DocxFactory
Version 1.2.14
C++
Tutorial

Written by Alon Blich

Table of Contents

Linux Installation

This chapter shows you how to install and setup DocxFactory on Linux.

1. Go to www.docxfactory.com and download the Linux installation package.

   For the Linux 32bit version download DocxFactoryLinux32.tar.gz.
   For the Linux 64bit version download DocxFactoryLinux64.tar.gz.

2. Before starting, it is recommended that you login as root so you have all the necessary permissions to perform the installation.

   Enter the commands below in the command line.

```
$ su -
```

3. Extract the DocxFactory/ directory into the /opt directory.

   Enter the commands below in the command line.

```
# cd /opt

# tar -zxvf DocxFactoryLinux32.tar.gz
```

   Note: If there is an older version installed then delete the DocxFactory directory first (do not overwrite it).

4. Add the DocxFactory/lib/ directory to the list of library directories to search when a program is run.

   Either permanently using ldconfig:

```
# echo /opt/DocxFactory/lib/ > /etc/ld.so.conf.d/DocxFactory.conf

# chmod 666 /etc/ld.so.conf.d/DocxFactory.conf

# ldconfig
```

   Or temporarily using the LD_LIBRARY_PATH environment variable before the program is run:

```
# export LD_LIBRARY_PATH=/opt/DocxFactory/lib:$LD_LIBRARY_PATH
```

   Note: You will need to restart all processes that use the DocxFactory.so (or simply restart the Computer) for the changes to take effect.

5. To compile C++ programs using DocxFactory with the GNU C++ Compiler:

    1. Add the DocxFactory/include/ directory to the list of include directories.

    2. Link to the libDocxFactory.so library.

    3. Compile and link to WordProcessingCompiler.cpp and WordProcessingMerger.cpp from the /DocxFactory/src/ directory.

    See example below.

```
# g++ -c <program.cpp> \
/opt/DocxFactory/src/WordProcessingCompiler.cpp \
/opt/DocxFactory/src/WordProcessingMerger.cpp \
-I/opt/DocxFactory/include

# g++ -o <executable> <program.o> \
WordProcessingCompiler.o WordProcessingMerger.o \
-L/opt/DocxFactory/lib -l:libDocxFactory.so
```

Note: The backslash "\" at the end of line causes the command to continue in the next line. The command is broken into several lines to fit in the page. You can write the command in a single line without backslashes.

Windows Installation

This chapter shows you how to install and setup DocxFactory on Windows.

1. Go to www.docxfactory.com and download the Windows installation .ZIP file.

   For the Windows 32bit version download DocxFactoryWin32.zip.
   For the Windows 64bit version download DocxFactoryWin64.zip.

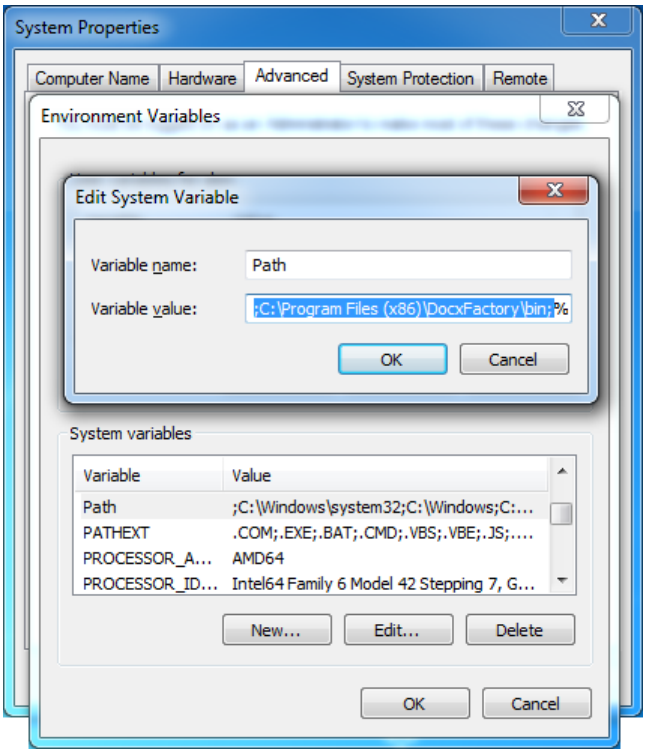
2. Extract the DocxFactory\ directory into the Program Files\ directory.

   The DocxFactoryWin32.zip is extracted to the C:\Program Files (x86)\ directory.
   The DocxFactoryWin64.zip is extracted to the C:\Program Files\ directory.

   Note: If there is an older version installed then delete the DocxFactory directory first (do not overwrite it). You may need to shutdown all processes using the DocxFactory.dll or the file will be locked and undeletable.

3. Add the DocxFactory\bin\ directory to the Windows PATH used to search for executables and libraries when a program is run.

   1. Right click My Computer and click Properties.
   2. In the System Properties window, click on the Advanced tab.
   3. In the Advanced section, click the Environment Variables button.
   4. In the Environment Variables dialog box, select the PATH variable in the Systems Variable section, click the Edit button and add to the PATH the "C:\Program Files (x86)\DocxFactory\bin;" directory with a semicolon to separate it from the other directories (see picture below).

