WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor Thesis

Title of the Bachelor Thesis:	BSF4ooRexx: Scripting the ODF Toolkit
Author (last name, first name):	Martin, Christoph
Student ID number:	0854504
Degree program:	Business, Economics and Social Sciences Major in Information Systems
Examiner (degree, first name, last name):	ao. Univ.Prof. Mag. Dr. Rony G. Flatscher

I hereby declare that

1. I have written this Bachelor thesis independently and without the aid of unfair or unauthorized resources. Whenever content was taken directly or indirectly from other sources, this has been indicated and the source referenced.

2. this Bachelor thesis has neither previously been presented for assessment, nor has it been published.

3. this Bachelor thesis is identical with the assessed thesis and the thesis which has been submitted in electronic form.

4. (only applicable if the thesis was written by more than one author): this Bachelor thesis was written together with (first name, surname). The individual contributions of each writer as well as the co-written passages have been indicated.

Date _____

_____
Signature

Danksagung

An dieser Stelle möchte ich mich herzlich bei meinen Eltern bedanken, deren Hilfe mir die erfolgreiche Gestaltung meines Studiums erleichtert hat.

Außerdem gilt mein Dank Herrn Dr. Flatscher, der mich während der Erstellung dieser Arbeit stets mit hilfreichen fachlichen Hinweisen unterstützt hat.

Contents

Abstract 5

List of figures 6

List of abbreviations 8

1 Introduction 10

2 Involved Components 12
2.1 Java 12
2.2 ooRexx 13
2.2.1 REXX 13
2.2.2 Object orientation 14
2.3 BSF4ooRexx 14
2.4 Apache ODF Toolkit 15
2.5 Apache Xerces 16
2.6 Apache OpenOffice 16
2.6.1 Writer 17
2.6.2 Calc 17
2.6.3 Impress 17

3 Installation Guide 18
3.1 Java Runtime 18
3.2 ooRexx 18
3.3 BSF4ooRexx 18
3.4 Apache Xerces 19
3.5 Apache ODF Toolkit 20
3.6 Apache OpenOffice 20

4 OpenDocument Format 21
4.1 Introduction and history 21
4.2 Inside an OpenDocument Format File 22
4.2.1 mimetype 22
4.2.2 META-INF/manifest.xml 23

4.2.3 meta.xml 23
4.2.4 settings.xml 24
4.2.5 styles.xml 25
4.2.6 content.xml 26

5 Apache ODF Toolkit 27
5.1 Introduction and history 27
5.2 Components 28
5.2.1 ODF Validator 28
5.2.2 ODF XSLT Runner 29
5.2.3 Simple API 29
5.3 ODFDOM 30
5.3.1 ODFDOM Layers 30

6 Nutshell Examples 33
6.1 ODT / Writer 34
6.1.1 Nutshell Example ODT 1 34
6.1.2 Nutshell Example ODT 2 38
6.1.3 Nutshell Example ODT 3 42
6.2 ODS / Calc 45
6.2.1 Nutshell Example ODS 1 45
6.2.2 Nutshell Example ODS 2 48
6.2.3 Nutshell Example ODS 3 51
6.3 ODP / Impress 55
6.3.1 Nutshell Example ODP 1 55
6.3.2 Nutshell Example ODP 2 59
6.3.3 Nutshell Example ODP 3 63

7 Conclusion and Outlook 66

8 References 67

Abstract

This paper illustrates the concepts required for automatically creating and manipulating ODF documents using the Apache ODF Toolkit and the easy-to-learn high-level programming language ooRexx. It describes the involved components and the OpenDocument Format Standard. The theoretical concepts are followed by short code examples, which exemplify the possibilities to create and manipulate text, spreadsheet and presentation documents.

Keywords:

AOO | Apache OpenOffice | BSF4ooRexx | ODF | ODFDOM | OOo OoRexx | OpenOffice | OpenOffice.org | Scripting

List of figures

Figure 1 - helloworld.rexx....13

Figure 2 - Editing the environment variable....19

Figure 3 - Structure of an ODF file....22

Figure 4 - MIME type of OpenOffice text document....23

Figure 5 - Excerpt of the manifest.xml file....23

Figure 6 - Excerpt of the meta.xml file....24

Figure 7 - Excerpt of the settings.xml file....24

Figure 8 - Condensed excerpt of the content.xml file....26

Figure 9 - The ODFDOM Layers [ADOP12]....30

Figure 10 - Nutshell Example ODT 1 Code ooRexx....34

Figure 11 - Nutshell Example ODT 1 Code Java....35

Figure 12 - Nutshell Example ODT 1 Resulting Document....37

Figure 13 - Nutshell Example ODT 2 Code....38

Figure 14 - Nutshell Example ODT 2 Resulting Document....41

Figure 15 - Nutshell Example ODT 3 Code....42

Figure 16 - Nutshell Example ODT 3 Resulting Document....44

Figure 17 - Nutshell Example ODS 1 Code....45

Figure 18 - Nutshell Example ODS 1 Resulting Document....47

Figure 19 - Nutshell Example ODS 2 Code....48

Figure 20 - Nutshell Example ODS 2 Resulting Document....50

Figure 21 - Nutshell Example ODS 3 Code 52

Figure 22 - Nutshell Example ODS 3 Resulting Document.....54

Figure 23 - Nutshell Example ODP 1 Code 56

Figure 24 - Nutshell Example ODP 1 Resulting Document.....58

Figure 25 - Nutshell Example ODP 2 Code 60

Figure 26 - Nutshell Example ODP 2 Resulting Document.....62

Figure 27 - Nutshell Example ODP 3 Code 63

Figure 28 - Nutshell Example ODP 3 Resulting Document.....65

List of abbreviations

API	Application Programming Interface
ASF	Apache Software Foundation
BSF4ooRexx	Bean Scripting Framework for ooRexx
DOM	Document Object Model
IEC	International Electrotechnical Commission
IS	International Standard
ISO	International Organization for Standardization
JDK	Java Development Kit
JRE	Java Runtime Environment
JTC	Joint Technical Committee
MIME	Multipurpose Internet Mail Extensions
OASIS	Organization for the Advancement of Structured Information Standards
ODF	Open Document Format
ODFDOM	A free OpenDocument Format library
ODP	OpenDocument Format Presentation Document
ODS	OpenDocument Format Spreadsheet Document
ODT	OpenDocument Format Text Document
ooRexx	Open Object Rexx
PDF	Portable Document Format

REXX	Restructured Extended Executor
RexxLA	Rexx Language Association
RGB	Additive color model (red green blue)
SAX	Simple API for XML
SWF	Shockwave Flash
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

1 Introduction

Office software is one of the most commonly used types of computer programs. With a growing amount of knowledge workers employed in the service sector in the developed world, software to help these workers boosting their productivity is essential. [WOff12]

Office software is usually distributed as a collection of different programs with a similar look and feel and a consistent user interface, a so called office suite. These office suites normally consist of at least a word processor, a spreadsheet software and a presentation program. Additional programs of the bundle can handle graphics, formulas, databases and so forth.

The one particular company which immediately comes to mind is certainly Microsoft, with its Microsoft Office Suite. Disadvantages of such a proprietary solution are restricted rights due to licenses, obstacles for customization and, of course, the created documents can only be opened and altered with the corresponding software, although there are some exceptions to this rule.

One popular alternative is the OpenDocument Format (ODF), an open standard for office documents. “ODF is defined via an open and transparent process at OASIS (Organization for the Advancement of Structured Information Standards) and has been approved unanimously by the Joint Technical Committee 1 (JTC1) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) as an International Standard (IS) in May 2006. It is available for implementation and use free of any licensing, royalty payments, or other restrictions.” [OASI06, p. 5]

This paper illustrates the concepts required for automatically creating and manipulating ODF documents using the ODFDOM API, which is part of the Apache ODF Toolkit. The Apache ODF Toolkit is a set of Java modules for creating, scanning and manipulating ODF documents.

The reference implementation of ODFDOM is written in Java. Due to the complex syntax of Java the human-friendly and easy to learn programming lan-

guage Open Object Rexx (ooRexx) is used in combination with Java in this paper.

Numerous components have already been mentioned above. These components and the way they are working together to achieve the desired objective are described in chapter 2.

After these theoretical concepts chapter 3 guides the reader through the required installation processes. It shows step by step what has to be done in order to create and manipulate ODF documents on your own Windows 7 machine.

In chapter 4 the OpenDocument Format Standard is described in greater detail. A look into the inside of an ODF document is provided.

In the fifth Chapter it is explained what the ODF Toolkit is and what it can be used for. Furthermore the individual components of the Toolkit are listed and briefly described.

Chapter 6 illustrates how the above-mentioned concepts and components can be combined to create and manipulate ODF documents with nine short code examples, so called nutshell examples.

2 Involved Components

Java will be camouflaged as ooRexx using the Bean Scripting Framework for ooRexx (BSF4ooRexx) for the purpose of using the Java reference implementation of ODFDOM, which is a part of the ODF Toolkit. Using the ODFDOM API OpenDocument Format (ODF) documents will be created and manipulated in so called nutshell examples in chapter 6. The free office document suite Apache OpenOffice will then be used to view these Examples.

In order to unfold these rather complex undertaking all mentioned components are described in this chapter.

2.1 Java

“The Java programming language is a general-purpose, concurrent, class-based, object-oriented language.” [Gosl05, p. 35] Named after Java coffee and appeared in 1995, it is now one of the most popular programming languages worldwide at the time of writing.

One of the main advantages of Java is the vast amount of already available code snippets, which can be combined to efficiently create new code instead of writing it all over again.

The syntax of Java is derived largely from C++, another widely-used programming language. This syntax is rather complex and cannot be described as human-friendly, especially for people new to programming concepts. [WJav12]

The reference implementation of ODFDOM, which is used to manipulate ODF documents in this paper, is written in Java. Due to the mentioned disadvantages of the Java syntax, the programming language ooRexx will be used to utilize Java. More information on this procedure follows in chapter 2.3.

2.2 ooRexx

Open Object Rexx (ooRexx) is an enhancement of classic REXX (Restructured Extended Executor). REXX itself is briefly described in 2.2.1 whereas the object orientation concepts provided by ooRexx are illustrated in 2.2.2.

2.2.1 REXX

REXX is an easy-to-learn high-level programming language developed at IBM by Mike Cowlishaw. It was designed to ease the process of programming as one can fathom with the following list of features REXX provides (excerpt):

- Simple and human-friendly syntax
- Just two dozen instructions
- No case-sensitivity
- Dynamic data typing (no declarations required)
- No reserved keywords
- Powerful built-in functions for string- and word-processing [WREX12]

These are just some of the most important key features of the programming language REXX. To further illustrate its simplicity the classic “Hello, world!” program is implemented in REXX in Figure 1.

```
Say "Hello, world!"
```

Figure 1 - helloworld.rexx

Please remember that the instruction `Say` could also be written with capital letters (`SAY`) or even as a mix of capital and non-capital letters (`SaY`) due to the case-insensitivity.

Compared to implementations in other programming languages like Java this is an incredibly simple “Hello, world!” program.

2.2.2 Object orientation

Open Object Rexx (ooRexx) adds object orientation to REXX. It is an Open Source Project managed by the Rexx Language Association (RexxLA). The RexxLA puts the advantages of object orientation as follows:

“Open Object Rexx includes features typical of an object-oriented language, such as subclassing, polymorphism, and data encapsulation. It is an extension of the classic Rexx language, which has been expanded to include classes (a base set of classes is supplied), objects, and methods. These extensions do not replace classic Rexx functions or preclude the development or running of classic Rexx programs.” [Rexx09]

“Object-oriented programmers solve problems by identifying and classifying objects related to the problem. Then they determine what actions or behaviors are required of those objects. Finally, they write the instructions to generate the classes, create the objects, and implement the actions. The main program consists of instructions that send messages to objects.” [Rexx09]

“Other advantages often associated with object-oriented technology are:

- Simplified design through modeling with objects
- Greater code reuse
- Rapid prototyping
- Higher quality of proven components
- Easier and reduced maintenance
- Cost-savings potential
- Increased adaptability and scalability” [Rexx09]

2.3 BSF4ooRexx

BSF4ooRexx stands for Bean Scripting Framework for ooRexx. It is a framework that basically “allows Java to be used as a huge Rexx function Library”. [Flat04, p. 4]

BSF4ooRexx camouflages Java as Open Object Rexx. The huge potential of this concept lies in the combination of the advantages of both programming languages. On the one hand the vast amount of already available Java code becomes accessible; on the other hand the code is still as easily understandable as any ooRexx program code. All Java classes and all Java methods become available in ooRexx.

Realized is this with an Object Rexx package, called `BSF.cls`. It

- “Defines routines, classes and methods which hide the procedural interface from Object Rexx programs,
- Allows to import Java classes explicitly into Object Rexx in the form of Object Rexx proxy classes,
- Allows to create Object Rexx proxy objects which interact with the appropriate Java objects.” [Flat04, p. 13]

It is sufficient to use an ordinary text editor to create ooRexx files. When using the Bean Scripting Framework the correct ending of the filename is “.rxj”. These files are then executed with the Open Object Rexx Interface. However, for convenience reasons it is recommended to use a more sophisticated editor with features like syntax highlighting, for example Vim (http://www.vim.org).

2.4 Apache ODF Toolkit

“The Apache ODF Toolkit is a set of Java modules that allow programmatic creation, scanning and manipulation of Open Document Format (ISO/IEC 26300 == ODF) documents.” [AODF12] ODF is an open international standard for office documents.

A more detailed description of ODF is given in chapter 4. More information on the Apache ODF toolkit can be found in chapter 5.

2.5 Apache Xerces

Due to the fact that ODF documents are basically a bundle of XML files and other objects an XML processing software is needed.

