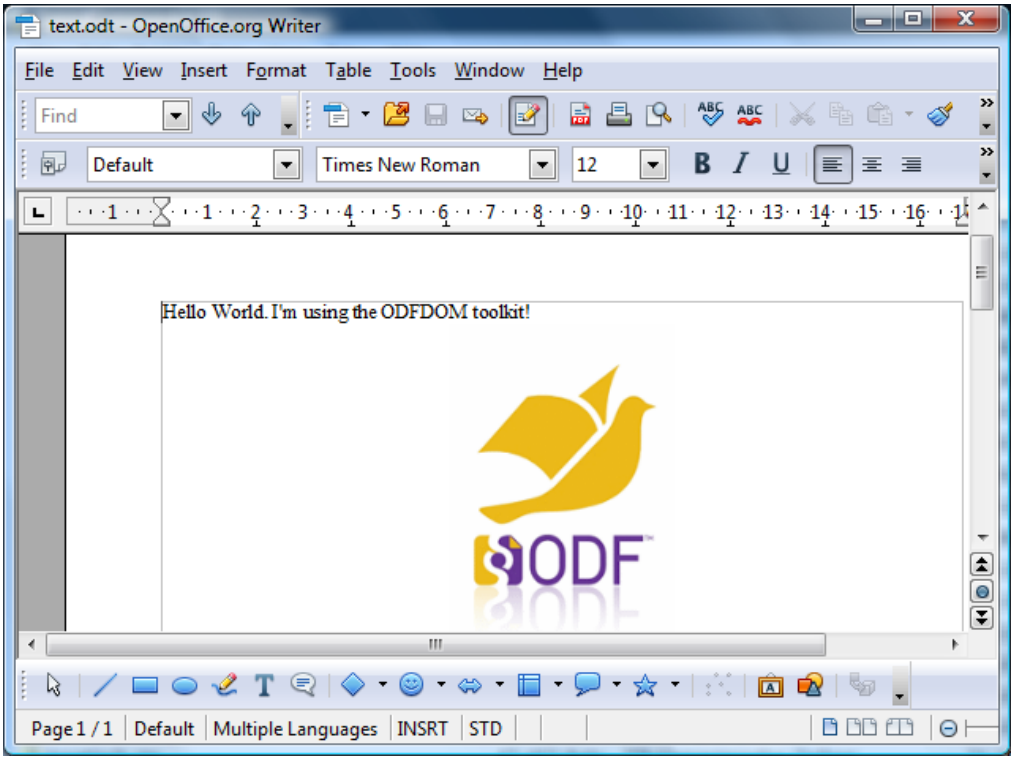
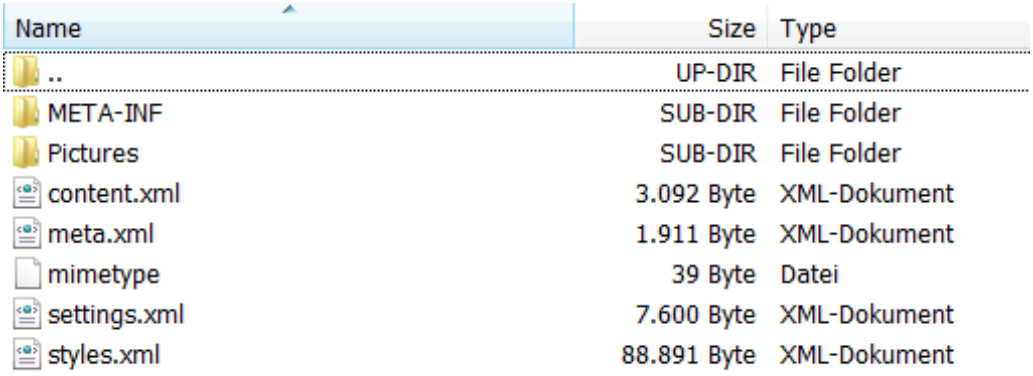


Figure 2: Example Text Document

It is possible to use any `ZIP` file tool to unpack an Open Document file because a `JAR` file is also a `ZIP` file. Once unpacked, the XML is human readable.

The next figure shows the unzipped text document.

7 *Multipurpose Internet Mail Extensions*



Name	Size	Type
..	UP-DIR	File Folder
META-INF	SUB-DIR	File Folder
Pictures	SUB-DIR	File Folder
content.xml	3.092 Byte	XML-Dokument
meta.xml	1.911 Byte	XML-Dokument
mimetype	39 Byte	Datei
settings.xml	7.600 Byte	XML-Dokument
styles.xml	88.891 Byte	XML-Dokument

Figure 3: Listing of Unzipped Text Document

3.2 The ODF Files in Detail

- `META-INF/manifest.xml`

  The manifest file contains a list of all other files in the archive. OpenOffice needs this information to read the ODF file. It is a meta-information about the whole `JAR` file. The following picture showes an example of the content.

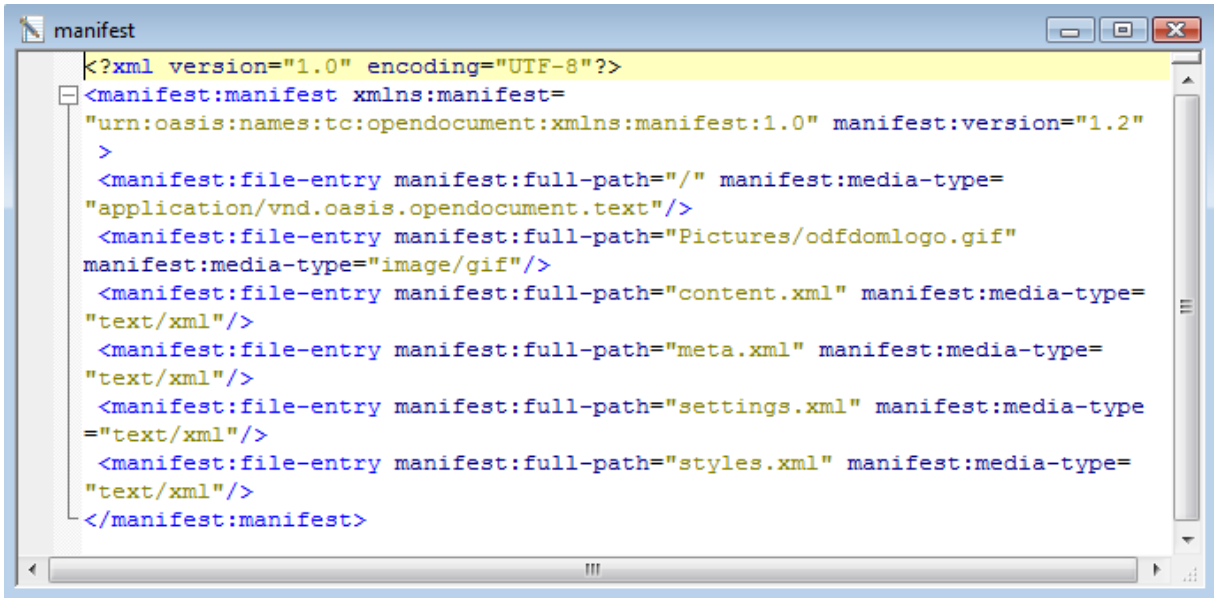


Figure 4: Content of manifest.xml

- `Pictures`

  All images which are included in the document are listed in the directory.

- `content.xml`

  This is the content of the document.

```
office:version="1.2">
    <office:scripts/>
  - <office:font-face-decls>
        <style:font-face svg:font-family="'Times New Roman'" style:name="Times New
            Roman" style:font-pitch="variable" style:font-family-generic="roman"/>
        <style:font-face svg:font-family="'Lucida Sans Unicode'" style:name="Lucida
            Sans Unicode" style:font-pitch="variable" style:font-family-
            generic="system"/>
        <style:font-face svg:font-family="Tahoma" style:name="Tahoma" style:font-
            pitch="variable" style:font-family-generic="system"/>
    </office:font-face-decls>
    <office:automatic-styles/>
  - <office:body>
    - <office:text>
      - <text:sequence-decls>
            <text:sequence-decl text:name="Illustration" text:display-outline-
                level="0"/>
            <text:sequence-decl text:name="Table" text:display-outline-level="0"/>
            <text:sequence-decl text:name="Text" text:display-outline-level="0"/>
            <text:sequence-decl text:name="Drawing" text:display-outline-level="0"/>
        </text:sequence-decls>
        <text:p text:style-name="Standard">Hello World. I'm using the ODFDOM
            toolkit!</text:p>
      - <text:p>
        - <draw:frame text:anchor-type="paragraph" svg:width="4.76cm"
            svg:height="7.812cm">
            <draw:image xlink:type="simple"
                xlink:href="Pictures/odfdomlogo.gif"/>
            </draw:frame>
        </text:p>
        <text:p>Ok...that looks fine!</text:p>
    </office:text>
    </office:body>
</office:document-content>
```

Figure 5: Excerpt out of content.xml

- `meta.xml`

  Some meta-information about the document like the creation-date or author, etc.

- `Mimetype`

  The MIME type for the document is defined in a single line.