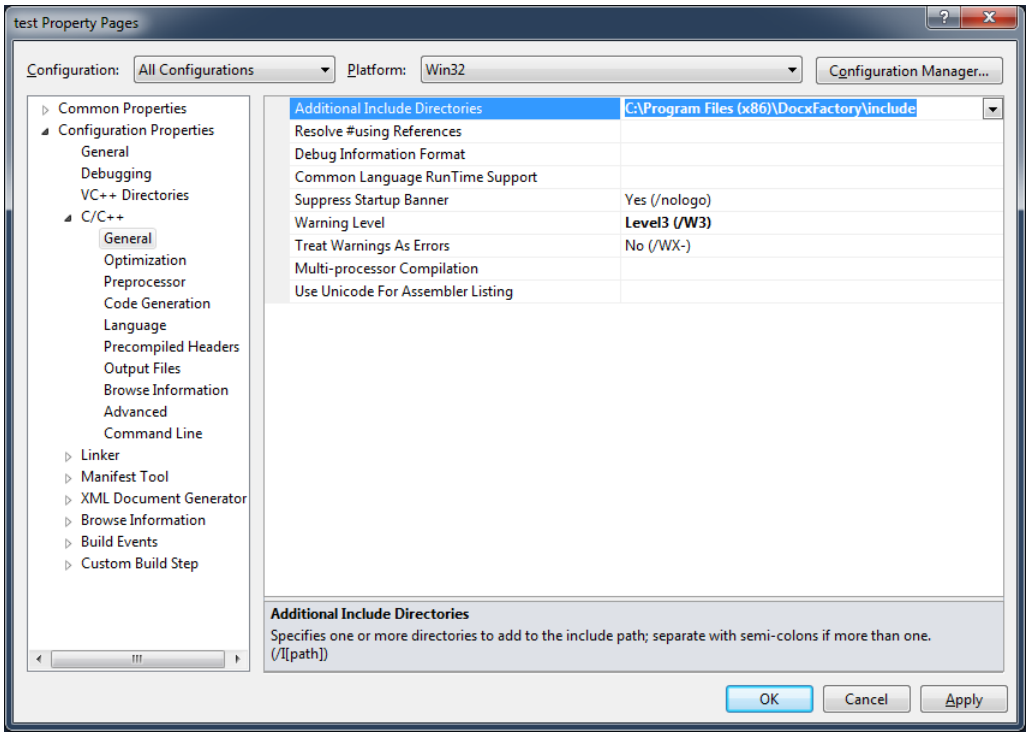


Note: You will need to restart all processes that use the DocxFactory.dll (or simply restart the Computer) for the changes in the PATH environment variable to take effect.

4. To compile C++ programs using DocxFactory with Visual Studio:

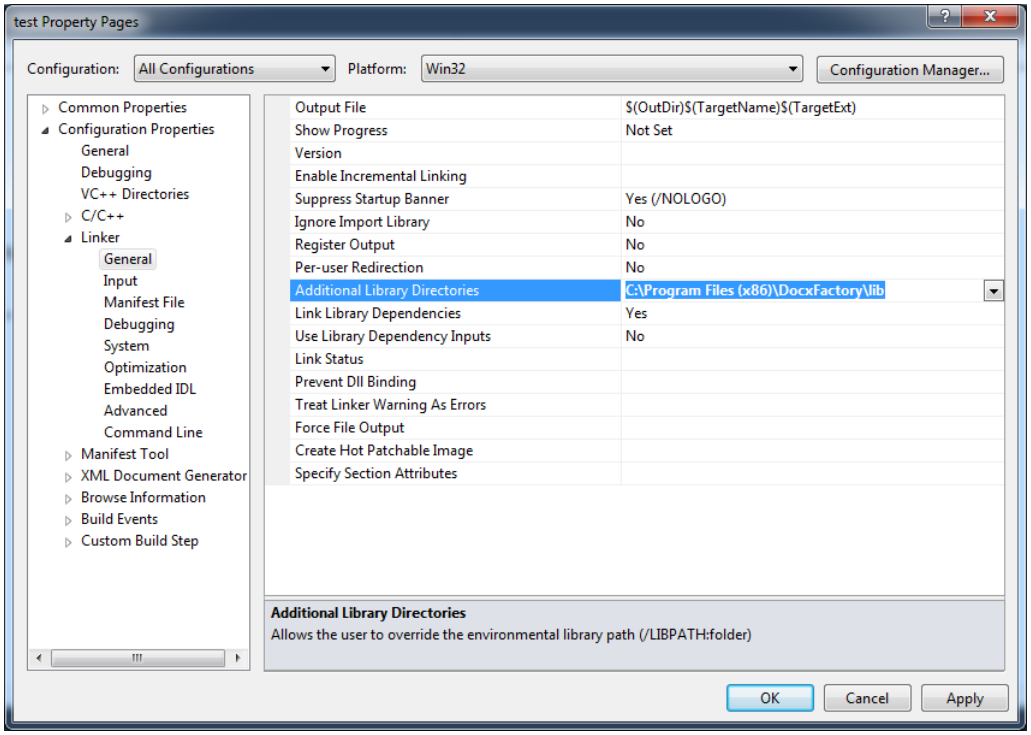
    1. Add the DocxFactory/include/ directory to the list of include directories.

        In the project properties, in the C/C++ properties, in the General properties, add to the Additional Include Directories the “C:\Program Files (x86)\DocxFactory\include;” directory with a semicolon to separate it from the other directories (see picture below).