Xerces, developed by the Apache Software Foundation (ASF), is an XML parser library for processing (i.e. parsing, manipulating, serializing etc.) XML files and has to be installed in order to run the nutshell examples presented in chapter 6. [AXer12]

Instructions on the installation process are available at chapter 3.4

2.6 Apache OpenOffice

As new ODF documents are created and existing ones are altered in chapter 6 adequate software to view these documents afterwards is required.

For this purpose the sophisticated free office software package Apache OpenOffice was chosen. It is better known as OpenOffice.org at the time of writing. In June 2011 it became a project of the Apache Software Foundation (ASF) and was renamed Apache OpenOffice. OpenOffice uses the OpenDocument Format as its default file format. It is written in C++ and Java and is available in over 110 languages, contributing to its worldwide distribution and usage. This popular office suite is comprised of six different applications:

- A word processor called Writer
- A spreadsheet software called Calc
- A presentation program called Impress
- A drawing software called Draw
- A database program called Base
- A tool for handling mathematical formulae called Math [AOpe12]

This paper is focused on three of them: Writer, Calc and Impress.

2.6.1 Writer

Apache OpenOffice Writer is a word processor. “A word processor is a computer application used for the production (including composition, editing, formatting, and possibly printing) of any sort of printable material.” [WWor12] Word Processors are used to create all kinds of extensive text documents, such as memos, business correspondence, letters etc.

Examples handling OpenDocument Format text documents (.odt) are shown in chapter 6.1. These odt files can be opened with OpenOffice Writer.

2.6.2 Calc

The spreadsheet component of the Apache OpenOffice software package is called Calc. “A spreadsheet is a computer application with tools that increase the user's productivity in capturing, analyzing, and sharing tabular data sets. It displays multiple cells usually in a two-dimensional matrix or grid consisting of rows and columns (in other words, a table, hence ‘tabular’).” [WSpr12]

Calc is comparable to Microsoft’s Excel both in look and feel and in range of features. Chapter 6.2 is dealing with OpenDocument Format spreadsheets.

2.6.3 Impress

Impress is designed to create presentations (slide shows). “A presentation program is supposed to help both: the speaker with an easier access to his ideas and the participants with visual information which complements the talk.” [WPre12]

Apache OpenOffice Impress is capable of exporting presentations to PDF files and SWF files. These Flash videos can be viewed on any computer with a Flash player installed. Examples handling OpenDocument Format presentation documents are shown in chapter 6.3.

3 Installation Guide

This chapter guides the reader through the installation processes of the required software. It shows where the needed software can be obtained. The instructions are intended for a Windows 7 system.

3.1 Java Runtime

Firstly a Java Runtime Environment (JRE) is required to be installed on your computer.

Please note that only the Runtime is needed, not the entire Java Development Kit (JDK), which would also include the Java compiler (`javac`). The JRE is the part of the Java software platform responsible for the Java bytecode execution.

The latest version of Java is available at http://www.java.com/de/download/. The website also provides a useful tool which checks whether Java is already installed or not.

3.2 ooRexx

The programming language ooRexx needs to be installed on your machine. It is free to download from http://www.oorexx.org/download.html. The latest release is version 4.1.0. Alternatively it is also available at Sourceforge: http://sourceforge.net/projects/oorexx/.

3.3 BSF4ooRexx

BSF4ooRexx camouflages Java as Open Object Rexx.

It is also available to download from the Sourceforge website: http://sourceforge.net/projects/bsf4oorexx/. It supports ooRexx version 4.1.0 and later.

3.4 Apache Xerces

Apache Xerces is an XML processor. Download the Java distribution from http://xerces.apache.org/mirrors.cgi.

Among the files of the package are two files called `xercesImpl.jar` and `xml-apis.jar`. Both have to be added to the environment variable CLASSPATH. With this entry Java is able to find the necessary classes.

Under Windows 7 go to the following path:

`Control panel / System / Advanced System Settings / Environment Variables`

And add the entire path (starting with the letter of the drive, usually "`C:`") of the two mentioned files to the "`CLASSPATH`" variable. Entries in the variable are separated by a semicolon (`;`). Figure 2 illustrates this procedure.

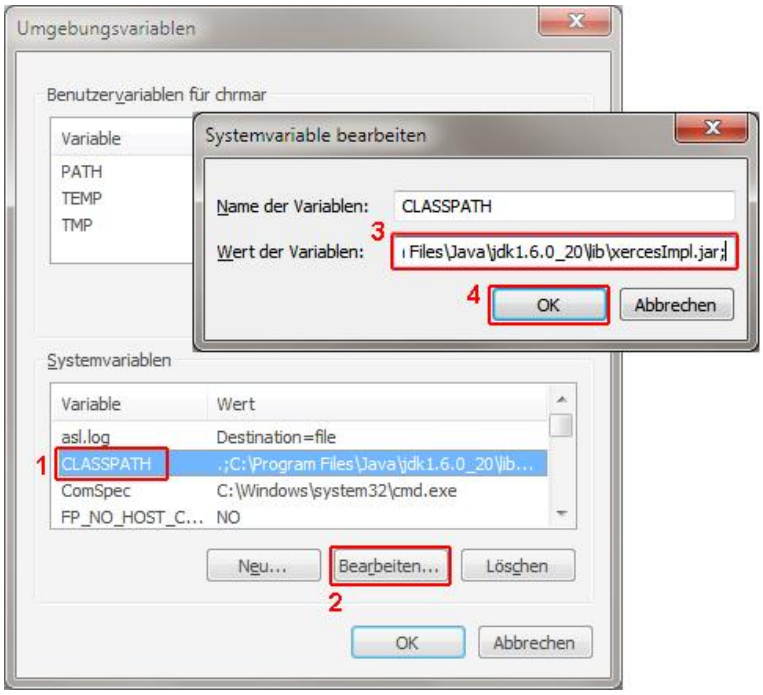


Figure 2 - Editing the environment variable

3.5 Apache ODF Toolkit

The Apache ODF Toolkit is currently undergoing incubation at the ASF at the time of writing in February 2012. “Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects.” [AODF12]

The first ODF Toolkit release under Apache (0.5-incubating) was published on January 14th 2012. It is ready to download from the following address: http://incubator.apache.org/odftoolkit/downloads.html.

The ODFDOM library, a .jar file, has to be added to the environment variable CLASSPATH too. For instructions on this procedure refer to chapter 3.4.

3.6 Apache OpenOffice

Finally a program to open and view the ODF documents is required. In this paper Apache OpenOffice was chosen. Please note that this is only one office software suite capable of opening and creating documents complying with the OpenDocument Format standard.

The latest version at is available at http://www.openoffice.org/download/.

4 OpenDocument Format

This chapter focuses on the OpenDocument Format for Office Applications. After a brief introduction it is shown how an OpenDocument Format file looks like and what it consists of.

4.1 Introduction and history

“The OpenDocument Format (ODF) is an open, XML-based document file format for office applications that create and edit documents containing text, spreadsheets, charts, and graphical elements.” [OASI06, p. 5]

The OpenDocument Format standard is maintained by the Organization for the Advancement of Structured Information Standards (OASIS). OASIS published the first version of ODF (1.0) in May 2005. It was created with the following objectives in mind. It should

- “be suitable for office documents containing text, spreadsheets, charts, and graphical documents;
- be compatible with XML v1.0 and W3C namespaces in XML v1.0 specifications;
- preserve the structure of the document to allow re-editing (for example, footnotes must be stored as structured footnotes, not just as text in the document that looks like a footnote);
- be friendly to transformations using the W3C’s Extensible Stylesheet Language (XSLT) or similar XML based languages or tools;
- keep the document’s content and layout information separate to enable independent processing; and
- ‘borrow’ from similar, existing standards wherever possible and permitted.” [Leib09, p. 85]

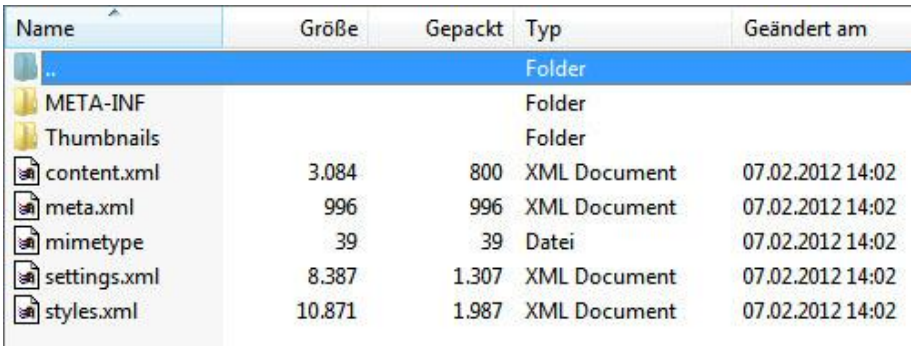
ODF became an official ISO standard (ISO/IEC 26300:2006) in May 2006. “The OpenDocument format aims to provide an open alternative to proprietary document formats.” [WOpe12]

4.2 Inside an OpenDocument Format File

“The ODF file structure uses a Zip file as a compressed archive to hold a series of XML files and other information (such as binary files containing embedded images) that describe the document's content and presentation.” [Ditc07, p. 17]

The Extensible Markup Language (XML) is a widely used markup language for describing structured, hierarchical data. Its tree structure consists mainly of elements and attributes, which describe these elements. [W3CX08]

The package of an ODF file can be viewed with a data compression utility such as WinRAR, WinZip or 7zip. Figure 3 shows the structure of a simple unzipped ODF text file package.



Name	Größe	Gepackt	Typ	Geändert am
..			Folder	
META-INF			Folder	
Thumbnails			Folder	
content.xml	3.084	800	XML Document	07.02.2012 14:02
meta.xml	996	996	XML Document	07.02.2012 14:02
mimetype	39	39	Datei	07.02.2012 14:02
settings.xml	8.387	1.307	XML Document	07.02.2012 14:02
styles.xml	10.871	1.987	XML Document	07.02.2012 14:02

Figure 3 - Structure of an ODF file

It is fascinating that a file as complex as a comprehensive presentation or a complicated spreadsheet basically comes down to plain XML files. The content of these XML files is described in the following chapters.

4.2.1 mimetype

MIME (Multipurpose Internet Mail Extensions) was originally an extension of the email format to support attachments and more. “MIME's use, however, has grown beyond describing the content of email to describe content type in general […]” [WMul12]

In the context of ODF the `mimetype` file is containing only a single line of code indicating the type of the document (text, spreadsheet etc.). [OASI07, p. 722]

Figure 4 shows this line for a text document.

```
application/vnd.oasis.opendocument.text
```

Figure 4 - MIME type of OpenOffice text document

4.2.2 META-INF/manifest.xml

The `manifest.xml` file is always located in the directory `META-INF`. “The main pieces of information stored in the manifest are:

- A list of all of the files in the package.
- The media type of each file in the package.
- If a file stored in the package is encrypted, the information required to decrypt the file is stored in the manifest.” [OASI07, pp. 711-717]

Figure 5 shows an excerpt of the file with the root element called `manifest` and some file-entries including the media-type and the path to the files.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <manifest:manifest xmlns:manifest="urn:oasis:names:tc:opendocument:xmlns:manifest:1.0">
3   <manifest:file-entry  manifest:media-type="application/vnd.oasis.opendocument.text"
    manifest:version="1.2"  manifest:full-path="/"/>
4   <manifest:file-entry manifest:media-type="text/xml"  manifest:full-path="content.xml"/>
5   <manifest:file-entry manifest:media-type="application/rdf+xml"
    manifest:full-path="manifest.rdf"/>
6   <manifest:file-entry manifest:media-type="text/xml"  manifest:full-path="styles.xml"/>
7   <manifest:file-entry manifest:media-type="text/xml"  manifest:full-path="meta.xml"/>
8   <manifest:file-entry manifest:media-type=""  manifest:full-path="Thumbnails/thumbnail.png"/>
9   <manifest:file-entry manifest:media-type=""  manifest:full-path="Thumbnails/"/>
10  <manifest:file-entry manifest:media-type="text/xml"  manifest:full-path="settings.xml"/>
11 </manifest:manifest>
```

Figure 5 - Excerpt of the manifest.xml file

4.2.3 meta.xml

Meta data is a term referring to data which describes other data. The expression “Data about data” is commonly used to describe Meta Data. [WMet12]

The `meta.xml` file in an ODF package contains information about the document itself, e.g.

- The version of ODF used in the document
- Document title
- Name of the author

- Time of the last save action
- Date of creation/modification
- Statistics like number of words, paragraphs or tables
- Language
- Keywords
- Number of editing cycles

Figure 6 shows a brief excerpt of the `meta.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
[...]
<office:meta>
 <meta:creation-date>2011-12-22T18:41:41.74</meta:creation-date>
 <meta:document-statistic meta:table-count="0" meta:image-count="0"
  meta:object-count="0" meta:page-count "1" meta:paragraph-count="1"
  meta:word-count="2" meta:character-count="12"/>
[...]
</office:meta>
```

Figure 6 - Excerpt of the meta.xml file

4.2.4 settings.xml

The `settings.xml` file contains application-specific information on settings like the zoom factor, window size or printer information. These pieces of information are independent from content, formatting or layout. [OASI07, p. 40]

In Figure 7 one can see the separation of view-settings such as the zoom factor and configuration-settings such as a boolean value (truth value, either true=1 or false=0) that specifies whether hidden text should be printed or not.

```
<?xml version="1.0" encoding="UTF-8"?>
<office:document-settings xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:config="urn:oasis:names:tc:opendocument:xmlns:config:1.0"
xmlns:ooo="http://openoffice.org/2004/office" office:version="1.2">
 <office:settings>
  <config:config-item-set config:name="ooo:view-settings">
   [...]
   <config:config-item config:name="ZoomFactor" config:type="short">100</config:config-item>
   [...]
  </config:config-item-set>
  <config:config-item-set config:name="ooo:configuration-settings">
   [...]
   <config:config-item config:name="PrintHiddenText" config:type="boolean">false</config:config-item>
   [...]
  </config:config-item-set>
 </office:settings>
</office:document-settings>
```

Figure 7 - Excerpt of the settings.xml file

4.2.5 styles.xml

Styles are an essential part of an ODF document. “OpenDocument makes heavy use of styles for formatting and layout. Most of the style information is here (though some is in content.xml).