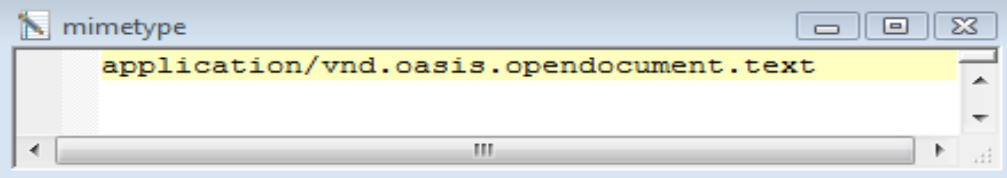


Figure 6: MIME Type

There are a lot of other types like:

`application/vnd.oasis.opendocument.spreadsheet`

`application/vnd.oasis.opendocument.presentation`

- `settings.xml`

  Settings.xml contains information which enables the application to show the correct settings. In a text document for example the visibility of the headers and footers or the zoom factor. In a spreadsheet the height and width of the rows and columns etc.

- `styles.xml`

  The styles which are used to format the content.

4 The ODFDOM Toolkit

The ODF Toolkit Union is an open source community that develops different tools to work with ODF documents. One of these tools is the ODFDOM. The project defines the ODFDOM API[8] as follows: "*ODFDOM is a free Open Document Format (ODF) library. Its purpose is to provide an easy common way to create, access and manipulate ODF files, without requiring detailed knowledge of the ODF specification. It is designed to provide the ODF developer community with an easy lightwork programming API portable to any object-oriented language." [OdDm01].*

Because of the modular structure a layered design is being used.

4.1 The XML Layer

The XML Layer offers the functionality for handling the office format like tables, images etc. [OdDm02].

The layer consists of two APIs, each enables a different access to the ODF document.

The ODF DOM API allows to access all XML elements of the ODF document. For each ODF XML-element and XML-attribute a separate class exists.

The ODF Document API enables the access at a very high level. Therefore it enables, for many common applications, an easy approach without detailed knowledge of the ODF XML implementation [OdDm03].

4.2 The ODF Package Layer

The ODF package layer consists of the ODF Package API, which handles all the features which are defined in the third part of the ODF 1.2 specification [OdSp01]. As explained in chapter 3, the ODF document simply consists of a few files representing the different elements of the document.

In the ODF standard all file streams, except the image folder, are defined. It is possible to write individual elements directly into the file streams. This requires the following prerequisites.

8 *Application Programming Interface*

All file streams of the document have to be unzipped. All the files of the package have to be enlisted in the `/META-INF/manifest.xml file`. The package has to start with the unpacked "`mimetype`" file to identify the type of the document [OdDm03].

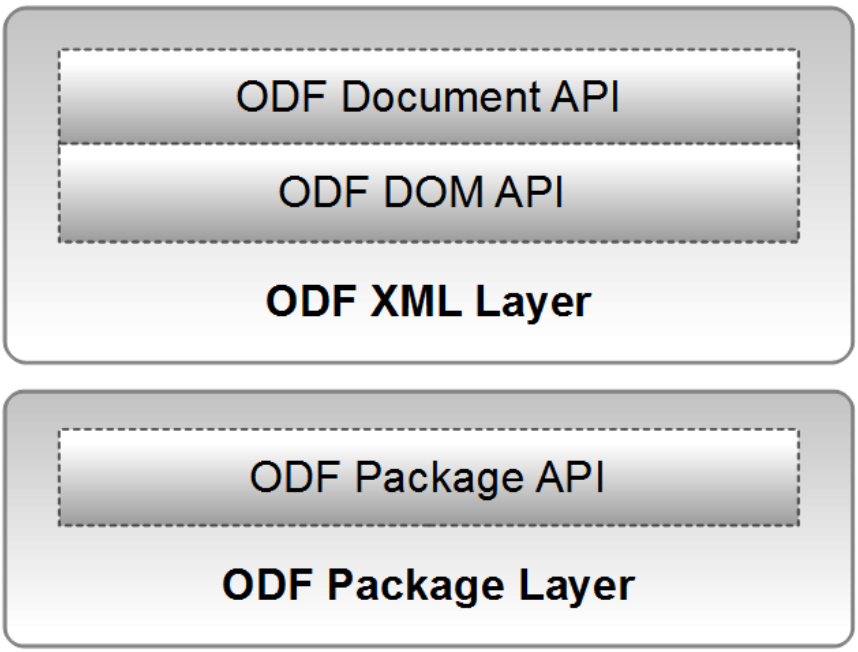


Figure 7: ODF Layers [OdOv01]

5 Examples

When running the examples some essential points have to be observed. The ooRexx files have the file extension `.rxj`, which means they need Java support.
The files `"importmaster.odt", "importtext.ods", "odfdomlogo.gif"` are used by some of the examples and have to be located in the same directory.
The examples must be used in a specific order, as they partially build on each other.

- Example 1 generates `"text.odt"`
- Example 2 uses `"text.odt" and generates "text2.odt"`
- Example 3 uses `"text.odt"` and generates `"newname.odt"`

- Example 4 needs `"importmaster.odt"` and `"importtext.ods"` creates multiple files `"letter#.odt"`

- Example 5 generates `"spreadsheet.ods"`
- Example 6 uses `"spreadsheet.ods"`
- Example 7 uses `"spreadsheet.ods"` and generates `"spreadsheet2.ods"`

- Example 8 generates `"presentation.odp"`
- Example 9 uses `"presentation.odp"`
- Example 10 uses `"presentation.odp"` and generates `"presentation2.odp"`

All generated files are stored in the same directory as the examples.

5.1 Create a Text Document ("createodt.rxj")

The first example shows how to create a text document. The first step is to import the appropriate class of the ODFToolkit. All classes, sub classses and methods are listed in the ODFDOM API documentation in detail [OdAp01]. The method `"newTextDocument"` is

passed on to the variable "outputOdt" and thus creates a new blank text document. With "addText" the specified text is attached to the end of the document. In a blank document, as in this case, the end is the first line. The next step inserts a blank paragraph.

When an image has to be inserted, the correct path to the desired image has to be specified. Therefore a URI is used. In Java a URI[9] object is described by the class "java.net.URI" [JaUr01]. As defined in the class description there are different types of URI specifications. In this example a relative path is used. This means that the path to the desired document describes just the difference to the path from which it was called. Since the image in this example is in the same folder, only the file name is needed.

The next step adds again a new paragraph, but this time with some content. The document now needs to be stored. Therefore the method "save" with the appropriate name has to be used.

The BSF4ooRexx class "BSF.CLS" helps to use common steps when working with BSF. The "::requires BSF.CLS" line at the end of the ooRexx script makes this support available.

```
-- Example 01 "createodt.rxj"
-- Create a new text document, write some text and add a picture

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
outputOdt=clz~newTextDocument

-- add text
outputOdt~addText("Hello World. I'm using the ODFDOM toolkit!")

-- add a picture in a new paragraph
outputOdt~newParagraph
outputOdt~newImage(.bsf~new("java.net.URI","odfdomlogo.gif"))

outputOdt~newParagraph("Ok...that looks fine!")

-- save the document
outputOdt~save("text.odt")

::requires BSF.CLS
```

Figure 8 Example 01 Creating a Text Document

The result of example 01 can be seen in the next figure.