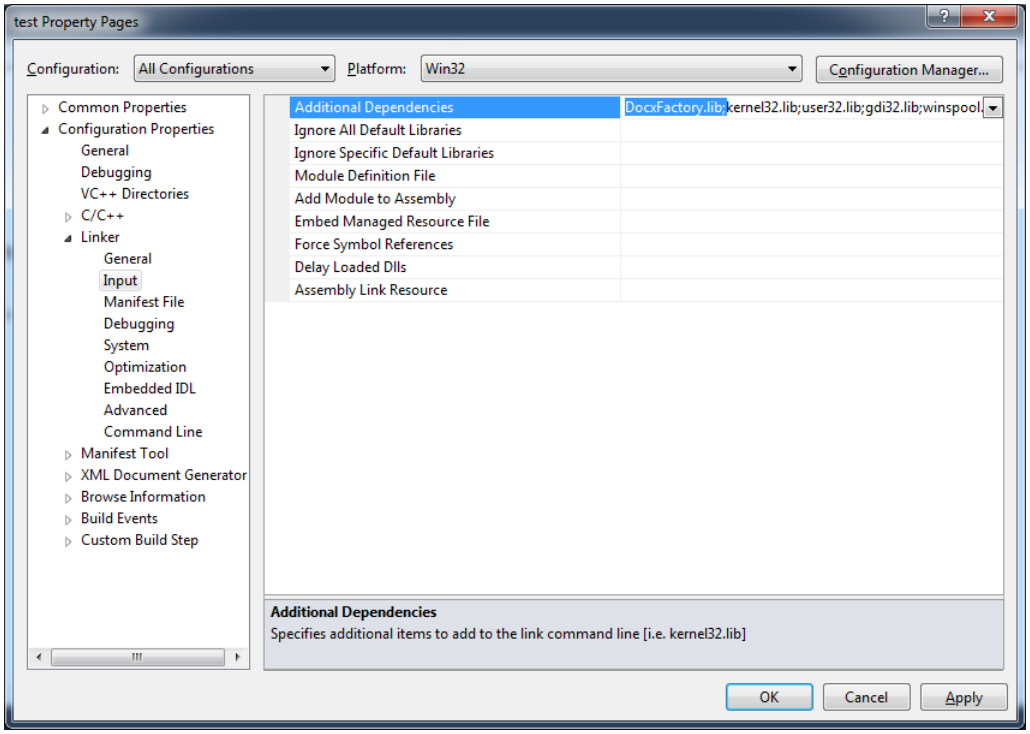


2. Link to the DocxFactory.lib library.

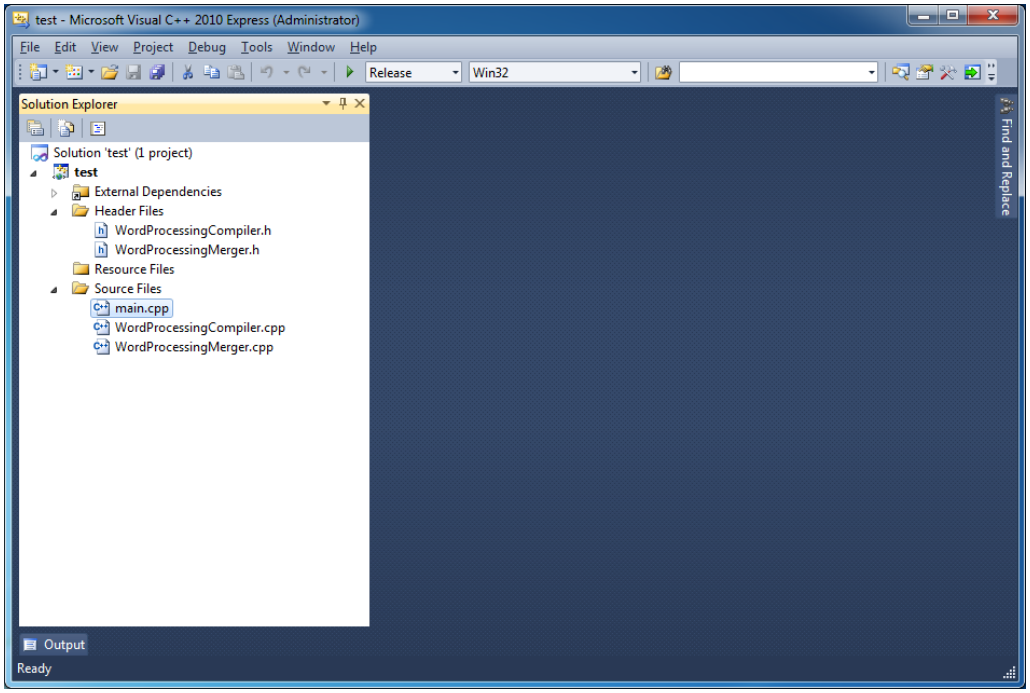
   In the project properties, change the Configuration to “All Configurations”. Then in the Linker properties, in the General properties, add to the Additional Library Directories the “C:\Program Files (x86)\DocxFactory\lib;” directory with a semicolon to separate it from the other directories (see picture below).



In the Linker properties, in the Input properties, add to the Additional Dependencies the “DocxFactory.lib;” library with a semicolon to separate it from the other libraries (see picture below).



3. Add the WordProcessingCompiler.cpp and WordProcessingMerger.cpp from the DocxFactory/src/ directory to your project.



Note: If the project uses precompiled headers then you will need to add #include "stdafx.h" to the beginning of the code exercises. For example:

```
#include "stdafx.h"
#include "WordProcessingMerger.h"

.
.
.

int _tmain(int argc, _TCHAR* argv[]) {

        .
        .
        .

}
```

Printing and Conversion Tools Installation

DocxFactory can only creates .DOCX files directly. To create .PDF, .HTML or other file formats, DocxFactory first creates a .DOCX file and then converts the file using conversion tools. Likewise to print, DocxFactory uses printing tools.

For the printing and conversion tools DocxFactory uses:

- Open Office (or LibreOffice) on Linux.
- Microsoft Office on Windows (Open Office can also be used on Windows as a free alternative).

Note: Installing printing and conversion tools is optional. If no printing and conversion tools are installed then you will still be able to generate .DOCX files but not print or save as other file formats.

Note: Open Office is not 100% compatible with Microsoft Word .DOCX file format meaning that it may not look exactly the same in Open Office and may never will even in future versions of Open Office simply because of the sheer size of the file format and all its different options. To get the best results it is recommended to use Microsoft Office for printing and converting.

Note: Future versions of DocxFactory will also support designing and creating Open Office .ODT files directly that will give the best results when printing and converting on Linux.