Styles types include:

- Paragraph styles
- Page styles
- Character styles
- Frame styles
- List styles” [WOpe12]

An interesting fact is that styles are even used for implementing manual formatting, e.g. when making a text bold within the Writer application. In such cases the application automatically creates new styles in the background and assigns them to the content accordingly. The ODF specification therefore distinguishes three types of styles:

- **“Common styles**
  Most office applications support styles within their user interface. Within this specification, the XML representations of such styles are referred to as styles. When a differentiation from the other types of styles is required, they are referred to as common styles. The term common indicates that this is the type of style that an office application user considers to be a style.
- **Automatic styles**
  An automatic style contains formatting properties that, in the user interface view of a document, are assigned to an object such as a paragraph. The term automatic indicates that the style is generated automatically. In other words, formatting properties that are immediately assigned to a specific object are represented by an automatic style. This way, a separation of content and layout is achieved.

- **Master styles**
  A master style is a common style that contains formatting information and additional content that is displayed with the document content when the style is applied. An example of a master style are master pages. Master pages can be used in graphical applications. In this case, the additional content is any drawing shapes that are displayed as the background of the draw page. Master pages can also be used in text documents. In this case, the additional content is the headers and footers. Please note that the content that is contained within master styles is additional content that influences the representation of a document but does not change the content of a document." [OASI07, p. 55]

The reasons for this approach are also explicated in the specification:

1. "The format and layout of the document get separated from the document content.
2. If two or more objects have the same formatting properties and styles assigned, the formatting properties that are assigned to the objects directly can be represented by a single automatic style for all objects. This saves disk space and allows styles to integrate seamlessly into the overall document style." [OASI07, p. 479]

4.2.6 content.xml

The most important file is the `content.xml` file as it contains all of the actual content of the document i.e. text, tables and so on. It is notably derived from HTML and quite legible to humans.

Figure 8 shows a (very condensed) excerpt of a text document's `content.xml` file containing a paragraph with the text "Hello World!".

```
1 <?xml version="1.0" encoding="UTF-8"?>
2  <office:document-content>
3   <office:body>
4     <office:text>
5       <text:p text:style-name="Standard">Hello World!</text:p>
6     </office:text>
7   </office:body>
8 </office:document-content>
```

Figure 8 - Condensed excerpt of the content.xml file

5 Apache ODF Toolkit

This chapter is dealing with the Apache ODF Toolkit. Since there have been recent changes concerning the ODF Toolkit at the time of writing a brief overview of the Toolkit's history is given in chapter 5.1. All components, particularly the ODFDOM API are described with a focus on the underlying layer model in chapter 5.2.

5.1 Introduction and history

"The Apache ODF Toolkit is a set of Java modules that allow programmatic creation, scanning and manipulation of Open Document Format (ISO/IEC 26300 == ODF) documents." [AODF12]

It was firstly proposed in 2006 by Rob Weir from IBM (Mr. Weir is still a contributor to the ODF Toolkit project at the time of writing):

"The 700-page ODF specification defines what an application and platform neutral office document in XML format looks like. But we are lacking high-level developer tools that allow us to be productive with this format. An ODF toolkit will go a long way to accelerating the arrival of new ODF-supporting applications and solutions since it reduces these difficulties that currently confront the developer." [Weir06, p. 1]

He argues that "there is very little code that directly manipulates XML" [Weir06, p. 1]. Instead, most developers would use DOM or SAX to manipulate XML. "For ODF to become more attractive to application developers, we need tools that are easy to master and that work with the tools the application developers already use." [Weir06, p. 1]

"The ideal tool for the application developer would:

- be simple to use, requiring a minimal amount of code to solve simple tasks. The application developer should be able to write a script to generate a formatted spreadsheet with a loan amortization table in 30 lines of code.

- allow easy integration with scripting hosts, such as Perl and Python.
- provide a high-level model, closer to the problem domain of application developers, with objects which represent spreadsheets, cell and column formats.” [Weir06, p. 2]

That is how the ODF Toolkit originated. In 2011 the project was moved to the Apache Incubator (parallel with OpenOffice). “Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects.” [AODF12]

The first ODF Toolkit release under Apache (0.5-incubating) was published on January 14th 2012.

5.2 Components

The ODF Toolkit is a compilation of different programs and tools, each performing a specific task concerning OpenDocument Format Documents. These various components are described in this chapter.

5.2.1 ODF Validator

“ODF Validator is a tool that validates OpenDocument files and checks them for certain conformance criteria.” [AVal12] It is available as a Java library.

The ODF Validator detects the ODF version of a give document automatically and validates the document accordingly. It validates the document against an OpenDocument schema, which has to be stored locally on the machine. “The schemas and the files that should be validated can be specified in a configuration file.” [AVal12] The Validator requires a Java Runtime Environment 1.5 or higher.

5.2.2 ODF XSLT Runner

“ODF XSLT Runner is a small Java application that allows you to apply XSLT stylesheets to XML streams included in ODF packages without extracting them from the package. It can be used from the command line.” [AXSL12]

A common usage scenario would be the transformation of an ODF document into XHTML. With the XSLT Runner this is possible without having to extract the ODF package prior to the transformation. [AXST12]

There are example stylesheets available on the homepage of the ODF Toolkit.

5.2.3 Simple API

Like ODFDOM the Simple API is a programming interface providing functions to create and modify ODF documents. It is a high-level Java API in the sense that it enables the user to perform relatively complex tasks with just a few lines of code. For example, it provides simple methods to create a new document or to save a document with just a single instruction. [ASim12]

Online one can find a very helpful and extensive demo page as well as a so called “Document Cookbook”. These two links are helpful getting started with the API more easily:

- http://incubator.apache.org/odftoolkit/simple/demo/index.html
- http://incubator.apache.org/odftoolkit/simple/document/cookbook/index.html

Please note that the URLs will probably change when the project overcomes the incubator status. In this case refer to the project homepage of the Apache Software Foundation.

5.3 ODFDOM

“ODFDOM is a free OpenDocument 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.” [ADOA12]

ODFDOM is one part of the Apache ODF Toolkit. The reference implementation is written in Java at the time of writing in February 2012.

5.3.1 ODFDOM Layers

As stated above, the primary objective is to provide an easy to use API for handling ODF documents. In order to achieve a well-designed modular structure the ODFDOM API follows a layered approach. There are two different ODFDOM layers, which are described in more detail in the following chapters. [ADOP12]

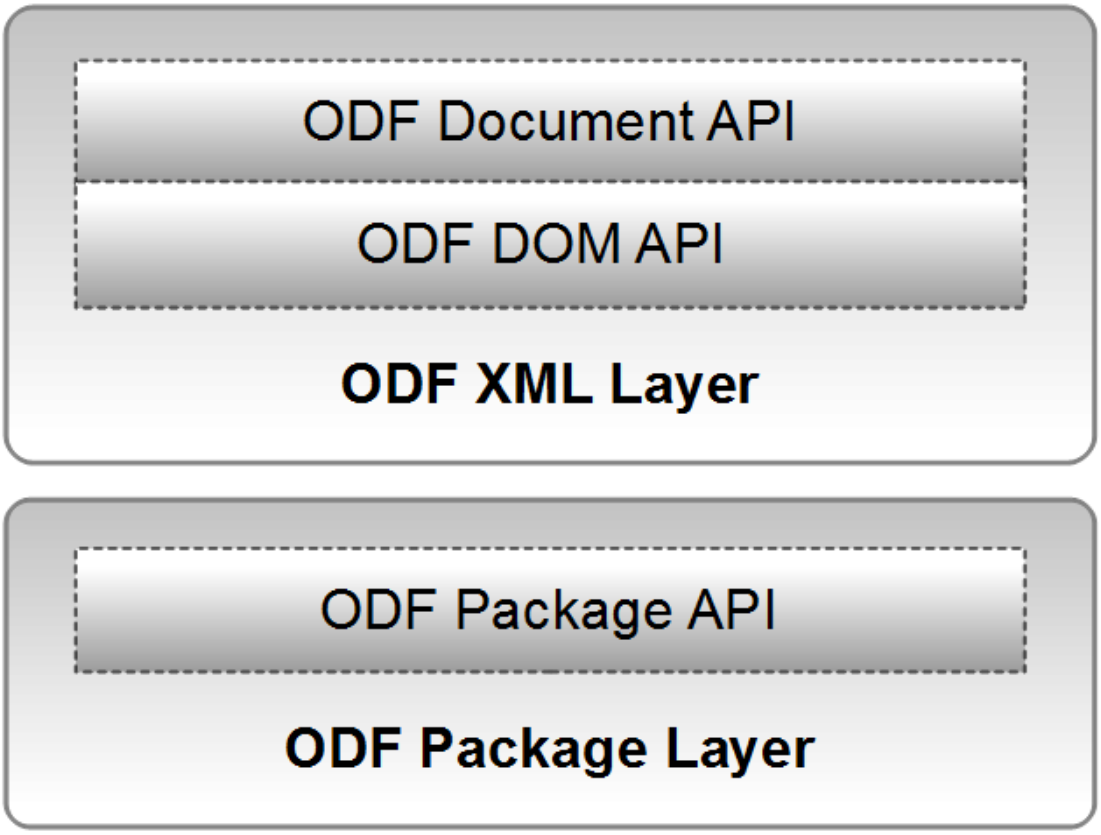


Figure 9 - The ODFDOM Layers [ADOP12]

5.3.1.1 XML Layer

“The ODF XML Layer provides all the features of an office format, such as tables, images, numbering etc. All features are defined in the first part of the ODF 1.2 specification describing the ODF XML schema.” [ADOL12]

It consists of two different APIs:

- ODF Document API
- ODF DOM API

Whereas the Document API provides a high level view on the features and enables complex procedures with little code, the DOM API offers detailed access to the XML on a very low level. [ADOL12]

ODF Document API

“This API is concerned about usability, hiding all ODF XML implementation details from the user, covering frequent user scenarios.” [ADOL12]

Examples for such user scenarios are changing the content of a specific spreadsheet cell or inserting a new foil into a presentation document. The Document API provides a very high level view on the document, offering simple instructions for such common scenarios.

A class of this API covers multiple ODF XML elements, i.e. each instruction has significant impacts on the underlying XML files of the document’s package. “As naming convention all sources of the ODF document functionality layer are organized beyond org.odftoolkit.odfdom.doc.*” [ADOL12]

ODF DOM API

The ODF DOM API, in contrast to the Document API, provides a low-level view on a document. It gives access to the single elements of the XML files. It is designed to easily manipulate a specific XML node and therefore altering the document.

“For every ODF XML element and ODF XML attribute defined by the ODF grammar (the RelaxNG schema) a unique class exists, providing methods for their allowed children.” [ADOL12]

“All sources of the typed DOM API are organized beyond org.odftoolkit.odfdom.dom.*” [ADOL12]

5.3.1.2 Package Layer

“The ODF Package Layer provides access to all resources stored within the ODF package, such as XML streams, images or embedded objects.” [ADOL12]

All ODF Package features are covered by the ODF Package API. These features take advantage of technologies like ZIP compression and W3C encryption.

The Package layer views a document like described in chapter 4.2, i.e. as a bundle of resources zipped to a package. For example, it is possible to add an image to an ODF Document package, which is never actually added (and thus not displayed) to the document.

6 Nutshell Examples

The expression “in a nutshell” means “summed up briefly”. The objective of the following nutshell examples is to give an insight into the possibilities and underlying concepts of the ODF Toolkit while being as short as possible. They are using the ODFDOM API (5.3), which is part of the Apache ODF Toolkit bundle. At the time of writing there are discussions among the developers on the coexistence of the two APIs (Simple API & ODFDOM API).

In this chapter nine examples are presented, three for text documents (ODT, 6.1), three for spreadsheet documents (ODS, 6.2) and three for presentation documents (ODP, 6.3). Each example starts with the code followed by a description of the code and ends with a screenshot of the resulting document opened with Apache OpenOffice.

It is recommended to work through these examples in the order in which they are presented, because they are partly built on each other. For a better understanding of the translation process from Java to ooRexx refer to the first ODT example in chapter 6.1.1.

A really helpful collection of further introductory examples using the Simple API can be found online, it is called the Simple ODF Cookbook: http://simple.odftoolkit.org/cookbook/.

6.1 ODT / Writer

In this chapter three examples dealing with ODF Text Documents are presented. For those readers not familiar with the programming language ooRexx, the first example (6.1.1) is exemplified both in Java (Figure 11) and ooRexx (Figure 10).

6.1.1 Nutshell Example ODT 1

In this nutshell example a new ODF Text Document (ODT) is created, some text is added to the document and the document is saved with a given file name.

Code ooRexx

```
1  odftd=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
2  TextDocument=odftd~newTextDocument
3
4  TextDocument~addText("Hello World!")
5  TextDocument~newParagraph("This is my first Text Document.")
6  TextDocument~newParagraph
7  TextDocument~save("writer_ne1.odt")
8
9  ::requires BSF.CLS
```

Figure 10 - Nutshell Example ODT 1 Code ooRexx

As one can see in Figure 10 ooRexx code is quite clean and straightforward. Few cryptic symbols like curly brackets or semicolons are required (as opposed to Java code). Instructions need not to be closed with a semicolon, instead the line break stands as an indicator for an instruction's end. However, it is possible to state more than one instruction in each line, in this case the instructions have to be separated with semicolons.