9 Uniform Resource Identifier

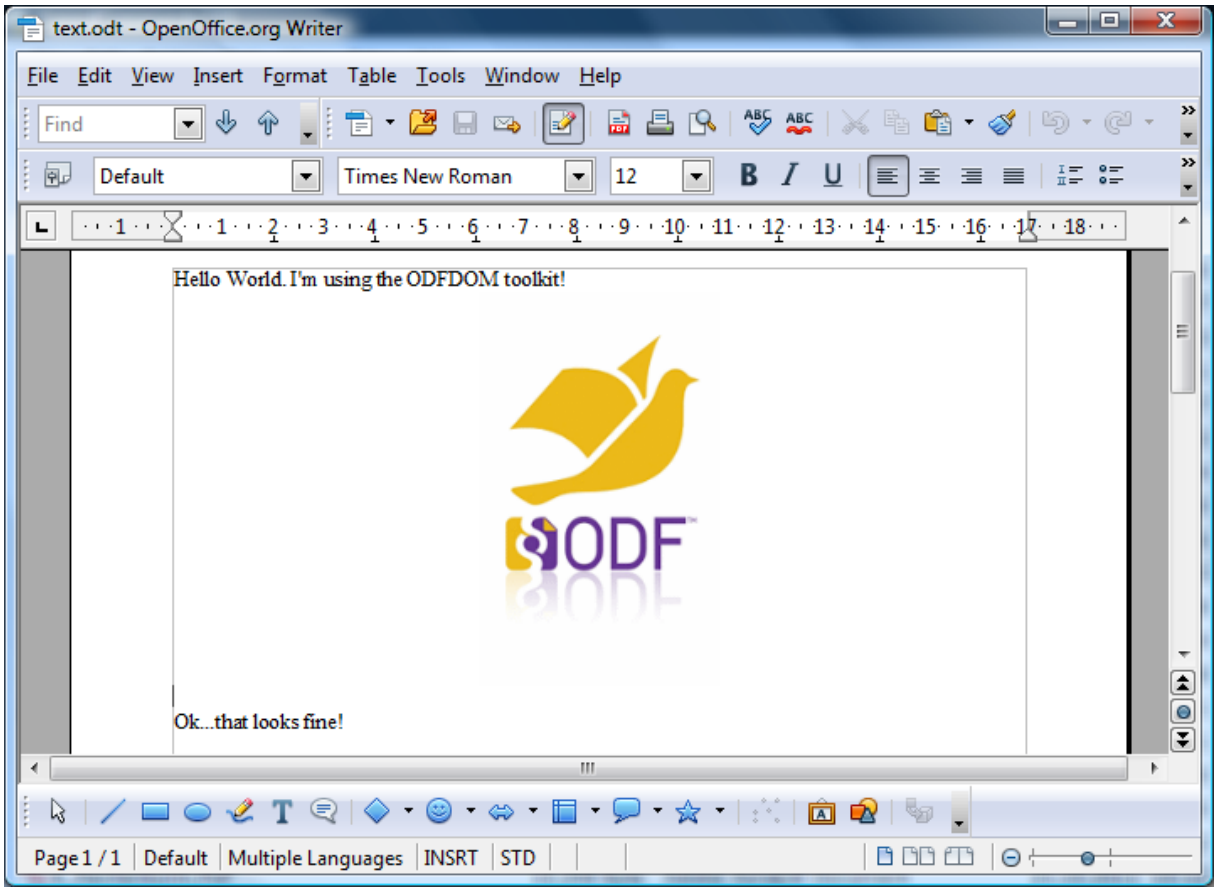


Figure 9: Result Example 01: "text.odt"

5.2 Modify a Text Document ("changeodt1.rxj")

Example 2 shows how to change an existing document.

First the existing file "text.odt" which was created in example 1 will be loaded. The loop creates three new paragraphs. They will be automatically added at the end of the existing document. The next step is to add some text. At last the modified document has to be saved.

```
-- Example 02 "changeodt1.rxj"
-- Open an existing text document and modify it

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
newOdt=clz~loadDocument("text.odt")

-- create some space
do 3
  newOdt~newParagraph()
end
```

```
-- add some text
newOdt~addText("I'm changing the text file!")

newOdt~save("text.odt")

::requires BSF.CLS
```

Figure 10 Example 02 Modifying a Text Document

The result of this example can be seen in figure 11.

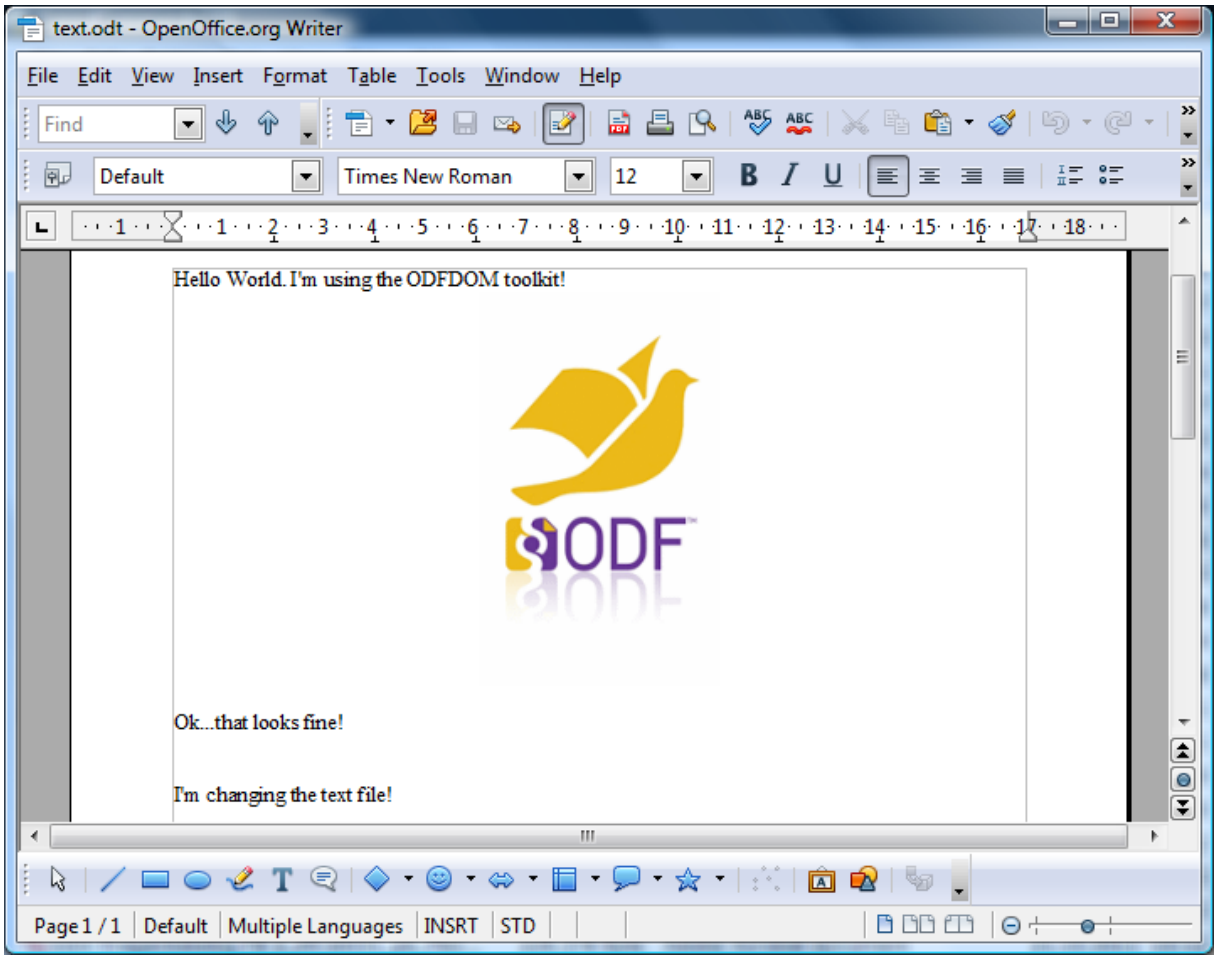


Figure 11: Result Example 02 "text.odt"

5.3 Save with a Different Name ("changeodt2.rxj")

This example is merely a variant of example 2. It is again an existing text document opened and edited. But this time it is saved under a different name. Therefore only the `"save"` method must be provided with a different filename than the original document, while the original document has not been changed.

The result looks as shown in the following figure.

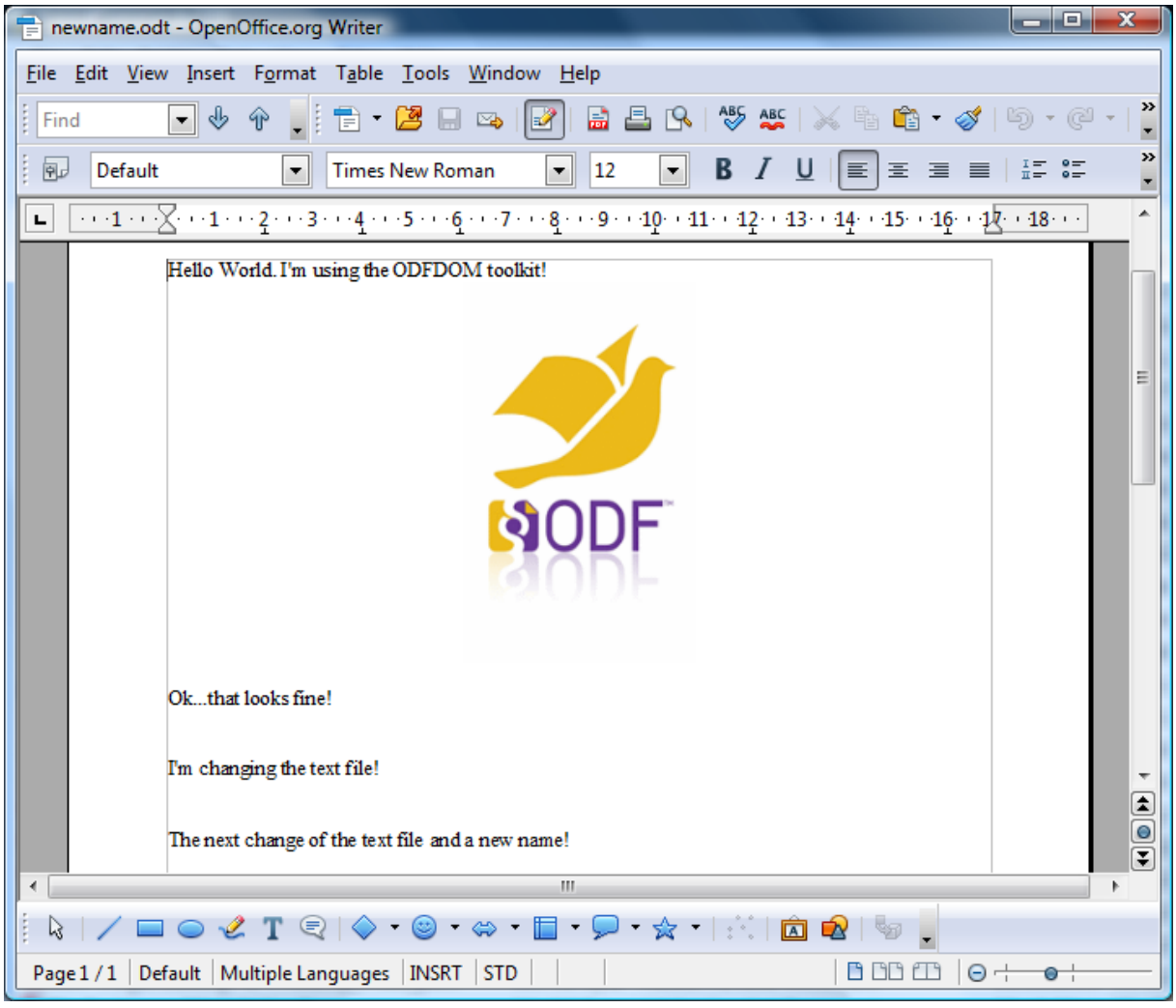


Figure 12: Result Example 03 `"newname.odt"`

```
-- Example 03 "changeodt2.rxj"
-- Open an existing text document and modify it

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
```

```
newOdt=clz~loadDocument("text.odt")

-- create some space
do 3
  newOdt~newParagraph()
end

-- add some text
newOdt~addText("The next change of the text file and a new name!")

-- save the document
newOdt~save("newname.odt")

::requires BSF.CLS
```