To use Open Office you will also need to install the Open Office SDK (see link below).

http://api.libreoffice.org/docs/install.html

The ODF Converter Integrator is an add on for Open Office that greatly improves Open Office compatibility with Microsoft Office file formats. If you are using Open Office then it is highly recommended that you also install the ODF Converter Integrator (see link below).

http://katana.oooninja.com/w/odf-converter-integrator

Introduction

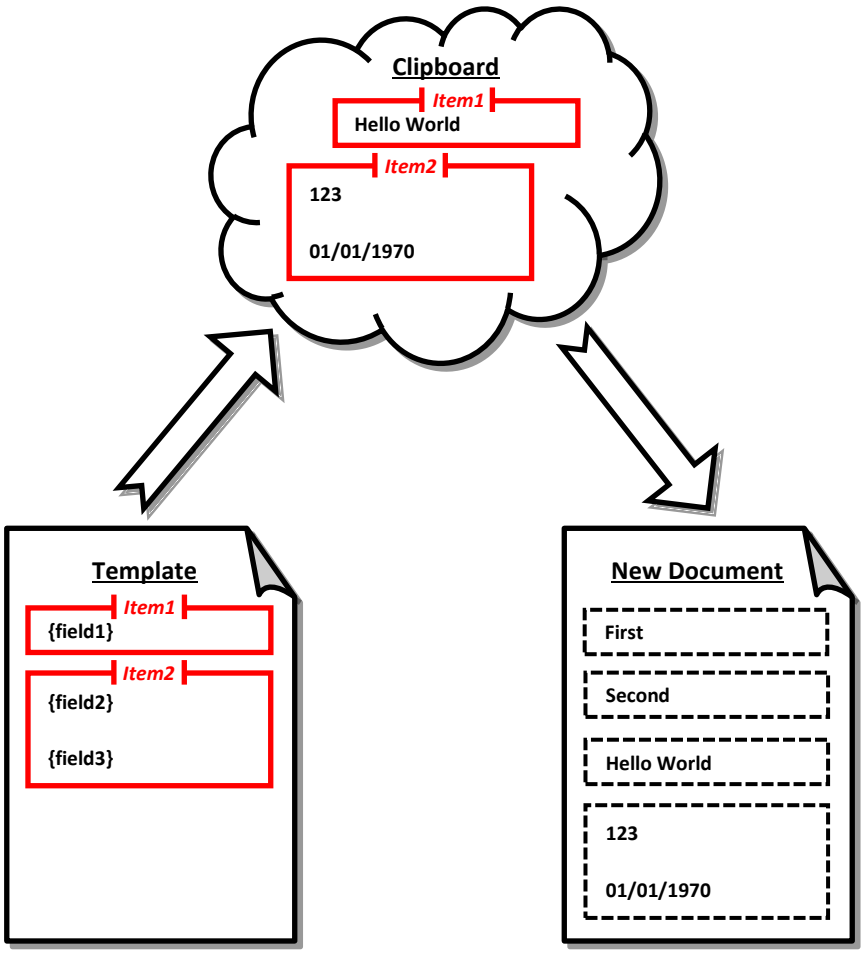
The general steps to create .DOCX files using DocxFactory are (see picture below):

1. You start by designing a template (a regular .DOCX file) in Microsoft Word (or another office suite like OpenOffice.org).

2. You then mark parts of the template using bookmarks, called items and add fields to your items by typing the field names in squiggly brackets. For example: {MyField}.

3. When creating a new document from a template, DocxFactory first copies all the items with their fields into the DocxFactory clipboard.

4. You then merge the data with the template by setting the field values of the items in the clipboard and pasting the items with the set field values to the end of the new document.



Note: DocxFactory can only create .DOCX files directly. To create .PDF, .HTML or other files, DocxFactory first creates a .DOCX file and then converts the file using Microsoft Office or OpenOffice.org running in the background. Future versions of DocxFactory will also support creating Microsoft Excel and OpenOffice.org files directly.

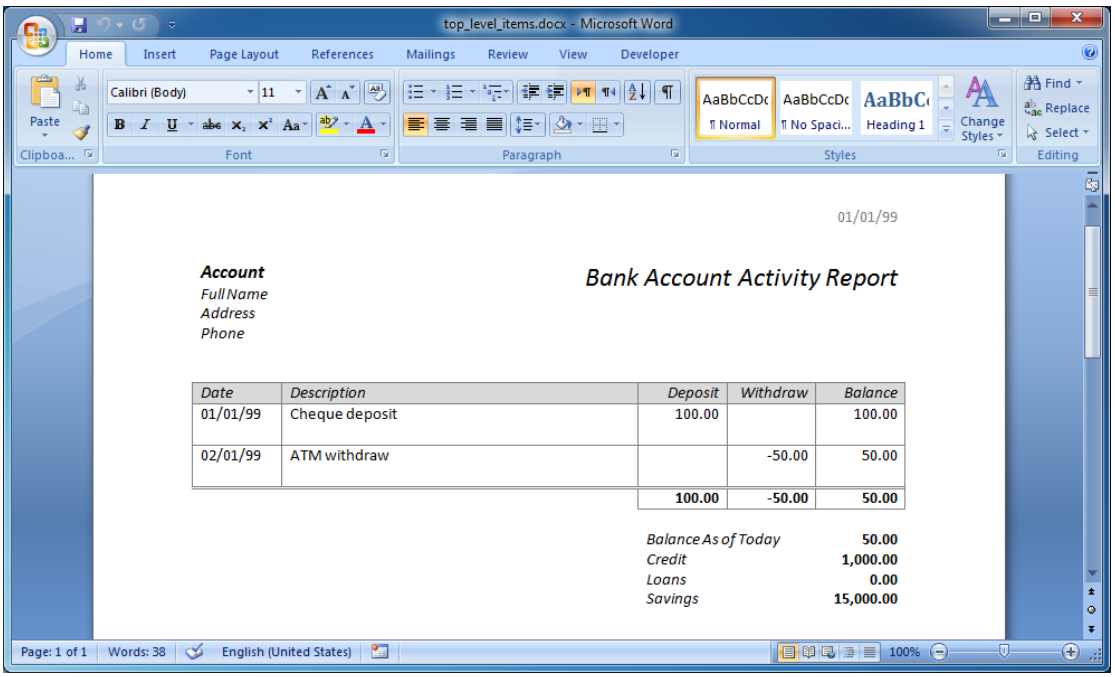
Top Level Items

Top level items are items that are not inside any other item. When a top level item is pasted it starts in a new page.