A very important symbol in ooRexx is the twiddle sign (~). It is also called the "message send" symbol. "Sending a message to an object results in performing some action; that is, it results in running some underlying code. The action-generating code is called a method. When you send a message to an object, you specify its method name in the message." [RPro09]

In ooRexx the syntax to call a method has the following structure:

```
object~method(parameters)
```

The corresponding Java code, resulting in the exact same ODF document is shown in Figure 11. Among the advantages of ooRexx are the dynamic data typing, which means that no declarations are required (as in line 4 of Figure 11) and the omitted case sensitivity. For further information regarding the programming language ooRexx refer to chapter 2.2.

Code Java

```
1  import org.odftoolkit.odfdom.doc.OdfTextDocument;
2  public class writer_ne1 {
3      public static void main(String[] args) {
4          OdfTextDocument TextDocument;
5          TextDocument=OdfTextDocument.newTextDocument();
6
7          TextDocument.addText("Hello World!");
8          TextDocument.newParagraph("This is my first Text Document.");
9          TextDocument.newParagraph();
10
11         TextDocument.save("writer_ne1.odt");
12     }
13 }
```

Figure 11 - Nutshell Example ODT 1 Code Java

Description

The following description (including line numbers) refers to the ooRexx code in Figure 10. First of all, look at line 9:

```
::requires BSF.CLS
```

Although this statement is placed in the last line of the examples, it is actually the first line to be carried out. When an ooRexx program is executed all lines are read and syntactically checked.

The two colons stand for a directive. A directive in ooRexx is a statement that gets carried out previous to all other non-directive statements. In this case it is a “requires directive”. “You use the ::REQUIRES directive when a program needs access to the classes and objects of another program.” [AFMM09, p. 42]

Line 9 loads the package `BSF.CLS`. The package supplies the class BSF, which enables us to create Java objects from within ooRexx. This line can be found in all the examples as it is required to camouflage Java as ooRexx and therefore to use the ODF Toolkit, which is available as a Java Archive (see 3.5).

In line 1 the BSF functionality is used to import the Java class `org.odftoolkit.odfdom.doc.OdfTextDocument`. Please note that the exact case is required and the name of the class has to be fully qualified, i.e. the full path to the class including the package. After this statement we can treat this imported Java class as if it was an ooRexx class.

The class `OdfTextDocument` (part of the ODF Document API, see 5.3.1.1) represents an empty ODF text document. It inherits several essential methods from the class `org.odftoolkit.odfdom.doc.OdfDocument`, e. g.:

- `loadDocument`
- `save`

At this point the Java class was only imported. Now an instance of the imported class is needed. At line 2 we send the message `newTextDocument`, i.e. the method’s name, to the class and we name the instance `TextDocument`. `TextDocument` stands now for an instance of the class `OdfTextDocument`.

Two methods of the class `OdfTextDocument` for adding some text to a text document are called from line 4 to line 6:

- `addText`
- `newParagraph`

The `addText` method requires a string parameter. The `newParagraph` method also accepts a string parameter, but it can be used without any parameters to add a line break to the document, as illustrated in line 6 of Figure 10.

Finally in line 7 the `save` method is carried out to save the created document with a file name specified in a parameter. Although this statement triggers complex actions the statement itself is very simple.

Result

Figure 12 shows a screenshot of the resulting ODF text document opened with OpenOffice Writer.

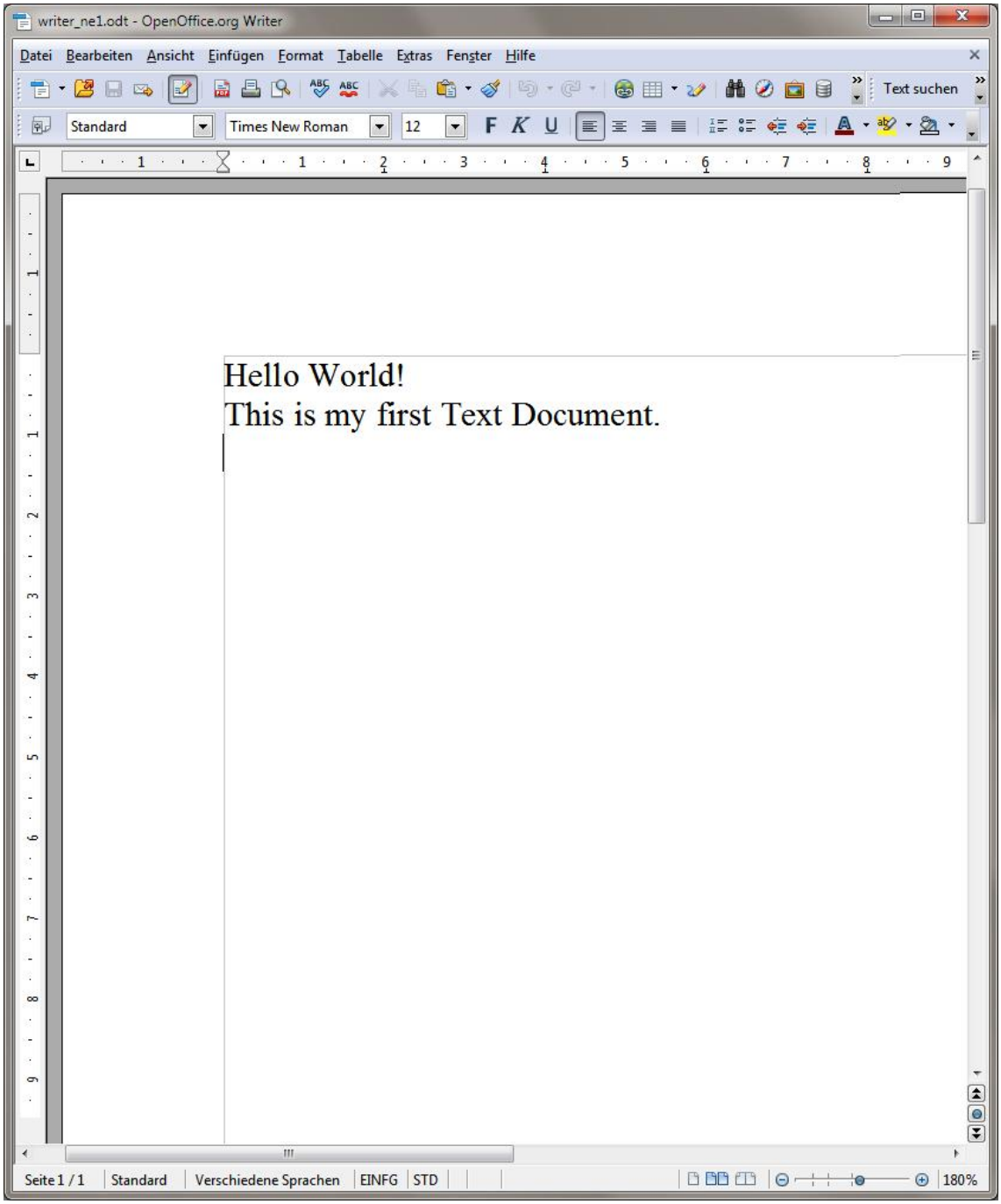


Figure 12 - Nutshell Example ODT 1 Resulting Document

6.1.2 Nutshell Example ODT 2

In this example a text document is created and a table is added to the document. Some cells of the table get assigned to an ooRexx variable by which they can be referenced from thereon. Text is added to the table and the cells of the table are formatted. Finally the document is saved with a specified name.

Code

```
1  odftd=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
2  odftable=bsf.import("org.odftoolkit.odfdom.doc.table.OdfTable")
3
4  outputOdt=odftd~newTextDocument
5  outputOdt~newParagraph("I'm inserting a table now.")
6  outputOdt~newParagraph
7
8  table=odftable~newTable(outputOdt, 2, 2)
9
10 cell_top_left=table~getCellByPosition(0, 0)
11 cell_top_right=table~getCellByPosition(1, 0)
12 cell_bottom_left=table~getCellByPosition(0, 1)
13
14 cell_top_left~setCellBackgroundColor("#CCCCCC")
15 cell_top_right~setCellBackgroundColor("#CCCCCC")
16
17 cell_top_left~setStringValue("This is...")
18 cell_top_right~setStringValue("... the header-row.")
19
20 cell_bottom_left~setStringValue("and this text is centered")
21 cell_bottom_left~setHorizontalAlignment("center")
22
23 outputOdt~save("writer_ne2.odt")
24
25 ::requires BSF.CLS
```

Figure 13 - Nutshell Example ODT 2 Code

Description

In the first two lines the required classes – stated with their fully qualified class name – for this example are imported:

- The `org.odftoolkit.odfdom.doc.OdfTextDocument` class just as in the first ODT example (6.1.1).
- The `org.odftoolkit.odfdom.doc.table.OdfTable` class for the table. Interestingly this class is also heavily used in spreadsheet documents, which basically consist of a table (see chapter 6.2).

We are already familiar with the statements in line 4 to 6. We create a new instance of the class `OdfTextDocument` and add some text to the document.

In line 8 an instance of the imported class `OdfTable` is created. In contrast to the simple constructor method of the `OdfTextDocument` class, this class has ten different constructor methods varying in the parameters. The parameters stated in this example are:

- The document object to which the table is added (`outputOdt`)
- The number of rows (`2`)
- The number of columns (`2`)

The table instance gets the simple name `table` by which it can be referenced.

It is possible to assign freely chosen names to certain cells or cell ranges. In some cases this can simplify handling the cells, because names are easier to work with than numbers of a coordinate system. Nevertheless the cells can still be referenced with their coordinates, which can be much more comfortable, e.g. when automatically manipulating a greater number of cells with a loop.

From line 10 to line 12 cell objects are assigned to the ooRexx variables `cell_top_left`, `cell_top_right` and `cell_bottom_left`. The name of this method is `getCellByPosition`. Its two parameters are the coordinates, column and row. Please note that both axes start with zero, common usage in programming languages. From now on we can refer to these table cells using the ooRexx variables.

The next method to be used in this example is the `setCellBackgroundColor` method in line 14 and 15. For colors the ODF Toolkit includes a particular class, `org.odftoolkit.odfdom.type.Color`. In this case we simply state the RGB code of the desired color in a parameter of the method to distinguish the top row of the table.

RGB stands for red, green and blue. In the example the hexadecimal notation is used. Digits one and two are for the amount of red that gets “added” to the color (RGB is an additive color model). The next two digits are for the amount of green and the last two digits are for the color blue. Each two digits range from a minimum of `00` to the maximum `FF` (255 in decimal). [WRGB12] Our color code `CCCCCC` results in a tone of grey (see Figure 14).

Our table is still without any content. In order to change that we add some text to the table cells in line 17 to 20 using the `setStringValue` method of the class `org.odftoolkit.odfdom.doc.table.OdfTableCell`. For adding numbers to the cells we would use the `setDoubleValue` method (see line 18 and 19 in Figure 19).

These set methods (`setStringValue`, `setCellBackgroundColor`, …) have a counterpart named get methods. With these methods the value can be retrieved again. This dichotomy is common usage in modern programming languages.

The last statement before we save the document can be found in line 21. The `setHorizontalAlignment` method is called. This method specifies where in the cell the text is placed. It accepts the following parameters:

- `center`
- `justify`
- `left` or `start`
- `right` or `end`

Result

The code illustrated in Figure 13 results in the document shown in Figure 14.

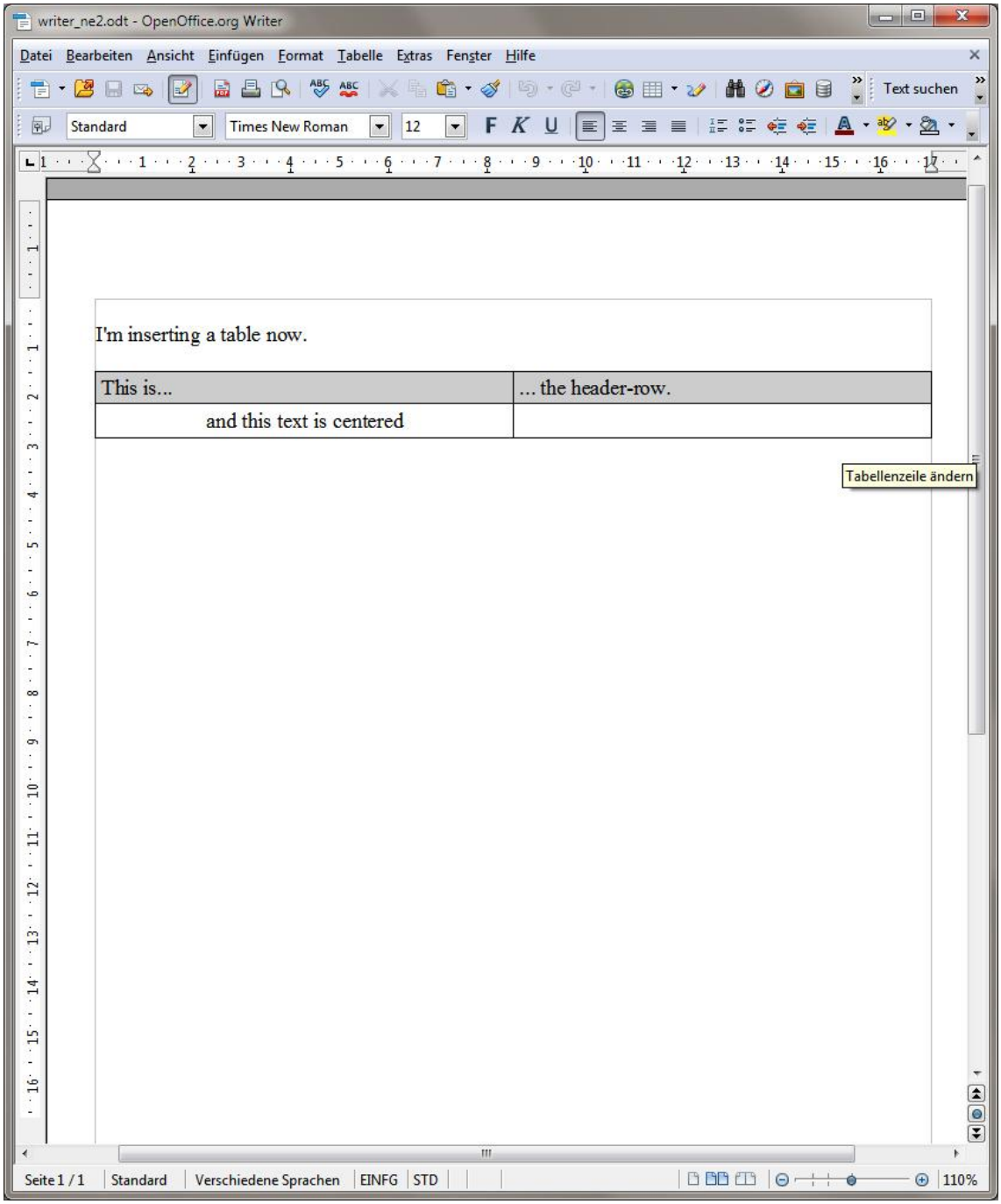


Figure 14 - Nutshell Example ODT 2 Resulting Document

6.1.3 Nutshell Example ODT 3

The third ODT nutshell example illustrates how to load an existing document and how to add a locally stored image to a text document.