Figure 13 Example 03 Saving with a Different Name

5.4 Create a Circular Letter ("changeodt3.rxj")

Example 4 generates letters from an existing address or name list. The basis is a template letter "`importmaster.odt`", an existing text document.

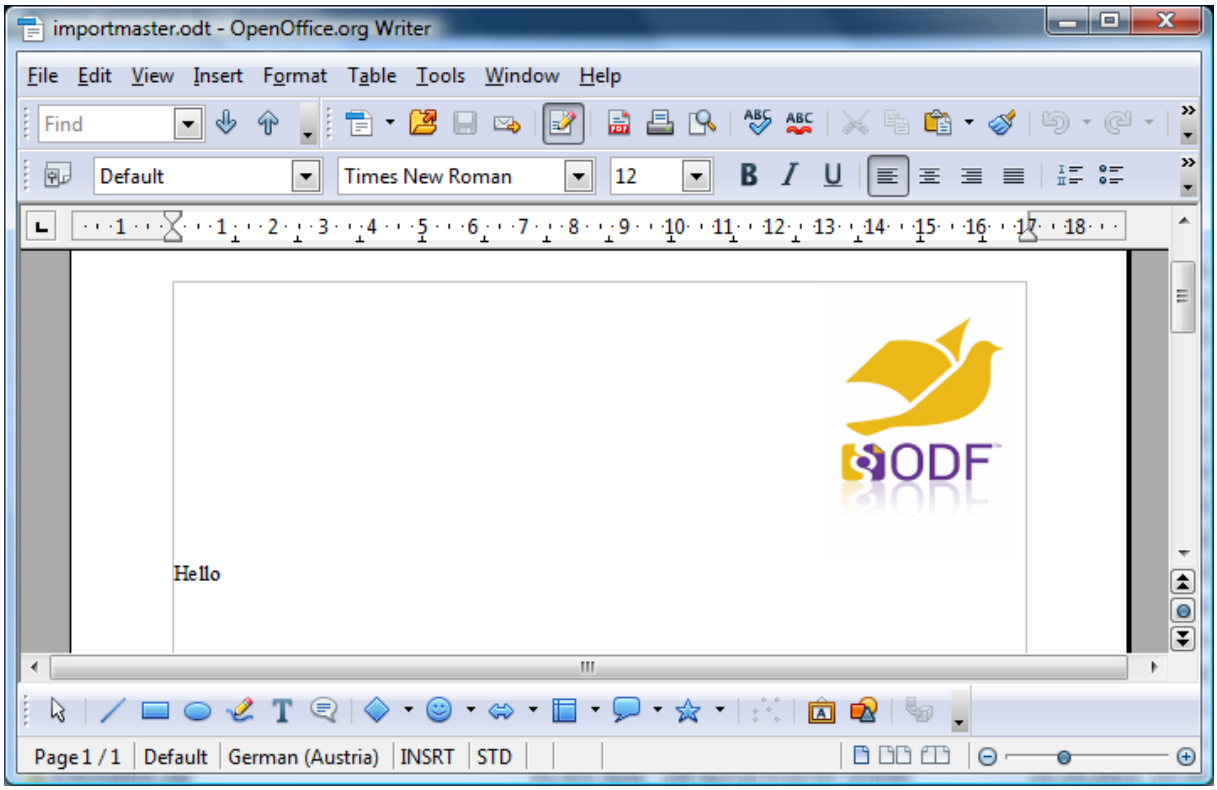


Figure 14: File "importmaster.odt"

Next, a class of the spreadsheet document "importtext.ods" (see Figure 15) will be created. The spreadsheet document fulfills the purpose of a database. The next step counts the number of entries in the database and stores the value in the counter variable "x". Since the row index in the first line starts with "0" the value of the counter variable has to be reduced by one so that the correct number of letters can be generated.

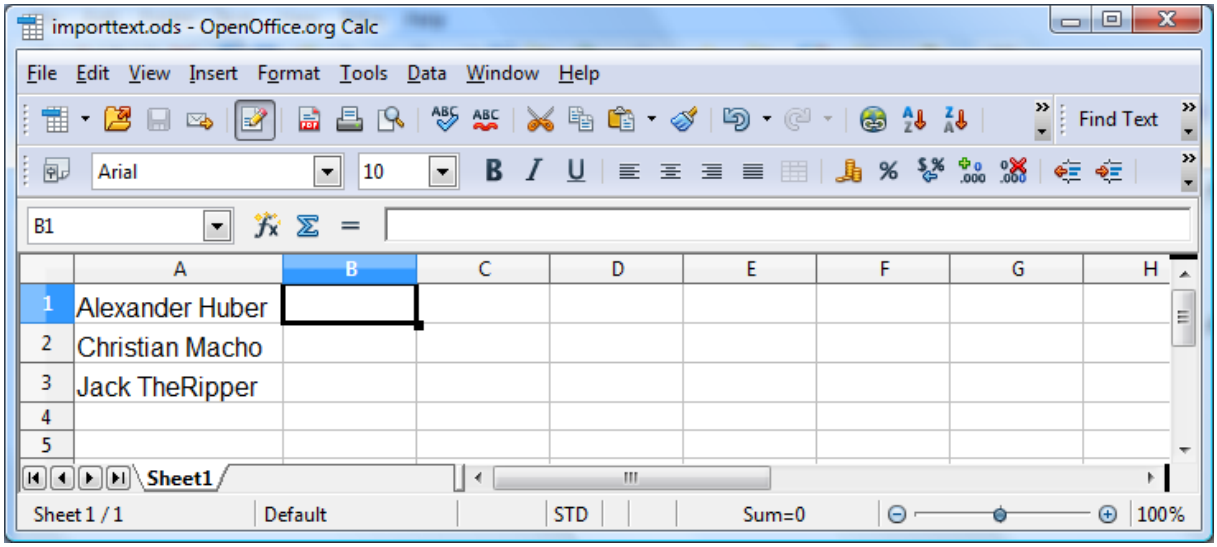


Figure 15: File "importtext.ods"

Next, the content of a cell is copied from the import file and inserted into the template letter. The row index of the corresponding cell is determined by the counter variable "x" .

Now the documents will be saved under a name with consecutive numbering. The numbering is based on the number of iterations of the loop. Whereby the iterations are controlled by the number of entries in the database.

```
-- Example 04 "changeodt3.rxj"
-- create a circular letter

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
newOdt=clz~loadDocument("importmaster.odt")

-- open the import file
clzs=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clzs~loadDocument("importtext.ods")

-- count the number of entries
odfTable=openOds~getTableList~get(0)
odfcollumn=odfTable~getColumnList()~get(0)
x=odfcollumn~getCellCount()
x=x-1
```

```
-- read input file, write new information and save the outputfile
-- create as many outputfiles as entries in the inputfile
do i=0 to x
  odfcell=odfTable~getCellByPosition(0,i)
  copycell=odfcell~getStringValue()
  newOdt=clz~loadDocument("importmaster.odt")
  newOdt~addText(copycell)
  newOdt~newParagraph
  newOdt~newParagraph
  newOdt~addText("I wish you a nice day!")
  newOdt~save("letter"i".odt")
end

::requires BSF.CLS
```

Figure 16 Example 04 Creating a Circular Letter

There have been three letters generated and the first one looks as follows:

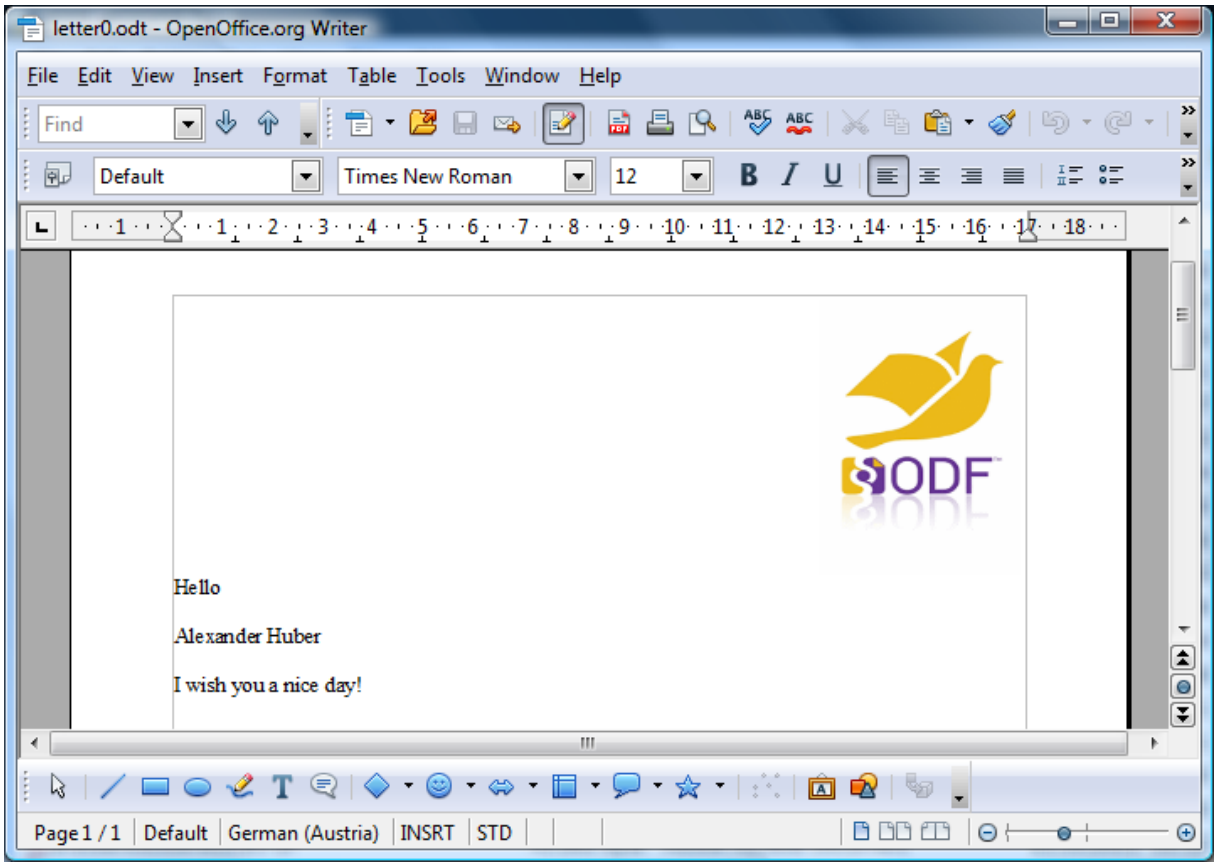


Figure 17: Result Example 04: The First Generated Letter

5.5 Create a Spreadsheet Document ("createods.rxj")

In the next example, a spreadsheet document is created. The first step is importing the corresponding class. For this example the class "OdfSpreadsheetDocument" is used. The next step will generate a blank spreadsheet. Up to here the procedure is similar to creating text documents. Now it is possible to set the focus on both, the individual tables as well as on the single cells. First, the method "table list" is needed, to call the first element (the first table) out of it. Please note that the index of the first table is "0"! To enter some data into the first cell in the first row ("A1"), the focus has to be set on it. Even here it has to be kept in mind, that both the index of the first column and of the first row is "0". The next step writes a string into the cell. The next two statement blocks show that the first parameter of the method "getCellByPosition" shows the row and the second the column.

To change the background colour of a cell, the method "setCellBackgroundColor" has to be used. The desired color is defined by a six-digit hexadecimal code which starts with a hash. The code is a combination of 3 bytes in hexadecimal notation (two digits = one byte), representing the intensity of the colors red, green and blue [WeCo01]. The color code "#99CCFF" which is used in Example 05 results in a light blue, which can be seen in figure 19 cell "C1".

Saving the document is analogous to a text document. In the argument the desired file name is specified.

```
-- Example 05 "createods.rxj"
-- create a spreadsheet document

-- create a new spreadsheet document
clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
outputOds=clz~newSpreadsheetDocument

-- get the first table
odfTable=outputOds~getTableList~get(0)

-- get cell "A1" and add some text
odfcell=odfTable~getCellByPosition(0,0)
odfcell~setStringValue('Hello World')
```

```
-- get cell "A3"
odfcell=odfTable~getCellByPosition(0,2)
odfcell~setStringValue('Hello Row')

-- get cell "C1", add text and change backgrund color
odfcell=odfTable~getCellByPosition(2,0)
odfcell~setCellBackgroundColor('#99CCFF')
odfcell~setStringValue('Hello Collum')

-- save the document
outputOds~save("spreadsheet.ods")

::requires BSF.CLS
```

Figure 18 Example 05 Create a Spreadsheet Document

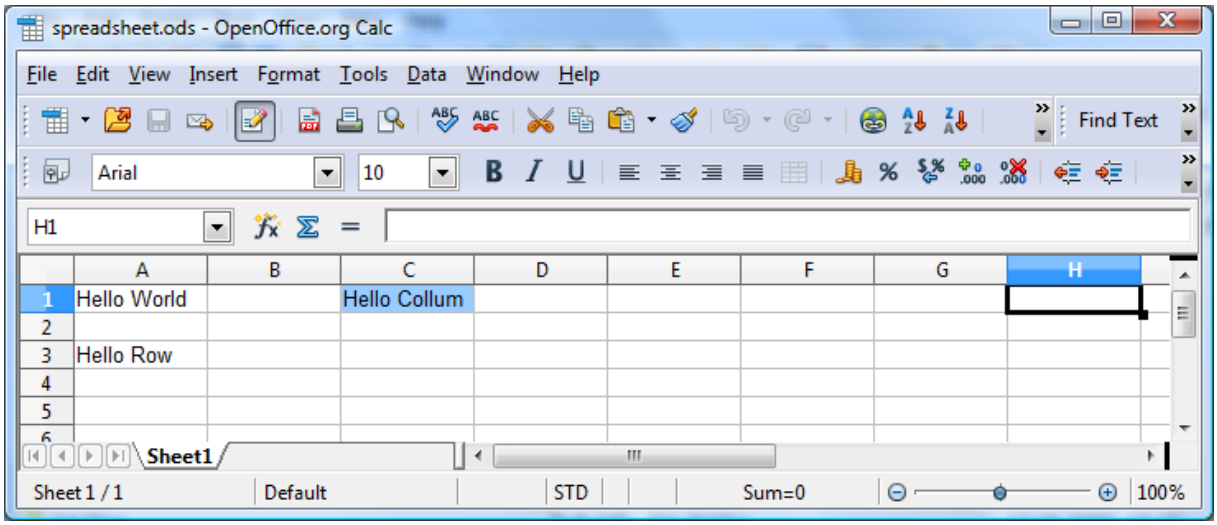


Figure 19: Result Example 05 "spreadsheet.ods"

5.6 Modify a Spreadsheet Document ("changeods1.rxj")

In this example it will be shown how an existing spreadsheet document can be accessed and modified.

The first step is to import the appropriate class. Now the document, which had been created in the last example, has to be loaded. In the next step the first table should be changed. Two

lines will be inserted between the first and the second line. The next line of code changes the name of the table to "Sepperl".

The further step writes a new value into cell "A1". Thus deletes the previously existing value. Furthermore, cells will be connected together. To do this, the cell area has to be defined. The arguments of the method "getCellRangeByPosition" are, first, the table name, furthermore, the upper left and lower right corner cell, of the area. Using the method "merge" combines the cells together. As a result they behave as if they were just one single cell. Here the range is limited to one line. A further change is the row height. In this example, the height of the third row gets changed. Therefore the third row will be called from the line index and the existing height will be delivered by the method "getHeight". The desired change of the value will be added to the height and then assigned to the row height. At the end the modified document has to be saved.