Top Level Items Exercise

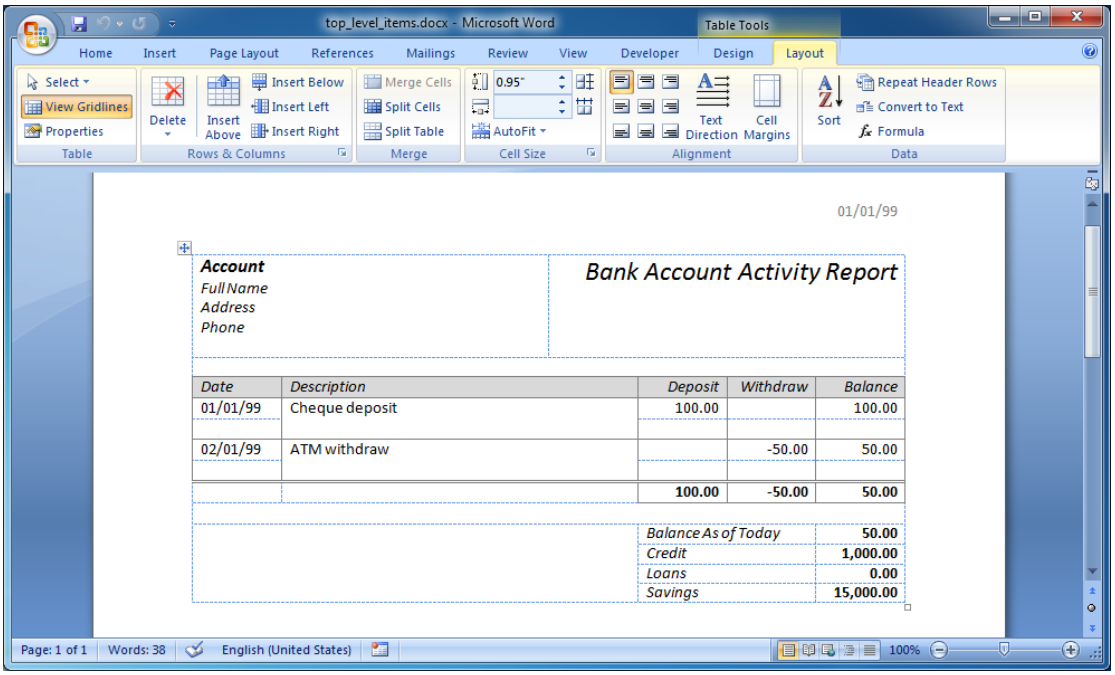
The following exercises in this tutorial walk you through creating a Bank Account Activity Report.

1. Open top_level_items.docx in the DocxFactory/exercises/templates/ directory (see picture below).



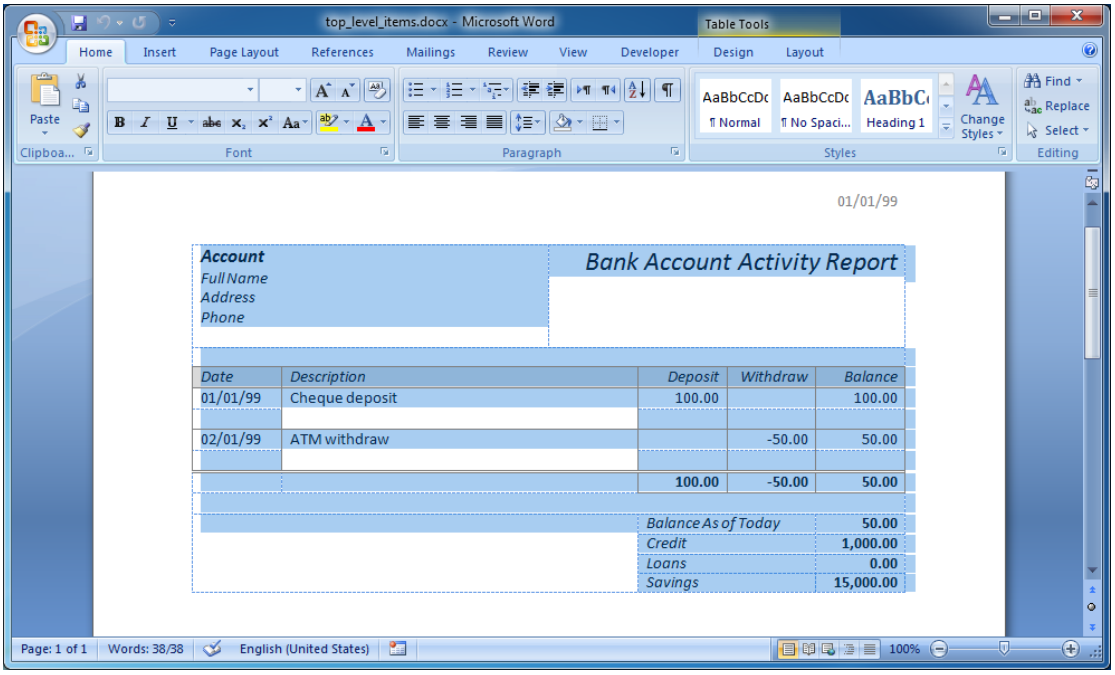
2. The recommended way to divide your layout into parts, in Microsoft Word, is using tables without borders. The table only purpose is layout and is completely hidden.

   To see the table, click inside the table and the Design and Layout ribbons will show. In the Layout ribbon, select View Gridlines (see picture below).