Code

```
1  odftd=bsf.import("org.odftoolkit.odfdom.doc.OdfTextDocument")
2  outputOdt=odftd~loadDocument("writer_ne1.odt")
3
4  outputOdt~newParagraph
5  outputOdt~newParagraph("And this is the third nutshell example.")
6  outputOdt~newParagraph("Inserting an image.")
7  outputOdt~newParagraph
8
9  imguri=.bsf~new("java.net.URI","bierfriedl.jpg")
10 outputOdt~newImage(imguri)
11
12 outputOdt~save("writer_ne3.odt")
13
14 ::requires BSF.CLS
```

Figure 15 - Nutshell Example ODT 3 Code

Description

As in the examples above, the class `OdfTextDocument` is imported in line 1.

Line 2 exemplifies another useful method of this class: the `loadDocument` method. By stating the path to the document (in this case just the name of the document because it is located in the same directory as the ooRexx script) in a parameter the entire document is obtained and can be manipulated.

So the ODF Toolkit allows importing and therefore editing existing documents with a single line of code which is quite convenient. Please note that this example will only work after the first nutshell example (see 6.1.1) is executed.

In the lines 4 to 7 some text is added to the document with the `newParagraph` method at the end of the document.

An instance of the java class `java.net.URI` is created via the BSF class in line 9. Therefore the `new` method of the BSF.CLS is used. Two parameters are stated:

- Fully qualified name of the Java class (`java.net.URI`)
- Path to the file (`bierfriedl.jpg`)

With the method `newImage` the image can easily be added to the text document (see line 10). Finally, the document is saved with the name `writer_ne3.odt`.

Result

Figure 16 shows a screenshot of the resulting ODF text document opened with OpenOffice Writer.

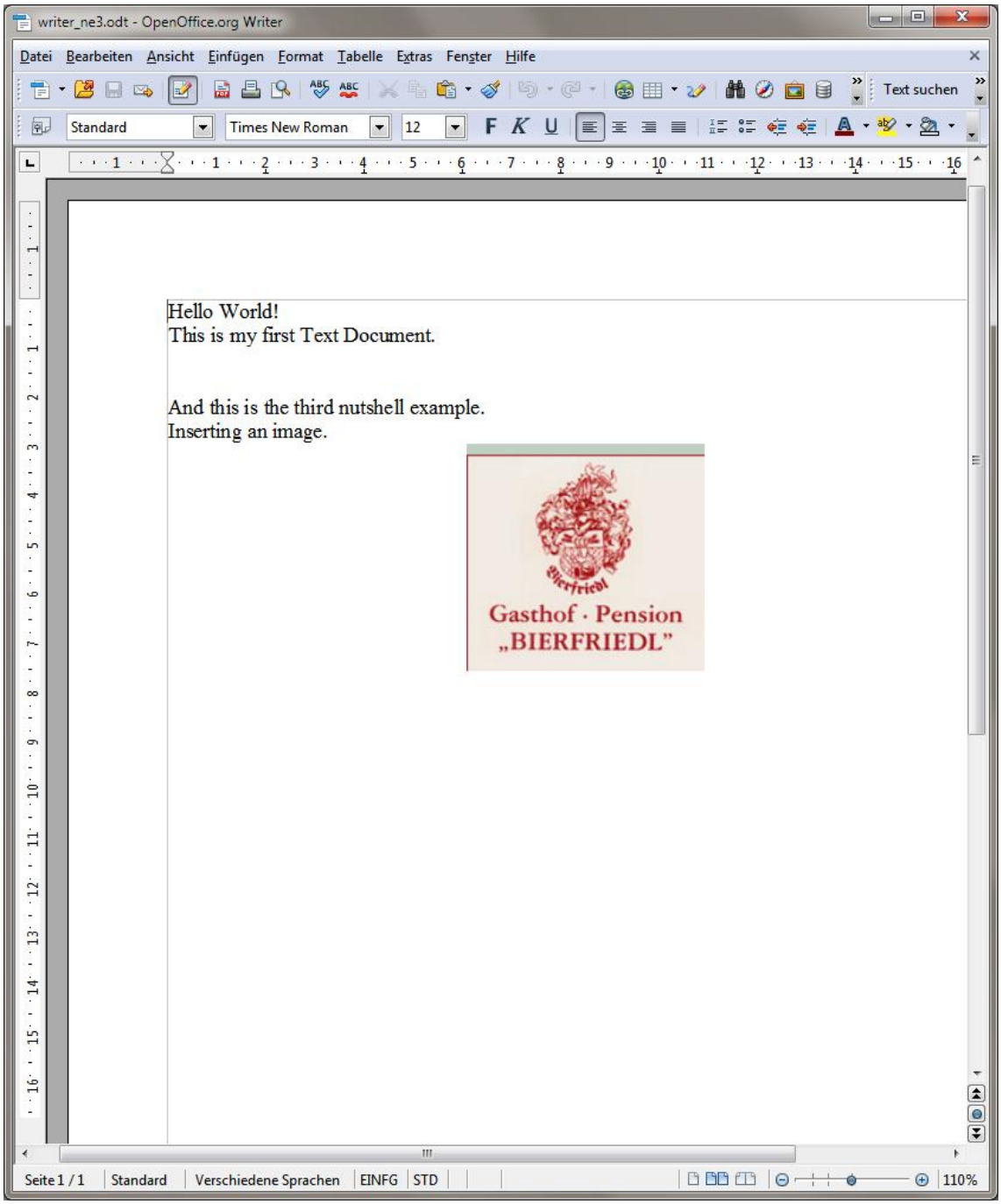


Figure 16 - Nutshell Example ODT 3 Resulting Document

6.2 ODS / Calc

In the three nutshell examples of this chapter spreadsheet documents (ODS) are created and altered.

6.2.1 Nutshell Example ODS 1

This code snippet exemplifies how to create a new spreadsheet document with the ODF Toolkit, add some text to the first worksheet (there are three of them in a document by default) and save the document.

Code

```
1  odfsd=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
2  SpreadsheetDocument=odfsd~newSpreadsheetDocument
3
4  odfTable=SpreadsheetDocument~getTableList~get(0)
5  odfCell=odfTable~getCellByPosition(0,0)
6
7  odfCell~setStringValue("Hello World!")
8
9  SpreadsheetDocument~save("calc_ne1.ods")
10
11 ::requires BSF.CLS
```

Figure 17 - Nutshell Example ODS 1 Code

Description

Whereas text documents possess a continuous space to which all kinds of objects such as text paragraphs, tables or images can be added, a spreadsheet document is fundamentally different. It is composed of a two-dimensional matrix like a table with rows and columns.

Due to this structure it is mandatory to specify the cell, i.e. the combination of row number and column number, when adding some content to the spreadsheet document.

In the first line of the example the `OdfSpreadsheetDocument` class is imported. A new instance of the class is created subsequently in line 2.

In line 4 the table of the first worksheet (with number 0) is obtained. The variable `odfTable` now references an instance of the class `org.odftoolkit.odfdom.doc.table.OdfTable`, which we already used in chapter 6.1.2.

We are familiar with line 5 in which we name the top left cell with the coordinates 0,0 `odfCell`. In line 7 a string gets added to this cell and finally the document is saved with the `save` method at line 9.

Result

Figure 18 shows a screenshot of the resulting ODF spreadsheet document opened with OpenOffice Calc.

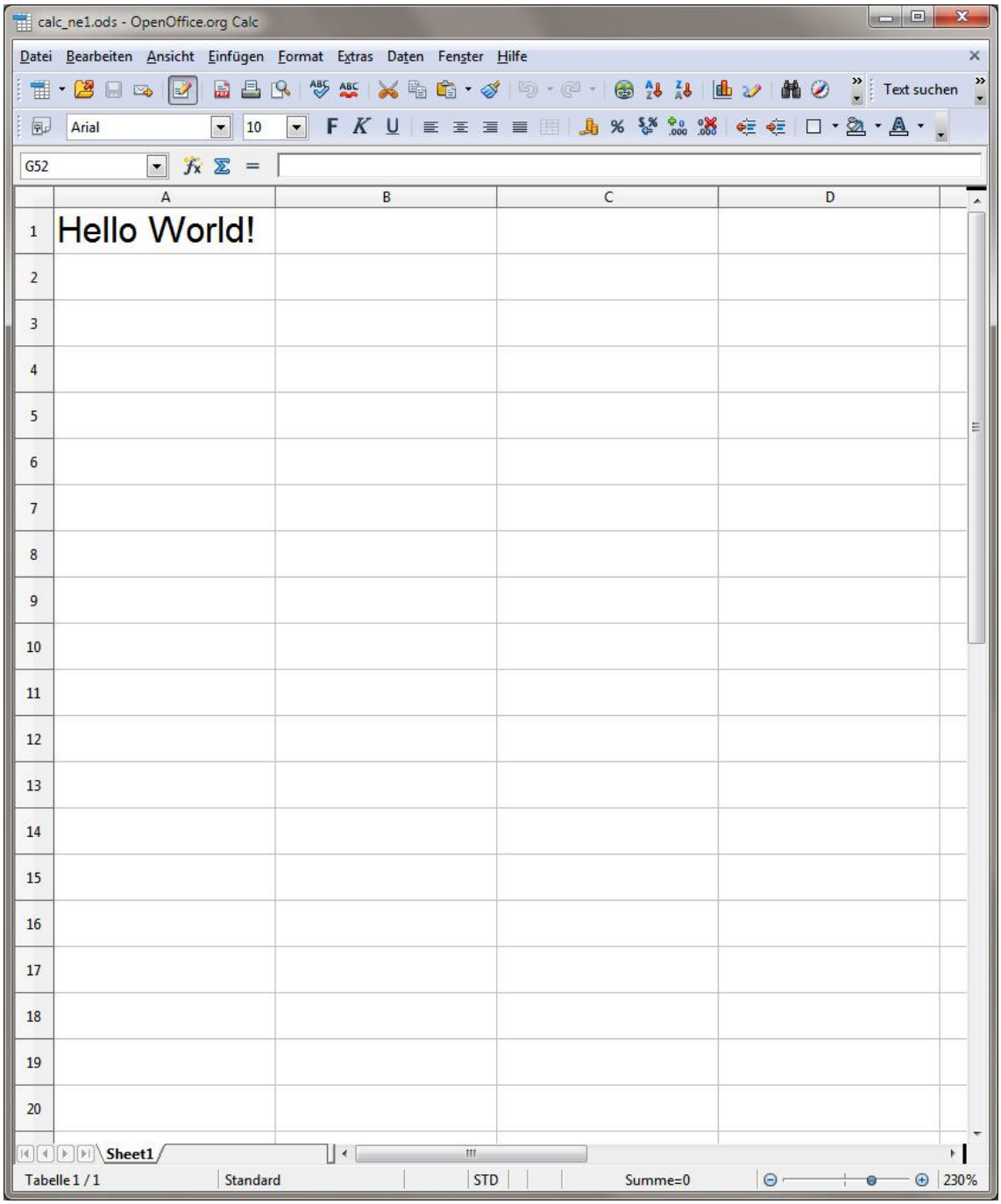


Figure 18 - Nutshell Example ODS 1 Resulting Document

6.2.2 Nutshell Example ODS 2

In this example a new spreadsheet document is created. Thereafter its cells are filled with some text and numbers. A formula is used to calculate a sum from these numbers and the result is displayed in a cell. Finally some cells of the worksheet are merged together and the document is saved.