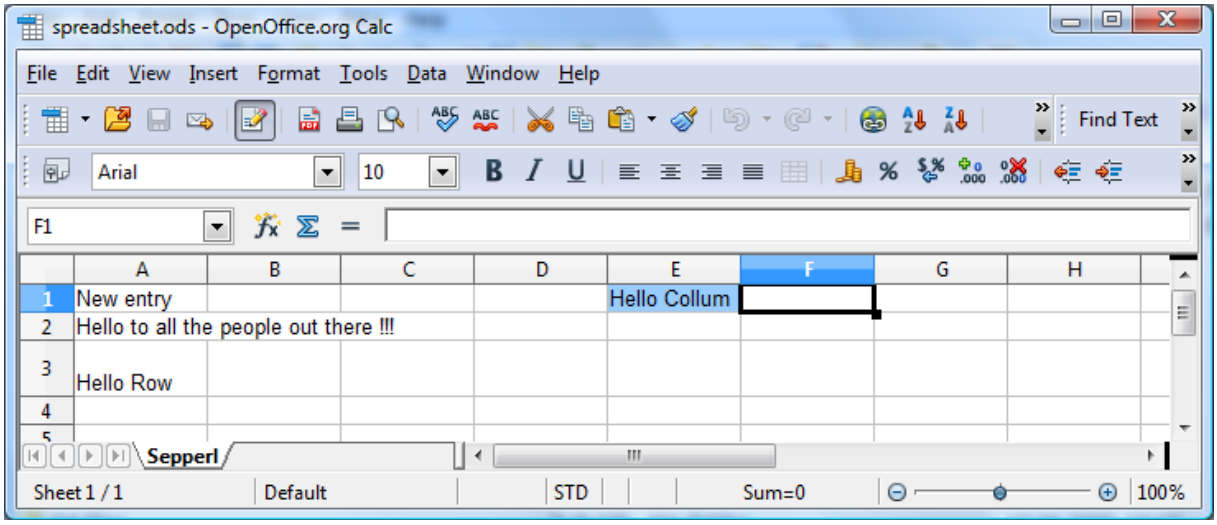


Figure 20: Result Example 06 "spreadsheet.ods"

```
-- Example 06 "changeods1.rxj"
-- change an existing spreadsheet

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("spreadsheet.ods")

odfTable=openOds~getTableList~get(0)

-- insert new columns
odfTable~insertColumnsBefore(1,2)
odfTable~setTableName("Sepperl")
```

```
-- change entry of cell "A1"
odfcell=odfTable~getCellByPosition(0,0)
odfcell~setStringValue('New entry')

-- merge some cells
cellrange=odfTable~getCellRangeByPosition("Sepperl.A2","C2")
cellrange~merge
odfcell=odfTable~getCellByPosition(0,1)
odfcell~setStringValue('Hello to all the people out there !!!')

-- change row height
odfRow=odfTable~getRowByIndex(2)
h=odfRow~getHeight
odfRow~setHeight(h+2,0)

-- save the document
openOds~save("spreadsheet.ods")

::requires BSF.CLS
```

Figure 21 Example 06 Modifying a Spreadsheet Document

5.7 Modify a Spreadsheet Part Two ("changeods2.rxj")

Example 7 shows again the loading and changing of an existing spreadsheet document. The content of a cell will be copied into a range of cells.

This is, as mentioned in Example 6, from the table list the first table and the desired cell. Now the content of the cell is written into the variable `"copycell"`. In the next step, a cell range is defined. In contrast to Example 6 the cell area is not defined by using the cell address instead of using the rows and columns index. The cell area is now addressed with the upper left corner cell. The content of the variable `"copy cell"` is written into the cell area, the subsequent formatting of course, refers also to the entire range of cells.

The parameter of the method `"setHorizontalAlignment"`can be `"center"`, `"end"`, `"justify"`, `"left"`, `"right"`, or `"start"`. The parameter of the method `"setVerticallignment"` can be `"auto"`, `"automatic"`, `"baseline"`, `"bottom"`, `"middle"`, or `"top"`.

In contrast to chapter 5.5 the cell background color is now defind as a light grey.

In the next step, another table will be created. This table gets inserted at the end of the document. Also the name of the sheet is being changed. Now, the content of the variable "copycell" is written into cell "B2".

```
-- Example 07 "changeods2.rxj"
-- change an existing spreadsheet

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
openOds=clz~loadDocument("spreadsheet.ods")

-- copy content of a cell
odfTable=openOds~getTableList~get(0)
odfcell=odfTable~getCellByPosition(0,1)
copycell=odfcell~getStringValue()

-- paste the content into a cell range and format it
cellrange=odfTable~getCellRangeByPosition(3,4,7,8)
cellrange~merge
odfcell=odfTable~getCellByPosition(3,4)
odfcell~setStringValue(copycell)
odfcell~setHorizontalAlignment("center")
odfcell~setVerticalAlignment("middle")
odfcell~setCellBackgroundColor('#f5f5f5')

-- create a new table and paste content
odfTable~newTable(openOds)
odfTable=openOds~getTableList~get(1)
odfTable~setTableName("R2D2")
odfcell=odfTable~getCellByPosition(1,1)
odfcell~setStringValue(copycell)

-- save the document
openOds~save("spreadsheet2.ods")

::requires BSF.CLS
```

Figure 22 Example 07 Modifying a Spreadsheet Part Two

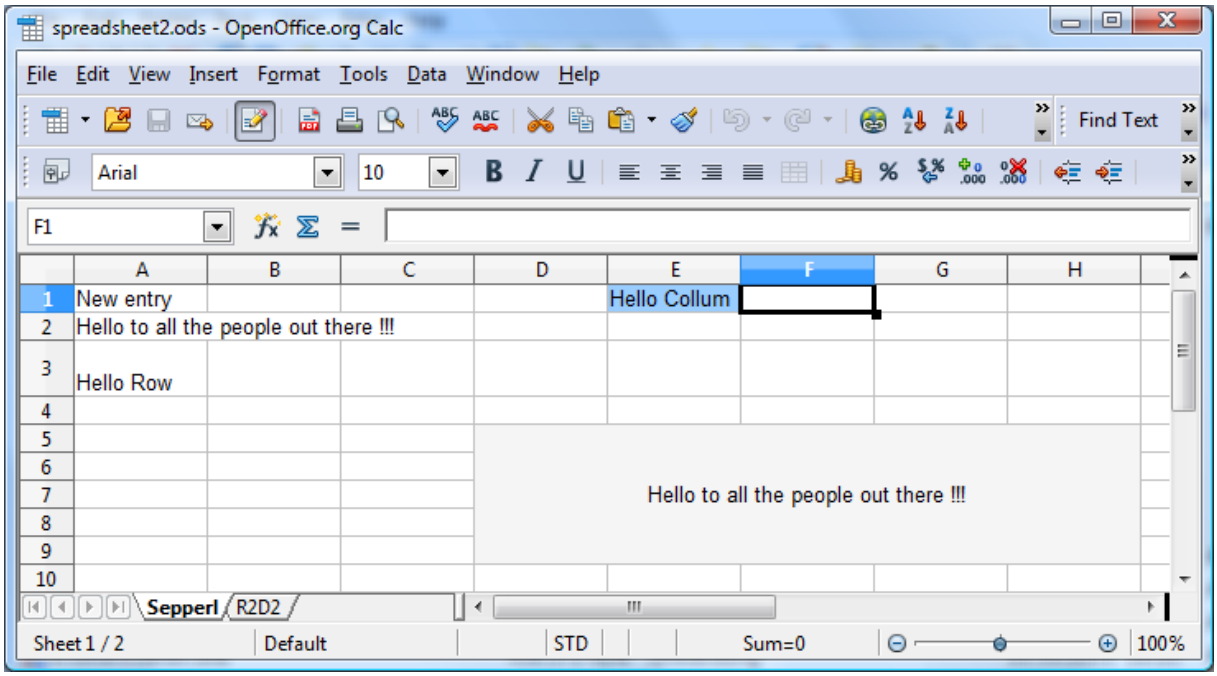


Figure 23: Result Example 07 "spreadsheet2.ods", Sheet "Sepperl"

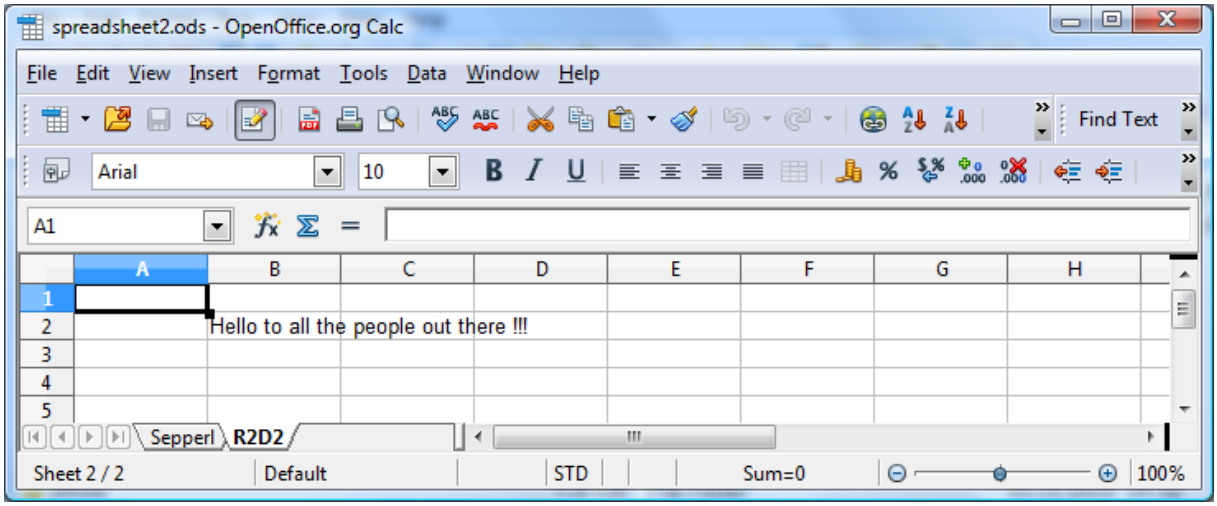


Figure 24: Result Example 07 "spreadsheet2.ods", Sheet "R2D2"

5.8 Create a Presentation Document ("createodp.rxj")

Example 8 shows how a presentation is created and gets filled with content.