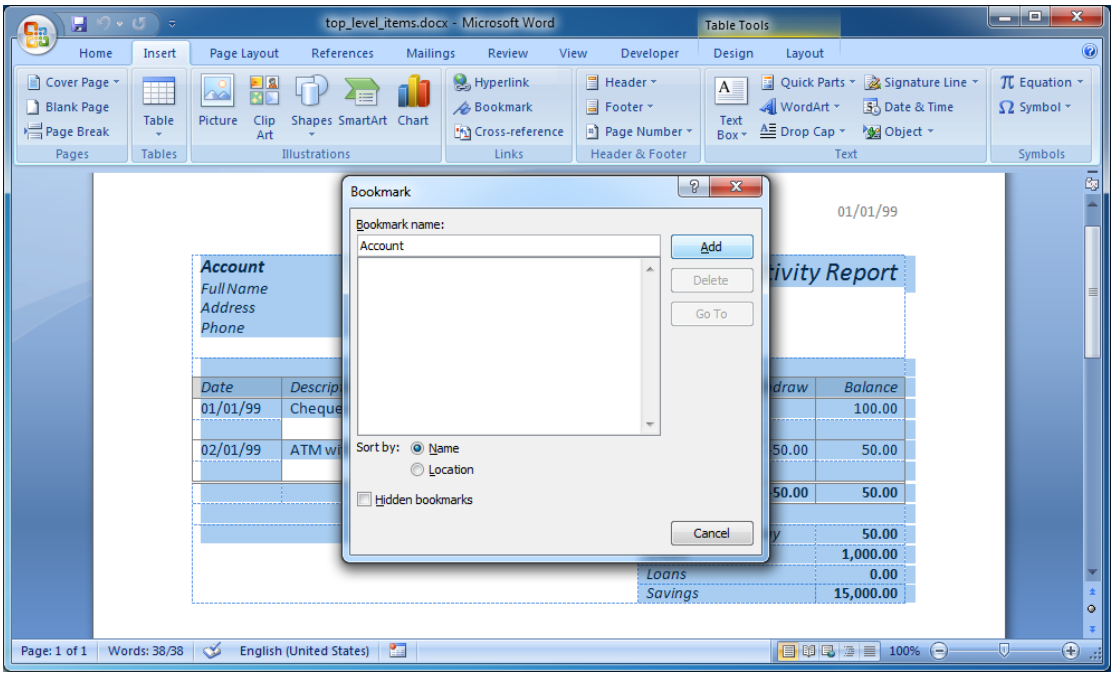
3. To create an item, first highlight the area of the item.

   Highlight the entire report (see picture below).



4. Then insert a bookmark for the highlighted area with the name of the item.

   In the Insert ribbon, press the Bookmark button to open the Bookmark dialog box. In the Bookmark name, type “Account” and press the Add button (see picture below).

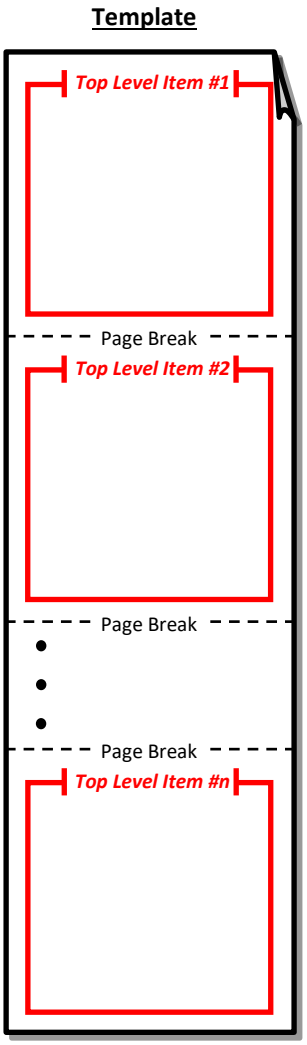


Note: A table cell cannot be marked as an item so it will not be possible to create a table with rows that have a different number of cells. A table row is the smallest part of a table that can be marked as an item. If a table cell is marked as an item then the area is expanded to mark the whole table row. Future versions will support marking columns for horizontal pasting.

Note: In most cases, you’ll find it easier if you create items that map closely to your data structure. For example: if you have a Bank Account table with the Bank Account’s data, you can create a Bank Account item with the Bank Account’s fields.

A top level item should start after the previous page break (or the start of the document) and end right before the next page break (or the end of the document) where the next top level item starts (see picture below).

For 2 reasons: 1. The start of a new page at the start of the top level item in the template is the start of a new page when the top level item is pasted in the new document. 2. Everything in the template should be in an item.



Even though you can: not put everything in items, put more than one top level item between page breaks, put page breaks in the middle of an item or not put any items in the template this is not the intended use and although the template will still compile and the program will still run you may not get the results you intended.

5. Before you can use the template you must compile the .DOCX file to a .DFW file - (D)ocx (F)actory (W)ordprocessing template file.

Copy and run the code below.

```
#include "WordProcessingCompiler.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingCompiler& l_compiler =
            WordProcessingCompiler::getInstance();

        time_t l_start = clock();

        l_compiler.compile(
            "/opt/DocxFactory/exercises/templates/top_level_items.docx",
            "/opt/DocxFactory/exercises/templates/top_level_items.dfw");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code compiles the template file. If the template is successfully compiled then the time it took to compile is displayed. If an exception is thrown then the description of the exception is displayed.

Note: The code in the exercises uses the Linux installation directories for running on Linux. To run the code on Windows, use the Windows installation directories.

Note: You can get the source .DOCX file from the compiled .DFW file, in case the .DOCX file is lost. The .DFW file is actually a .ZIP file. You can open the .ZIP file (on Windows you can add a .ZIP extension to the end of the file name and open the .ZIP file with the Windows Explorer) and find the .DOCX file inside in the root directory with a # prefix.

The code introduces the WordProcessingCompiler singleton and its compile function (see details below).

DocxFactory::WordProcessingCompiler Singleton

Provides functions for compiling template files.

Header File:

```
#include "WordProcessingCompiler.h"
```

Member Functions:

```
getInstance   - Returns the singleton object instance.