Code

```
1  odfsd=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
2  SpreadsheetDocument=odfsd~newSpreadsheetDocument
3
4  odfTable=SpreadsheetDocument~getTableList~get(0)
5
6  A3=odfTable~getCellByPosition(0,2)
7  B3=odfTable~getCellByPosition(1,2)
8  A4=odfTable~getCellByPosition(0,3)
9  B4=odfTable~getCellByPosition(1,3)
10 A5=odfTable~getCellByPosition(0,4)
11 B5=odfTable~getCellByPosition(1,4)
12 A6=odfTable~getCellByPosition(0,5)
13 B6=odfTable~getCellByPosition(1,5)
14
15 A3~setStringValue("Product A"); A4~setStringValue("Product B")
16 A5~setStringValue("Product C"); A6~setStringValue("Sum")
17
18 B3~setDoubleValue("99.99"); B4~setDoubleValue("19.99")
19 B5~setDoubleValue("59.99")
20 B6~setFormula("=B3+B4+B5")
21
22 headline=odfTable~getCellRangeByPosition(0,1,1,1)
23 headline~merge
24 headline=odfTable~getCellByPosition(0,1)
25 headline~setStringValue("CALCULATION")
26
27 SpreadsheetDocument~save("calc_ne2.ods")
28
29 ::requires BSF.CLS
```

Figure 19 - Nutshell Example ODS 2 Code

Description

The lines one to four of Figure 19 are similar to the example in 6.2.1. A new spreadsheet document is created and the table of the first worksheet is obtained.

Spreadsheet software commonly uses letters for the columns and numbers for the rows. These two elements are then combined to describe a single cell (e.g. “A1” is the top left cell). This naming convention is realized to name the variables accordingly in the lines 6 to 13 with the familiar method `getCellByPosition`.

In the lines 15 to 18 one can see two instructions in each line. These instructions have to be separated by a semicolon. The semicolon can be omitted in lines with a single instruction.

The second column (B) is filled with some numbers using the method `setDoubleValue` in lines 18 and 19. The values of these cells are added together in line 20. The `setFormula` method expects a formula or any string as parameter.

In line 22 the method `getCellRangeByPosition` names a range of adjacent cells `headline`. The parameter consists of four numbers which specify the range of cells:

- Start column
- Start row
- End column
- End row

The two cells `A2` and `B2` are merged together to one cell in line 23. This new cell – with the coordinates 0, 1 – is named `headline` again in line 24 and filled with a string in line 25.

Result

The code illustrated in Figure 19 results in the document shown in Figure 20.

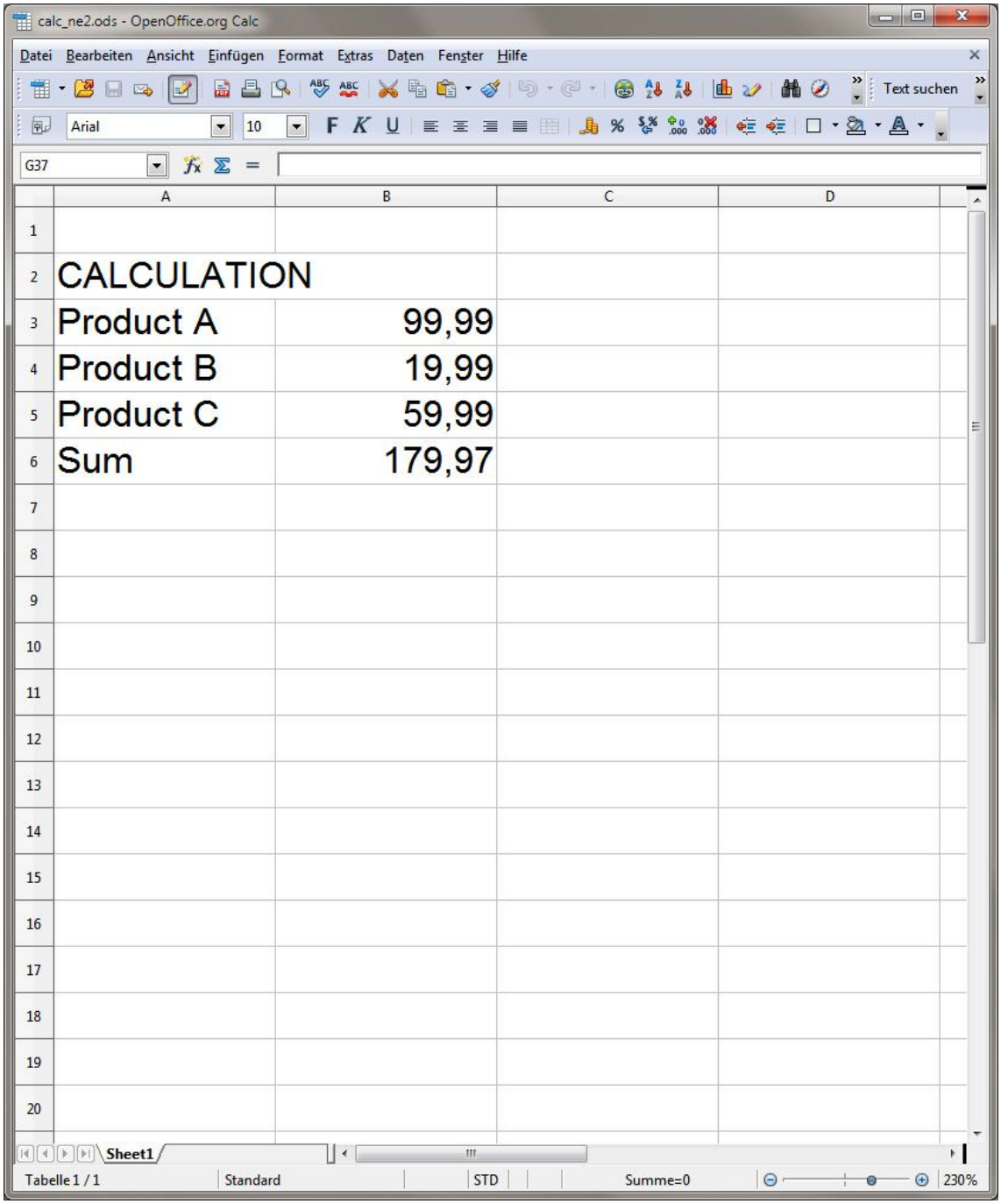


Figure 20 - Nutshell Example ODS 2 Resulting Document

6.2.3 Nutshell Example ODS 3

In the third ODS nutshell example an existing spreadsheet document (the one created in 6.2.2) is loaded. The name of one of the worksheets is changed and a new worksheet is added to the document. Some cells are formatted and an entire row is erased. Finally the document is saved with the name `calc_ne3.ods` (leaving the original document `calc_ne2.ods` unchanged).

Code

```
1  odfsd=bsf.import("org.odftoolkit.odfdom.doc.OdfSpreadsheetDocument")
2  SpreadsheetDocument=odfsd~loadDocument("calc_ne2.ods")
3
4  odfTable=SpreadsheetDocument~getTableList~get(0)
5  odfTable~setTableName("Calculation")
6
7  odfTable~newTable(SpreadsheetDocument)
8  odfTableNew=SpreadsheetDocument~getTableList~get(1)
9  odfTableNew~setTableName("New")
10
11 headline=odfTable~getCellByPosition(0,1)
12 headline~setHorizontalAlignment("center")
13 headline~setVerticalAlignment("middle")
14 headline~setCellBackgroundColor("#CCCCCC")
15
16 headline=odfTable~getRowByIndex(1)
17 row2=odfTable~getRowByIndex(2); row3=odfTable~getRowByIndex(3)
18 row4=odfTable~getRowByIndex(4); row5=odfTable~getRowByIndex(5)
19 headline~setHeight(15,0); row2~setHeight(7,0)
20 row3~setHeight(7,0); row4~setHeight(7,0); row5~setHeight(9,0)
21
22 columnA=odfTable~getColumnByIndex(0); columnA~setWidth(40)
23 columnB=odfTable~getColumnByIndex(1); columnB~setWidth(20)
24
25 A3=odfTable~getCellByPosition(0,2)
26 B3=odfTable~getCellByPosition(1,2)
27 A4=odfTable~getCellByPosition(0,3)
28 B4=odfTable~getCellByPosition(1,3)
```

```
29 A5=odfTable~getCellByPosition(0,4)
30 B5=odfTable~getCellByPosition(1,4)
31 A6=odfTable~getCellByPosition(0,5)
32 B6=odfTable~getCellByPosition(1,5)
33
34 A3~setVerticalAlignment("middle")
35 A4~setVerticalAlignment("middle")
36 A5~setVerticalAlignment("middle")
37 A6~setHorizontalAlignment("right")
38 A6~setCellBackgroundColor("#EEEEEE")
39 B6~setCellBackgroundColor("#EEEEEE")
40
41 odfTable~removeRowsByIndex(0,1)
42
43 SpreadsheetDocument~save("calc_ne3.ods")
44
45 ::requires BSF.CLS
```

Figure 21 - Nutshell Example ODS 3 Code

Description

The convenient `loadDocument` method, which we already saw in 6.1.3 is also available for spreadsheet documents. It is used in line 2 to load the document created in 6.2.2.

In line 5 the method `setTableName` is carried out in order to rename the first worksheet of the document. From line 7 to line 9 a new worksheet is created, obtained and renamed `new`.

We are already familiar with the `setHorizontalAlignment` method in line 12 from the example in 6.1.2. There is a pendant called `setVerticalAlignment` for the vertical positioning, which accepts the following values:

- `auto` or `automatic`
- `baseline`
- `bottom`
- `middle`
- `top`

In order to change the height of a row we need to obtain an entire row with the `getRowByIndex` method. In the parameter of the method the number of the row is stated (starting with 0, as usual).

In line 19 and 20 we change the height of some rows with the `setHeight` method. The value in the parameter is given in millimeters.

Columns can also be obtained. This is necessary to change their width. The corresponding methods are `getColumnByIndex` and `setWidth`. These are used in the lines 22 and 23.

Self-explanatory names are given to the variables representing the cell objects ranging from `A3` to `B6` in line 25 to 32.

In the following lines we are using the methods `setVerticalAlignment`, `setHorizontalAlignment` and `setCellBackgroundColor` again to format the cells and make the document look nicer.

Finally in line 41 the method `removeRowsByIndex` is called. This method enables us to erase one or more entire rows of a table. The first parameter is the number of the first row to delete (0 for the first row) and the second parameter is the number of rows to delete, in this case just the top row, which was empty.

Result

Figure 22 shows a screenshot of the resulting ODF spreadsheet document opened with OpenOffice Calc.

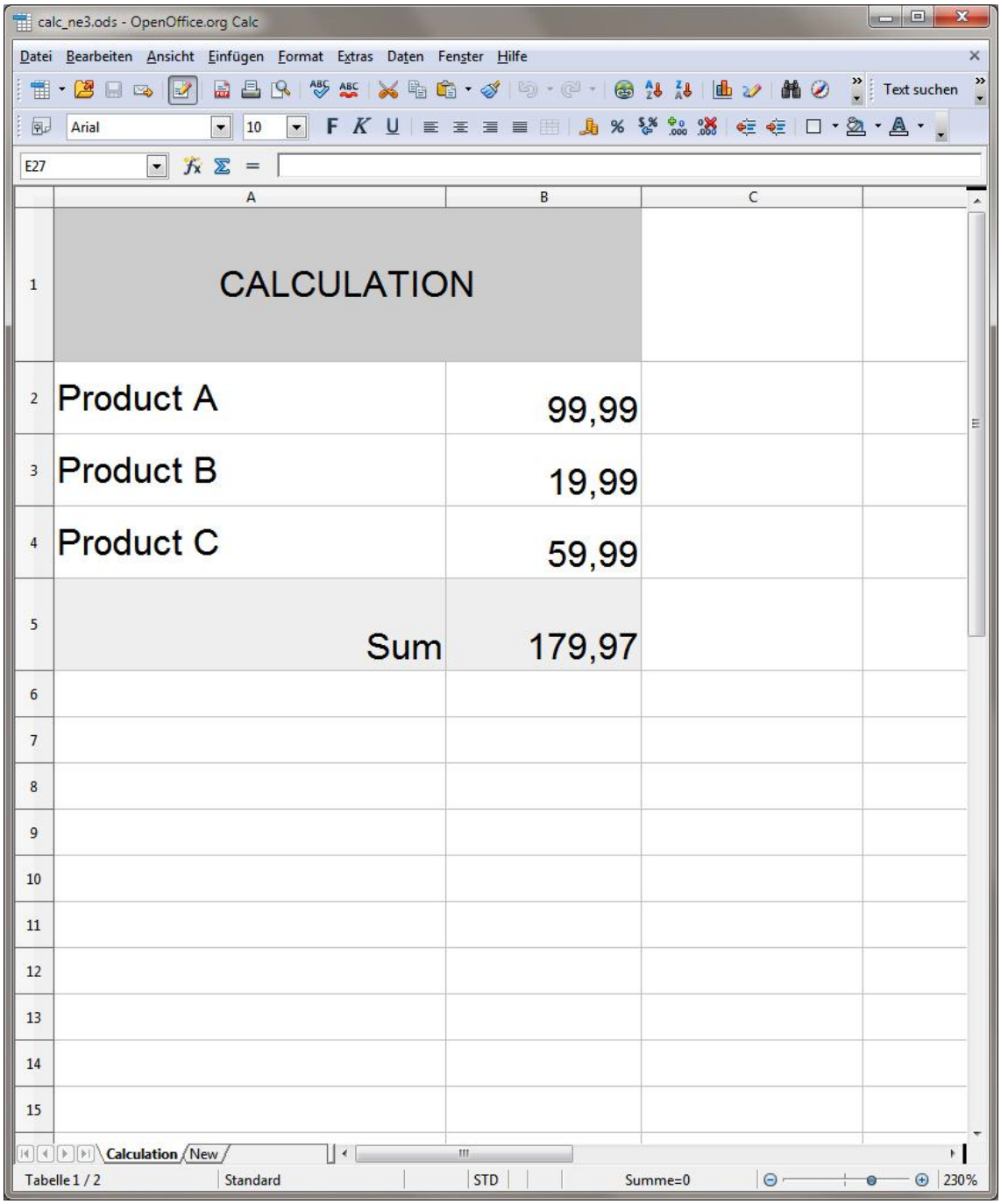


Figure 22 - Nutshell Example ODS 3 Resulting Document

6.3 ODP / Impress

Finally, this last chapter with nutshell examples deals with presentation documents with the file name ending ".odp". The first example shows the different underlying concept of a presentation document in contrast to a text or spreadsheet document.