The first step corresponds to the already known procedure from the previous examples. The appropriate class for a presentation document has to be loaded. The method

"newPresentationDocument" creates a blank presentation. This includes a blank page. To be able to access the elements, the existing content is called. At the time of the production of the document a blank page was generated and this is the actual content of the document. The first element (the first page) has to be called. The next step is to insert text into the page. Text can only be inserted into a frame. Therefore a frame has to be created with "newDrawFrameElement". X and Y coordinates get defined for the frame with the next two lines of code. The coordinate origin starts at the upper left corner of the page. Furthermore, the frame also requires a height and a width value. Next, a text box element is inserted into the frame and into this a text paragraph. Now it is possible to write the desired information into the text box. The final task is to save the document.

```
-- Example 08 "createodp.rxj"
-- create a presentation document

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
outputOdp=clz~newPresentationDocument
officePresentation=outputOdp~getContentRoot()

-- get the existing first slide
page=officePresentation~getFirstChild()

-- create a frame and define the orientation
frame=page~newDrawFrameElement()
frame~setSvgXAttribute("12cm")
frame~setSvgYAttribute("3cm")
frame~setSvgWidthAttribute("4cm")
frame~setSvgHeightAttribute("3.5cm")

-- include a textbox into the frame
textBox=frame~newDrawTextBoxElement()

-- and add some text
para=textBox~newTextPElement()
para~addContent(' Hello world!')

-- save the document
outputOdp~save("presentation.odp")

::requires BSF.CLS
```

Figure 25 Example 08 Creating a Presentation Document

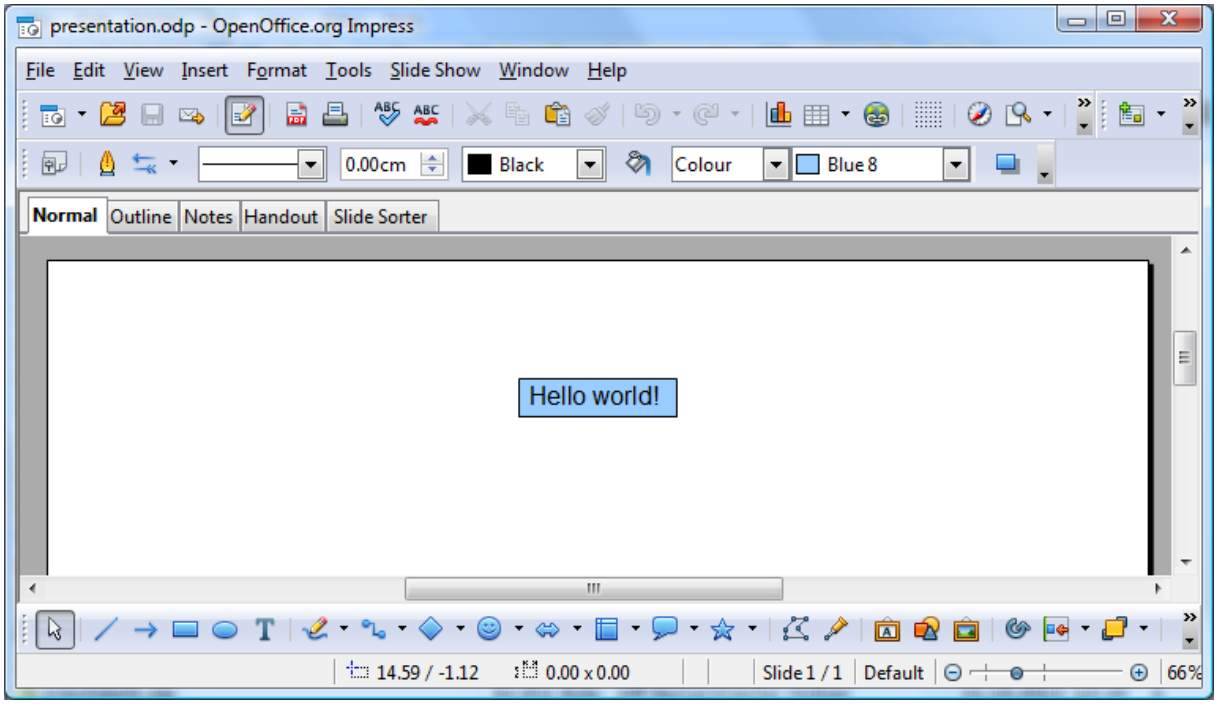


Figure 26: Result Example 08 "presentation.odp"

5.9 Modify a Presentation Document ("changeodp1.rxj")

The next example shows how to change an existing presentation document.

Therefore an existing presentation will be loaded and an empty page will be inserted. This page is automatically appended to the end of the existing document. Now, as explained in Example 8, a frame is defined with appropriate size and position. Into this frame an image is inserted.

```
-- Example 09 "changeodp1.rxj"
-- change an existing presentation document

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
openOdp=clz~loadDocument("presentation.odp")
officePresentation=openOdp~getContentRoot()

-- create a new slide
page=officePresentation~newDrawPageElement("")

-- create a frame and define the orientation
frame=page~newDrawFrameElement()
```

```
frame~setSvgXAttribute("12cm")
frame~setSvgYAttribute("5cm")
frame~setSvgWidthAttribute("5cm")
frame~setSvgHeightAttribute("7cm")

-- include an image element into the frame
image=frame~newDrawImageElement()
image~newImage(.bsf~new("java.net.URI","odfdomlogo.gif"))

-- save the document
openOdp~save("presentation.odp")

::requires BSF.CLS
```

Figure 27 Example 09 Modifying a Presentation Document

As a result the document consists of two slides.

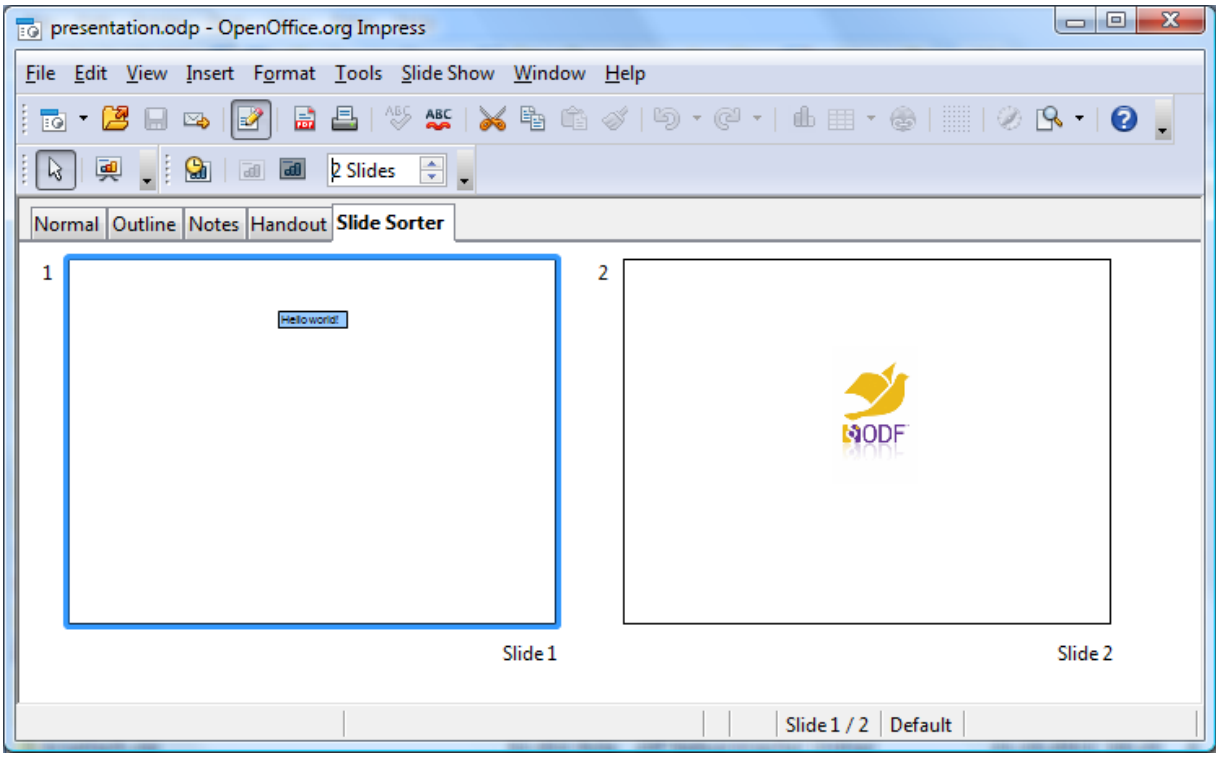


Figure 28: Result Example 09 "presentation.odp"

5.10 Modify and add Parts of a Presentation ("changeodp2.rxj")

Example 10 shows how to insert, delete and modify some elements into an existing presentation.

In the first step the required document will be called and the image, which was inserted in Example 9, will be removed. The next step is to address the last page. On this page the first element is needed. This is the inserted frame. The frame itself should not be removed, only the image inside the frame. The first item in the frame is the image, which should be removed. With "`removeChild`" the image gets removed. The frame is still available. Now a text box is inserted into the existing frame. For this, the start position (upper left corner) and the width are determined. The text box is also filled with content.