Compile       - Compiles a .DOCX file to a .DFW file.

setTempDir    - Sets the temporary directory used for saving temporary
                files.

getTempDir    - Returns the temporary directory.

getWorkDir    - Returns the current working directory used for resolving
                relative paths.
```

DocxFactory::WordProcessingCompiler::compile Function

Compiles a .DOCX file to a .DFW file.

Declaration**:**

```
void compile(const string& p_sourceFile, const string& p_targetFile);
```

Parameters:

```
p_sourceFile - The source .DOCX file.
p_targetFile – The target .DFW file.
```

Notes:

- Relative paths are resolved relative to the current working directory. For example: “dir/template.docx” is resolved to “[working directory]/dir/template.docx”.
- The target file must end with a “.dfw” file extension. If it does not then a “.dfw” file extension is added to the end of the file name.
- If the target file is blank then the source file is used.

6. Create a new document from the template we created, paste the top level item and save the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/top_level_items.dfw");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Account");
        }

        l_merger.save("/tmp/top_level_items.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code introduces the WordProcessingMerger singleton and its load, paste and save functions (see details below).

DocxFactory::WordProcessingMerger Singleton

Provides functions for merging templates with data.

Header File:

```
#include "WordProcessingMerger.h"
```

Member Functions:

```
load                - Creates a new document from a .DFW template.

save                - Saves the new document to a .DOCX file or
                      possibly converts to a .PDF, .HTML or other.

print               - Prints the new document.

close               - Closes an open document. Used for checking fields
                      and items only without saving.

setClipboardValue   - Sets a field value of an item in the clipboard.

setChartValue       - Sets a value in the chart field values matrix by
                      category and series.

paste               - Pastes an item from the clipboard with its field
                      values to the new document.

merge               - Merges XML or JSON with the template.

getItems            - Returns a comma separated list of all the items
                      in the template.

getFields           - Returns a comma separated list of all the fields
                      in the template.

getItemFields       - Returns a comma separated list of all the fields
                      in an item.

getItemParent       - Returns the parent of an item.

setUpdateTocMethod  - Sets the table of contents automatic update
                      method.

setCodePage         - Sets the code page for strings passed to
                      setClipboardValue function.

setNumFracSep       - Sets the number fraction separator sign.

setNumThSep         - Sets the number thousand separator sign.

setDateFormat       - Sets the date format.

setYearOffset       - Sets the year offset for 2 digit years.

setFirstWeekDay     - Sets the first week day.

setWeekDayNames     - Sets the week day names.
```

```
setMonthNames          - Sets the month names.

getCodePage            - Returns the code page.

getNumFracSep          - Returns the number fraction separator sign.

getNumThSep            - Returns the number thousand separator sign.

getDateFormat          - Returns the date format.

getYearOffset          - Returns the year offset.

getFirstWeekDay        - Returns the first day of the week.

getWeekDayFullNames    - Returns the full week day names.

getWeekDayShortNames   - Returns the short week day names.

getMonthFullNames      - Returns the full month names.

getMonthShortNames     - Returns the short month names.

setTempDir             - Sets the temporary directory used for saving
                         temporary files.

getTempDir             - Returns the temporary directory.

getWorkDir             - Returns the current working directory used for
                         resolving relative paths.
```

DocxFactory::WordProcessingMerger::load Function

Creates a new document and cuts all the template items into the DocxFactory clipboard.

Declaration**:**

```
void load(const string& p_templateFile);
```

Parameters:

```
p_templateFile - The template .DFW file.
```

Notes:

- Relative paths are resolved relative to the current working directory. For example: “dir/template.dfw” is resolved to “[working directory]/dir/template.dfw”.

DocxFactory::WordProcessingMerger::paste Function

Pastes an item from the DocxFactory clipboard with all its field values on to the new document.

Declaration**:**

```
void paste(const string& p_itemName);
```

Parameters:

```
p_itemName – The item name.
```

Notes:

- The item names are not case sensitive. For example: “Account” and “account” refer to the same item.

DocxFactory::WordProcessingMerger::save Function

Saves the new document out to a file. After saving, the document is closed and no more data can be merged until a new template is loaded.

Declaration**:**