6.3.1 Nutshell Example ODP 1

The first nutshell example of this section is, like the others in 6.1.1 and in 6.2.1, designed to show how to create a new document, add some text to it and save it.

Adding text to a text document is the easiest example as the document class itself comes with a method (actually two) to conveniently manage the task. A spreadsheet document needs information on the placement of the text, i.e. the worksheet and the coordinates of its table.

By contrast presentation documents are built as an empty, white space without a natural text flow. For adding text to it one needs to specify where on the foil the text should be positioned. In addition, text can only be added to containing elements, not directly to the foil itself. To illustrate this concept, we add some text to the foil in this example without specifying where.

Code

```
1  odfpd=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
2  PresentationDocument=odfpd~newPresentationDocument
3  Presentation=PresentationDocument~getContentRoot
4
5  foil=Presentation~getFirstChild
6
7  frame=foil~newDrawFrameElement
8  textBox=frame~newDrawTextBoxElement
9
10 text=textBox~newTextPElement
11 text~addContent("Hello World!")
12
13 PresentationDocument~save("impress_ne1.odp")
14
15 ::requires BSF.CLS
```

Figure 23 - Nutshell Example ODP 1 Code

Description

In the first line, we import the class `OdfPresentationDocument`. The code in the second line creates an empty presentation document. In line 3 the method `getContentRoot` is carried out. This method obtains the content root (class `OfficePresentationElement`), which represents the XML tag `office:presentation`.

In line 5 the method `getFirstChild` of the class `OfficePresentationElement` is called. The class inherits this method, which performs an XML navigation task, from `org.apache.xerces.dom.ParentNode`. It basically obtains the first foil of the presentation document.

In order to add some text to the foil a frame is created in line 7, a text box in line 8 and a text element, to which the text is later added, in line 10. This is a rather complex undergoing for a simple task compared to the other document types (see 6.1.1 and 6.2.1). Still the result (Figure 24) will not be pleasing as we don't specify where on the foil to add all these elements.

The variable `text` is of the class `OdfTextParagraph`. This class includes a method called `addContent`, which accepts a string as parameter.

The saving process is as simple as for the other document types. The document gets the name `impress_ne1.odp`. As usual the Bean Scripting Framework functionality for the Java support is loaded in the last line of the example.

Result

The code illustrated in Figure 23 results in the document shown in Figure 24. As we neither specified the exact place of the text nor the size of the text element or the containing elements we get this unwanted result where the text is placed at the top left and the letters are written vertically. These mistakes are corrected in the next example in 6.3.2.

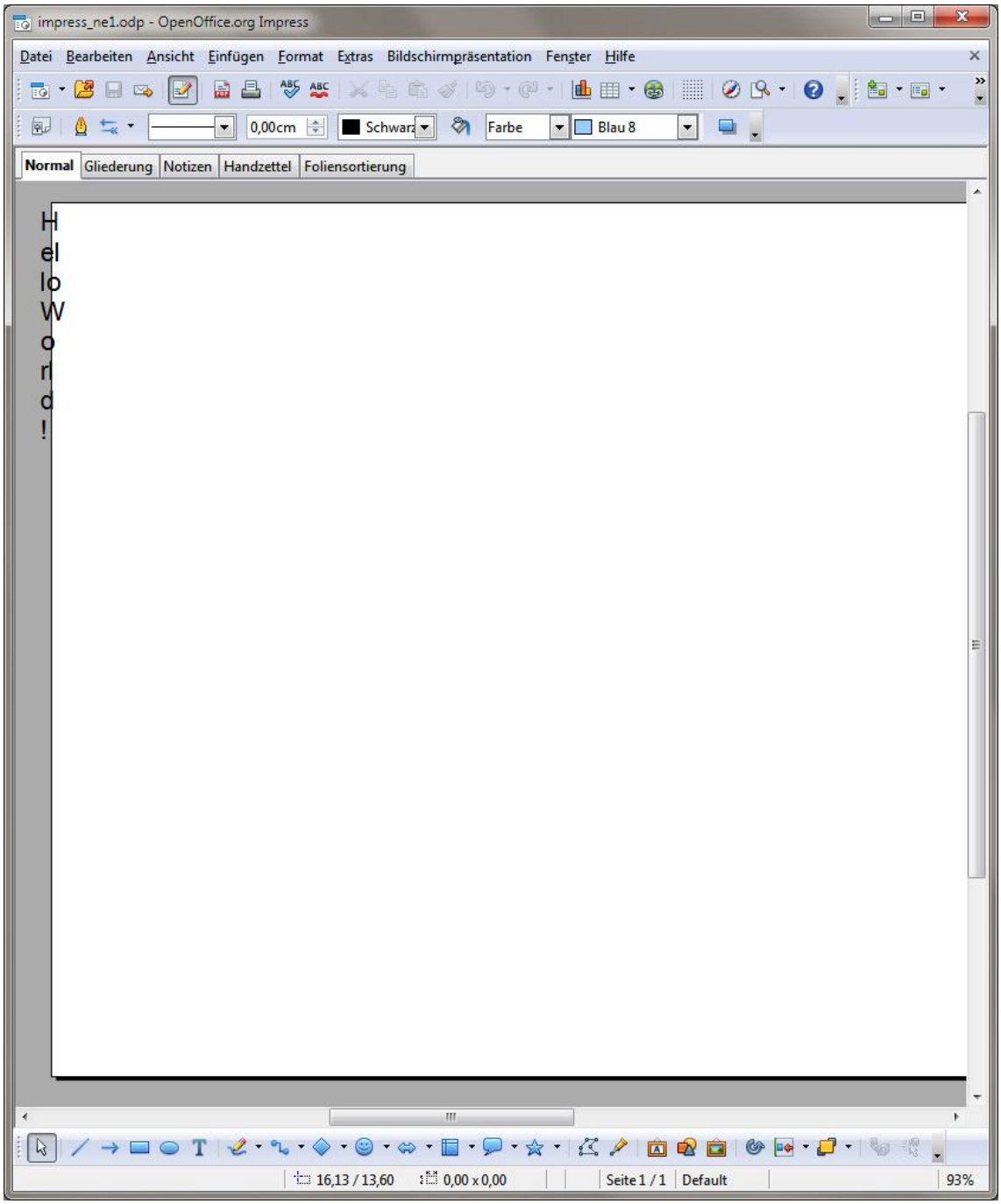


Figure 24 - Nutshell Example ODP 1 Resulting Document

6.3.2 Nutshell Example ODP 2

In 6.3.1 we saw how to add text to a presentation document. Although we used three different elements for this purpose the result was still unsatisfactory. The text was displayed partly outside the foil and only one character was placed in each line.

This example shows how elements (containing text, images or other objects) are correctly positioned on a slide and how the width and height of these elements can be specified.

Some text is inserted in one frame element; an image is added to another frame element on a new slide. Additionally, the order of the slides is changed and finally the document is saved.

Code

```
1  odfpd=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
2  PresentationDocument=odfpd~newPresentationDocument
3  Presentation=PresentationDocument~getContentRoot
4
5  slide1=Presentation~getFirstChild
6
7  frame1=slide1~newDrawFrameElement
8  frame1~setSvgXAttribute("12cm")
9  frame1~setSvgYAttribute("3cm")
10 frame1~setSvgWidthAttribute("4cm")
11 frame1~setSvgHeightAttribute("3.5cm")
12
13 textBox=frame1~newDrawTextBoxElement
14
15 text=textBox~newTextPElement
16 text~addContent("Hello world!")
17
18 slide2=Presentation~newDrawPageElement("")
19
20 frame2=slide2~newDrawFrameElement
21 frame2~setSvgXAttribute("12cm")
22 frame2~setSvgYAttribute("5cm")
```

```
23 frame2~setSvgWidthAttribute("5cm")
24 frame2~setSvgHeightAttribute("7cm")
25
26 image=frame2~newDrawImageElement
27 image~newImage(.bsf~new("java.net.URI","bierfriedl.jpg"))
28
29 PresentationDocument~moveSlide(1,0)
30
31 PresentationDocument~save("impress_ne2.odp")
32
33 ::requires BSF.CLS
```

Figure 25 - Nutshell Example ODP 2 Code

Description

We are already familiar with the first five lines of the code. The necessary Java class is imported, a new document is created and the content root and the first foil of the presentation document is obtained.

The first frame element is assigned to the ooRexx variable named `frame1` in line 7.

In the next four lines (8 to 11) the following methods of the class `org.odftoolkit.odfdom.dom.element.draw.DrawFrameElement` are carried out to specify the position and the size:

- `setSvgXAttribute` and `setSvgYAttribute`
  These methods specify the position of the frame element. The distance to the left border of the foil (`setSvgXAttribute`) and the distance to the top of the foil (`setSvgYAttribute`) are given in centimeters.

- `setSvgWidthAttribute` and `setSvgHeightAttribute`
  These methods define the size of the frame element by stating the width and the height in centimeters. Slides are 28 centimeters wide and 21 centimeters high by default.

Now that the `FrameElement` is sufficiently described with these four values we can add a `TextBoxElement` (line 13) and a `TextPElement` (line 15) to it. The

actual text is inserted in line 16. In conclusion, adding text to a presentation slide is a far more complex task than adding it to a text document.

In line 18 a new slide is added to the document with the method `newDrawPageElement`. The name of a master page can be specified as a parameter. Our presentation document now contains two slides (named `slide1` and `slide2`).

A second `FrameElement` named `frame2` is created and added to `slide2` in line 20. Once again we specify the size and the position of the new element with the four methods described above (lines 21 to 24).

An image element is drawn in line 26. We are already familiar with the code in line 27 from the nutshell example in 6.1.3.

Among other useful functions like the `save` or the `loadDocument` method the class `org.odftoolkit.odfdom.doc.OdfPresentationDocument` also provides a convenient method to move slides and therefore to change the order of the slides. This method – named `moveSlide` – expects two parameters:

- The current index of the slide that need to be moved.
- The index of the destination position.

The first slide always has the index zero. So our code `moveSlide(1,0)` moves the second slide containing the image to the index 0 and therefore to the start of the presentation.

The code in line 31 saves our presentation document with the name `impress_ne2.odp`.

Result

Figure 25 is a screenshot showing the resulting ODP document opened with Apache OpenOffice Impress.

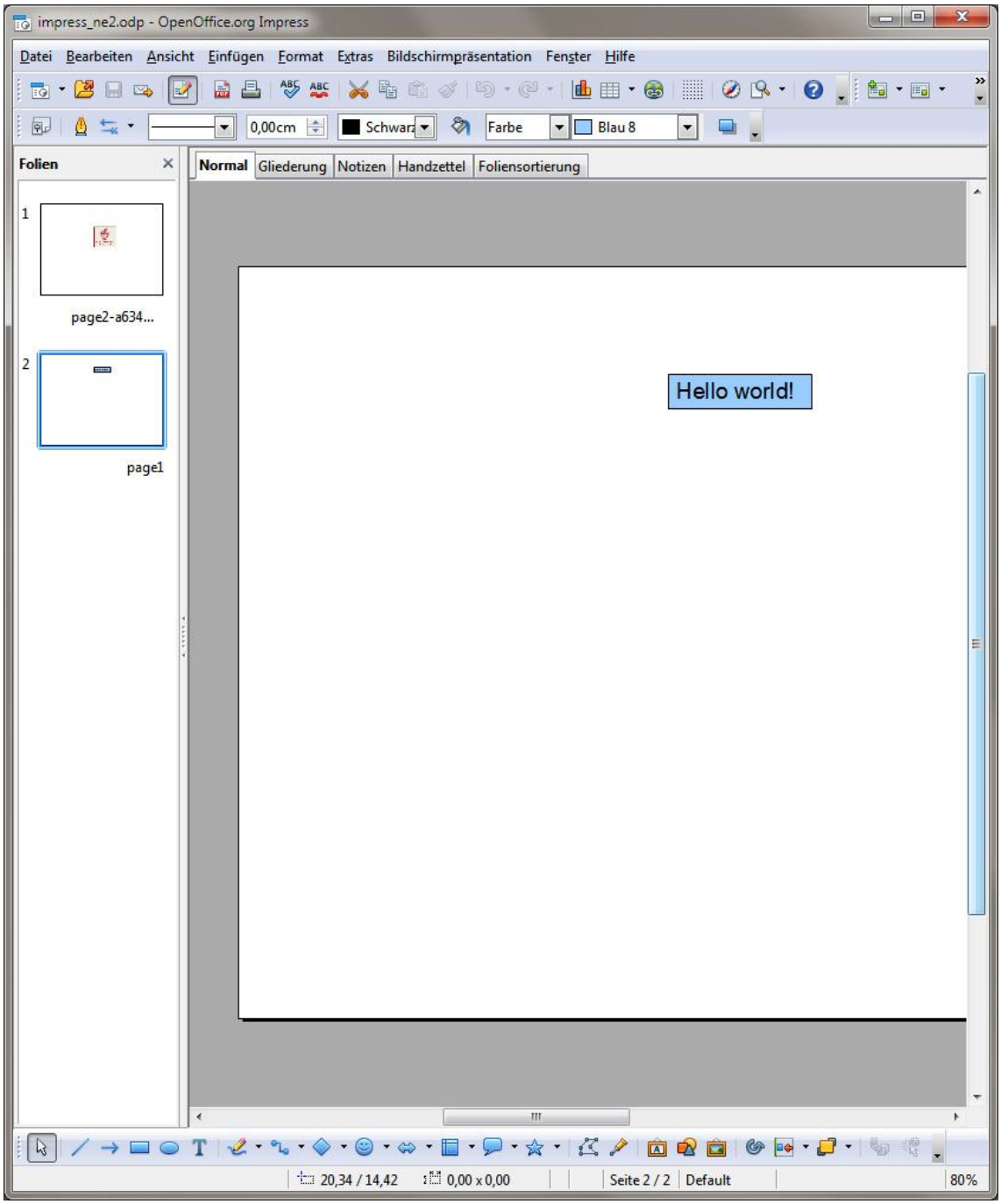


Figure 26 - Nutshell Example ODP 2 Resulting Document

6.3.3 Nutshell Example ODP 3

In this last nutshell example some other useful functions regarding presentation documents, like copying an entire slide from one presentation to another, are exemplified.