The loop in the next step creates three additional pages, each of them contains a frame with an inserted text box. The loop counter-variable is used in the text as the numbering. It starts to count with number 2. So the first page which is inserted, gets: "`Page 2`" as text. But this page is page three. The first page was produced in Example 8. The second page was added in Example 9 and changed in the current example. The slides 3-5 were added in the last step. But the pasted text on these three pages shows "`Page: 2-4`". To correct this intentional "error" the second page gets moved to the last position. It has to be noted that the arguments of the method "`moveSlide`" start with index "`0`" and therefore the second page has to be addressed with "`1`". The final step is, like all other examples, to save the document.

```
-- Example 10 "changeodp2.rxj"
-- remove and add elements to a presentation document

clz=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
openOdp=clz~loadDocument("presentation.odp")
officePresentation=openOdp~getContentRoot()

-- remove the image from the last page
page=officePresentation~getLastChild()
frame=page~getFirstChild()
image=frame~getFirstChild()
frame~removeChild(image)

-- include a textbox into the existing frame
textBox=frame~newDrawTextBoxElement()
frame~setSvgXAttribute("11cm")
frame~setSvgYAttribute("8cm")
frame~setSvgWidthAttribute("6.5cm")
para=textBox~newTextPElement()
para~addContent('This is the last page!')

-- generate three pages with text box and numbering
do i=2 to 4
  page=officePresentation~newDrawPageElement("")
```

```
   frame=page~newDrawFrameElement()
   frame~setSvgXAttribute("12.5cm")
   frame~setSvgYAttribute("8cm")
   frame~setSvgWidthAttribute("3cm")
   frame~setSvgHeightAttribute("3.5cm")
   textBox=frame~newDrawTextBoxElement()
   para=textBox~newTextPElement()
   para~addContent('Page: 'i)
end

-- move the 2nd page to the end
openOdp~moveSlide(1,5)

-- save the document
openOdp~save("presentation2.odp")

::requires BSF.CLS
```

Figure 29 Example 10 Modifyind and Adding Parts of a Presentation

Now the presentation consists of 5 slides and the result is stored in "presentation2.ods".

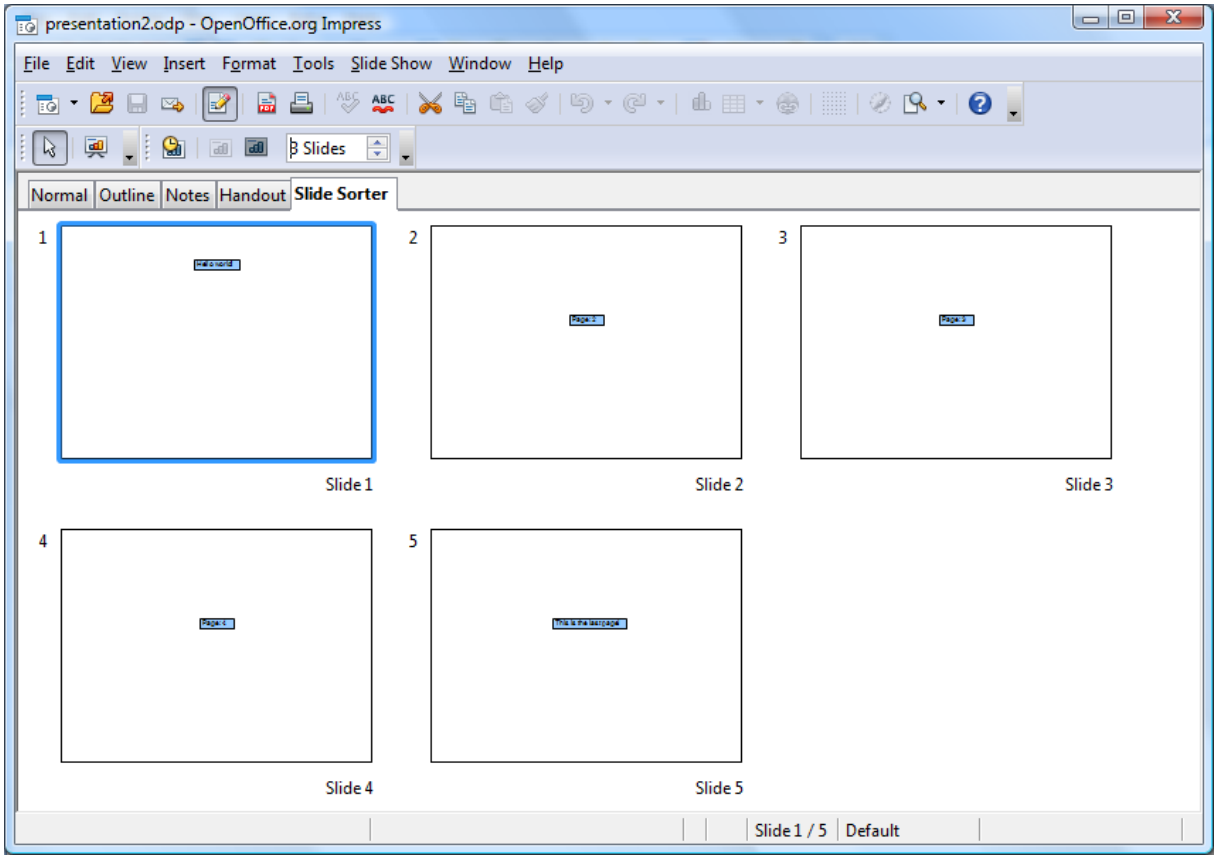


Figure 30: Result Example 10 "presentation2.odp"

6 Conclusion

The toolkit ODFDOM is a very powerful API to handle ODF documents. Due to the rather sparse documentation, it was a great challenge for the Author to familiarize himself with the different methods. In extending the work, the great potential that lies within this tool, slowly came out.

Probably the documentation problematic may change in the future. The ODF toolkit has been welcomed in the incubator by the Apache Software Foundation [ApSo02]. This is a reasonable chance for the future, to have more resources and opportunities available, to develop the tools and the corresponding documentation. This may help future developers, breaking new in this matter, with their work.

Hopefully, this work contributes a small piece to this.

7 List of references

[ApSo01] Apache Software Fundation, http://www.apache.org, as of 2011-06-12.

[ApSo02] Apache ODF Toolkit (Incubating), http://incubator.apache.org/odftoolkit/index.html, as of 2011-10-12

[BsRx01] The Bean Scripting Framework for ooRexx, http://wi.wu-wien.ac.at:8002/rgf/wu/lehre/autojava/material/foils/, as of 2011-11-04.

[JaAr01] Jar Java Archive File Format, http://en.wikipedia.org/wiki/JAR_%28file_format%29, as of 2011-09-20.

[JaSd01] Java™ 2 SDK, Standard Edition Documentation, http://download.oracle.com/javase/1.4.2/docs/index.html, as of 2011-06-25.

[JaUr01] java.net.URI, http://download.oracle.com/javase/1.4.2/docs/api/java/net/URI.html, as of 2011-07-20.

[OaSi01] OASIS standards organization, http://www.oasis-open.org/, as of 2011-09-30.

[OdAp01] Package org.odftoolkit.odfdom, http://odfdom.odftoolkit.org/0.8.7/odfdom/apidocs/index.html, as of 2011-08-02

[OdDm01] ODFDOM - the OpenDocument API, http://odftoolkit.org/projects/odfdom/pages/Home, as of 2011-06-23.

[OdDm02] ODFDOM Layer, http://odftoolkit.org/projects/odfdom/pages/ProjectOverview, as of 2011-08-12.

[OdDm03] ODFDOM API, http://odftoolkit.org/projects/odfdom/pages/Layers, as of 2011-08-16.

[OdOv01] odftoolkit.org ODFDOM Project Overviev, http://odftoolkit.org/projects/odfdom/pages/ProjectOverview, as of 2011-06-10.

[OdSp01] ODF specification, http://docs.oasis-open.org/office/v1.2/OpenDocument-v1.2-part3.html, as of 2011-09-12.

[SuMi01] Sun Microsystems, http://www.oracle.com/us/sun/index.htm, as of 2011-09-30.

[ViEd01] Vim the editor, http://www.vim.org/download.php, as of 2011-06-13.

[WeCo01] Web colors, http://en.wikipedia.org/wiki/Web_colors, as of 2011-11-06

8 Download Links

- **OOo** – Open Office.org, http://download.openoffice.org, as of 2011-11-04.
- **J2SE** – Sun Java2 Standard Edition, http://www.java.com, as of 2011-11-04.
- **OORexx** – OpenObjectRexx, http://www.oorexx.org/download.html, as of 2011-11-04.
- **BSF4ooRexx** – Bean Scripting Framework for ooRexx, http://sourceforge.net/projects/bsf4oorexx/files/, as of 2011-11-04.
- **ODF** - Java odftoolkit 0.8.6, http://odftoolkit.org/projects/odfdom/downloads/download/previous-versions/releases/odfdom-0.8.6-binaries.zip, as of 2011-11-04.
- **Xerces** - Java XSLT processor, http://tweedo.com/mirror/apache//xerces/j/Xerces-J-src.2.11.0-xml-schema-1.1-beta.zip, as of 2011-11-04.

9 Statement

I hereby declare that I have made this work independently and without use of other than the specified resources. All parts that were taken literally or in spirit from published or unpublished writings, are identified as such. The work has not been presented as part of another test in the same or similar form or extracts of it.

Baden, 2011-11-03

Ing. Günter Mayer