Code

```
1  odfpd=bsf.import("org.odftoolkit.odfdom.doc.OdfPresentationDocument")
2  PresentationDocument=odfpd~newPresentationDocument
3
4  PresentationDocument2=odfpd~loadDocument("impress_ne2.odp")
5
6  PresentationDocument~copyForeignSlide(1, PresentationDocument2, 0)
7  PresentationDocument~deleteSlideByIndex(0)
8
9  PresentationDocument~save("impress_ne3.odp")
10
11 ::requires BSF.CLS
```

Figure 27 - Nutshell Example ODP 3 Code

Description

Presentations are a communication and collaboration tool. It is therefore a common task to merge presentations together or to split them up, rearranging slides across multiple presentation documents. The ODF Toolkit comes with powerful features in this area.

In the first two lines a new document is created by importing the corresponding Java class and creating a new instance of said class.

In line 4 the presentation document created in 6.3.2 (named `impress_ne2.odp`) is loaded.

Line 6 includes a call of the method `copyForeignSlide`. This method provides a convenient way to exchange slides between different presentation documents. It expects three parameters:

- Destination index, i.e. the new position of the copied foil

- Source document
- Source index, i.e. the foils position in the source document

In this example the first slide of the document `PresentationDocument2` gets copied to the second position in the current document.

In line 9 the first slide, which is empty, is erased from the document with another useful and powerful method, the `deleteSlideByIndex` method. The number given in the parameter represents the index of the slide to be deleted.

Result

The code illustrated in Figure 27 results in the document shown in Figure 28.

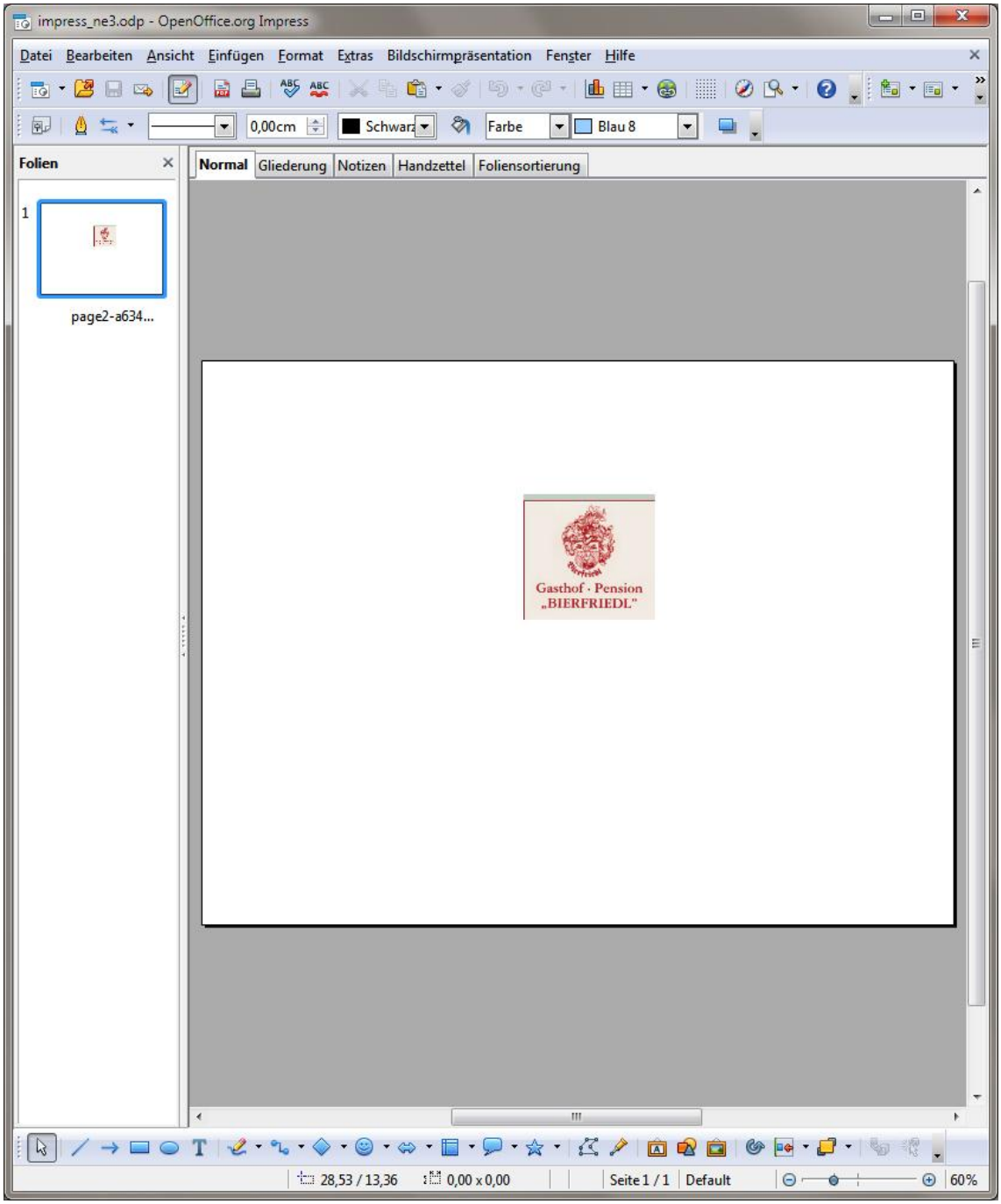


Figure 28 - Nutshell Example ODP 3 Resulting Document

7 Conclusion and Outlook

Office software is ubiquitous in modern business life and therefore essential for the efficiency and success of organizations and businesses. It is among the most commonly used types of computer programs. For letters and memos text documents are created. Calculations are done with spreadsheet documents and presentations are commonplace in meetings and at trade fairs.

Despite the high market share of Microsoft with its Office Suite there is a powerful and open source alternative, which mitigates disadvantages of proprietary solutions like licenses or obstacles for customization: Apache OpenOffice.

Due to the frequent usage of office software there is demand for automation of simple or repetitive tasks like serial letters. Apache OpenOffice comes with feature rich API's: the Simple API and the ODFDOM API. At the time of writing there are discussions among the developers on the coexistence of these two programming interfaces. The API's are written in Java, bundled together and distributed in the Apache ODF Toolkit.

With the Bean Scripting Framework for ooRexx it is surprisingly easy to use the Toolkit without having to deal with the painful difficulties of the Java syntax. As exemplified in this paper, developers and IT departments can easily create and edit text, spreadsheet or presentation documents.

Now that the free OpenOffice moved to the Apache Software Foundation it has a bright future ahead. With further improvements of the API's more features and functions are likely to be implemented.

8 References

[ADOA12] Apache Incubator: ODFDOM - The OpenDocument API, 2012
http://incubator.apache.org/odftoolkit/odfdom/index.html
[Accessed 24 February 2012]

[ADOL12] Apache Incubator: The ODFDOM Layers, 2012
http://incubator.apache.org/odftoolkit/odfdom/Layers.html
[Accessed 26 February 2012]

[ADOP12] Apache Incubator: ODFDOM Project Overview, 2012
http://incubator.apache.org/odftoolkit/odfdom/ProjectOverview.html
[Accessed 24 February 2012]

[AFMM09] Ashley, David W.; Flatscher, Rony G.; Hessling, Mark; McGuire, Rick; Miesfeld, Mark; Peedin, Lee; Tammer, Rainer; Wolfers, Jon: Open Object Rexx Programming Guide Version 4.0.0, 14.08.2009
http://www.oorexx.org/docs
[Accessed 4 March 2012]

[AODF12] Apache Incubator: Apache ODF Toolkit (incubating), 2012
http://incubator.apache.org/odftoolkit
[Accessed 4 February 2012]

[AOpe12] Apache Software Foundation: Apache OpenOffice (incubating), 2012
http://incubator.apache.org/openofficeorg/index.html
[Accessed 4 February 2012]

[ASCo12] Apache Software Foundation: Simple ODF Cookbook, 2012
http://simple.odftoolkit.org/cookbook
[Accessed 3 March 2012]

[ASim12] Apache Incubator: Simple API, 2012
http://incubator.apache.org/odftoolkit/simple/index.html
[Accessed 23 February 2012]

[AVal12] Apache Incubator: ODF Validator, 2012
http://incubator.apache.org/odftoolkit/conformance/ODFValidator.html
[Accessed 23 February 2012]

[AXer12] Apache Software Foundation: The Apache Xerces Project, 2011
http://xerces.apache.org
[Accessed 4 February 2012]

[AXSL12] Apache Incubator: ODF XSLT Runner, 2012
http://incubator.apache.org/odftoolkit/xsltrunner/ODFXSLTRunner.html
[Accessed 23 February 2012]

[AXST12] Apache Incubator: ODF XSLT Runner and ODF XSLT Runner Task Examples, 2012
http://incubator.apache.org/odftoolkit/xsltrunner/ODFXSLTRunnerExamples.html
[Accessed 21 March 2012]

[Ditc07] Ditch, Walter: XML-based Office Document Standards, *JISC Technology & Standards Watch 1.0,* 08 2007

[Flat04] Flatscher, Rony G.: Camouflaging Java as Object REXX, *International Rexx Symposium, Böblingen,* 05 2004

[Gosl05] Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad: The Java Language Specification, Addison-Wesley, 2005

[Leib09] Leiba, Barry: OpenDocument Format: The Standard for Office Documents, *IEEE Internet Computing Vol. 13,* pp. 83-87, 2009

[Maye11] Mayer, Günter: Scripting the ODF Toolkit (ODFDOM), 2011
http://wi.wu.ac.at/rgf/diplomarbeiten
[Accessed 30 January 2012]

[OASI06] OASIS: Open by Design - The Advantages of the OpenDocument Format (ODF), 2006
http://www.oasis-open.org/committees/download.php/21450/oasis_odf_advantages_10dec2006.pdf
[Accessed 27 January 2012]

[OASI07] OASIS: Open Document Format for Office Applications (OpenDocument) v1.1 Specification, 2007
http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html
[Accessed 9 February 2012]

[Rexx09] Rexx Language Association: About Open Object Rexx - Object-oriented Programming, 2009
http://www.oorexx.org/about.html
[Accessed 2 February 2012]

[RPro09] Rexx Language Association: Open Object Rexx: Programming Guide - Modeling Objects, 2009
http://www.oorexx.org/docs/rexxpg/x871.htm
[Accessed 3 March 2012]

[W3CX08] World Wide Web Consortium: Extensible Markup Language 1.0 Specification, 2008
http://www.w3.org/TR/REC-xml
[Accessed 14 February 2012]

[WCal12] Wikipedia: OpenOffice.org Calc, 2012
http://en.wikipedia.org/w/index.php?title=OpenOffice.org_Calc&oldid=470222583
[Accessed 6 February 2012]

[Weir06] Weir, Rob: A Proposal for an OpenDocument Developers Kit, 2006
http://www.robweir.com/blog/publications/ODF_Toolkit_Proposal.pdf
[Accessed 15 February 2012]

[WJav12] Wikipedia: Java (programming language), 2012
http://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=474158381
[Accessed 31 January 2012]

[WMet12] Wikipedia: Metadata, 2012
http://en.wikipedia.org/w/index.php?title=Metadata&oldid=473611207
[Accessed 9 February 2012]

[WMul12] Wikipedia: Multipurpose Internet Mail Extensions, 2012
http://en.wikipedia.org/w/index.php?title=MIME&oldid=473237813
[Accessed 9 February 2012]

[WOff12] Wikipedia: Office suite, 2012
http://en.wikipedia.org/w/index.php?title=Office_suite&oldid=469001631
[Accessed 26 January 2012]

[WOpe12] Wikipedia: OpenDocument technical specification, 2012
http://en.wikipedia.org/w/index.php?title=OpenDocument_technical_specification&oldid=445678724
[Accessed 7 February 2012]

[WPre12] Wikipedia: Presentation program, 2012
http://en.wikipedia.org/w/index.php?title=Presentation_program&oldid=468044768
[Accessed 6 February 2012]

[WREX12] Wikipedia: REXX, 2012
http://en.wikipedia.org/w/index.php?title=REXX&oldid=471379104
[Accessed 2 February 2012]

[WRGB12] Wikipedia: RGB color model, 2012
http://en.wikipedia.org/w/index.php?title=RGB_color_model&oldid=473759578
[Accessed 6 March 2012]

[WSpr12] Wikipedia: Spreadsheet, 2012
http://en.wikipedia.org/w/index.php?title=Spreadsheet&oldid=482178720
[Accessed 17 March 2012]

[WWor12] Wikipedia: Word processor, 2012.
http://en.wikipedia.org/w/index.php?title=Word_processor&oldid=473138864
[Accessed 26 January 2012]