```
void save(const string& p_targetFile);
```

Parameters:

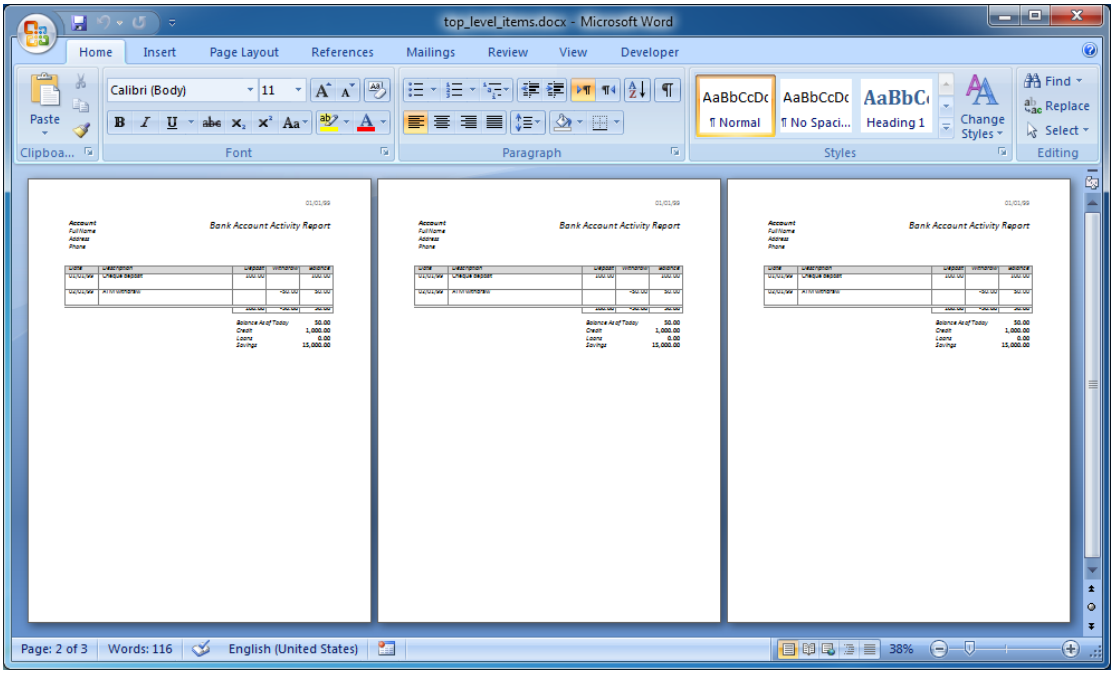
```
p_targetFile – The target file.
```

Notes:

- If the target file is not a .DOCX file (for example: .PDF file) then DocxFactory first saves the document as a .DOCX file and then converts the file to the target file.
- Relative paths are resolved relative to the current working directory. For example: “dir/template.docx” is resolved to “[working directory]/dir/template.docx”.

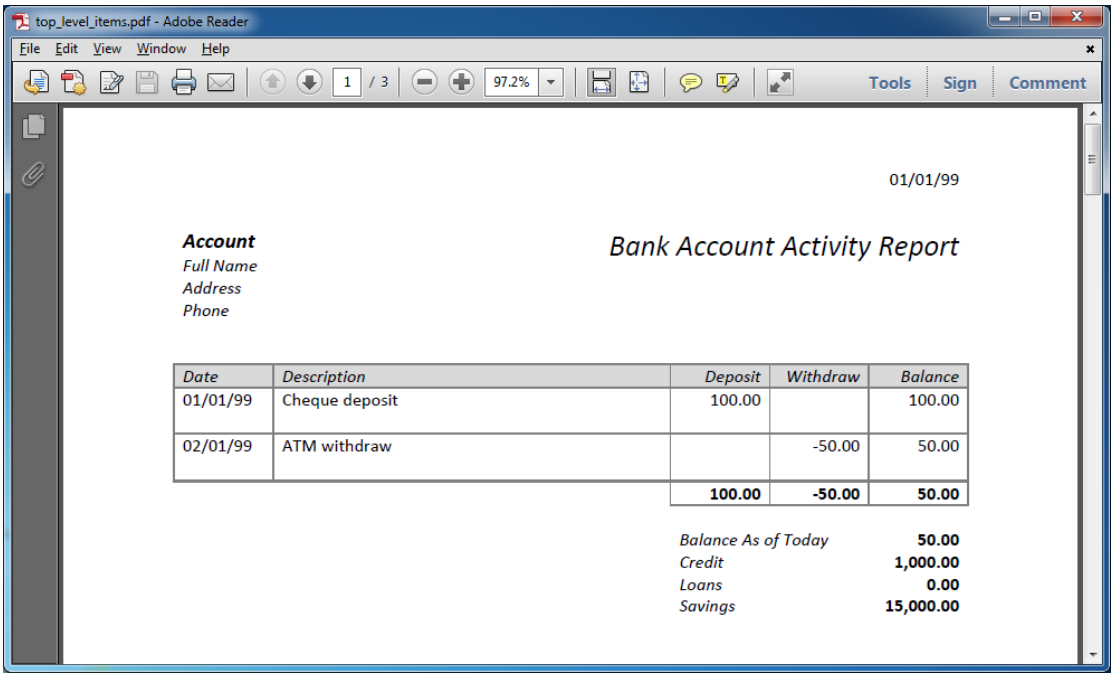
7. Open the created .DOCX file.

   As you can see, every top level item starts in a new page (see picture below).



8. Save as .PDF file.

   In your code, change the save file to "/tmp/top_level_items.pdf". Run and open the file (see picture below).



Note: DocxFactory can only create .DOCX files directly. To create .PDF, .HTML or other file formats, DocxFactory first creates a .DOCX file and then converts the file using conversion tools. Likewise to print, DocxFactory uses printing tools.

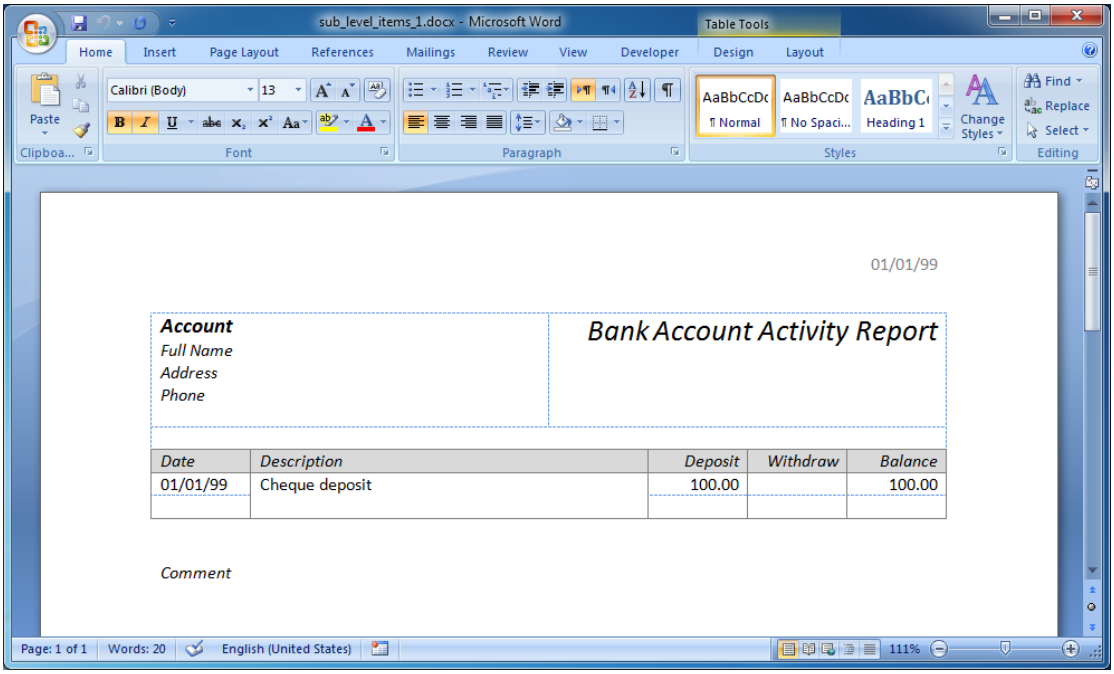
You must have printing and conversion tools installed to print and convert to other file formats (see the chapter on printing and conversion tools installation in the beginning for details).

Sub Level Items

A sub level item is an item that is inside another item (sub level items can have sub level items inside them. There is no limit to the number of levels of nested sub level items you can have). When a sub level item is pasted it is repeated in its place.

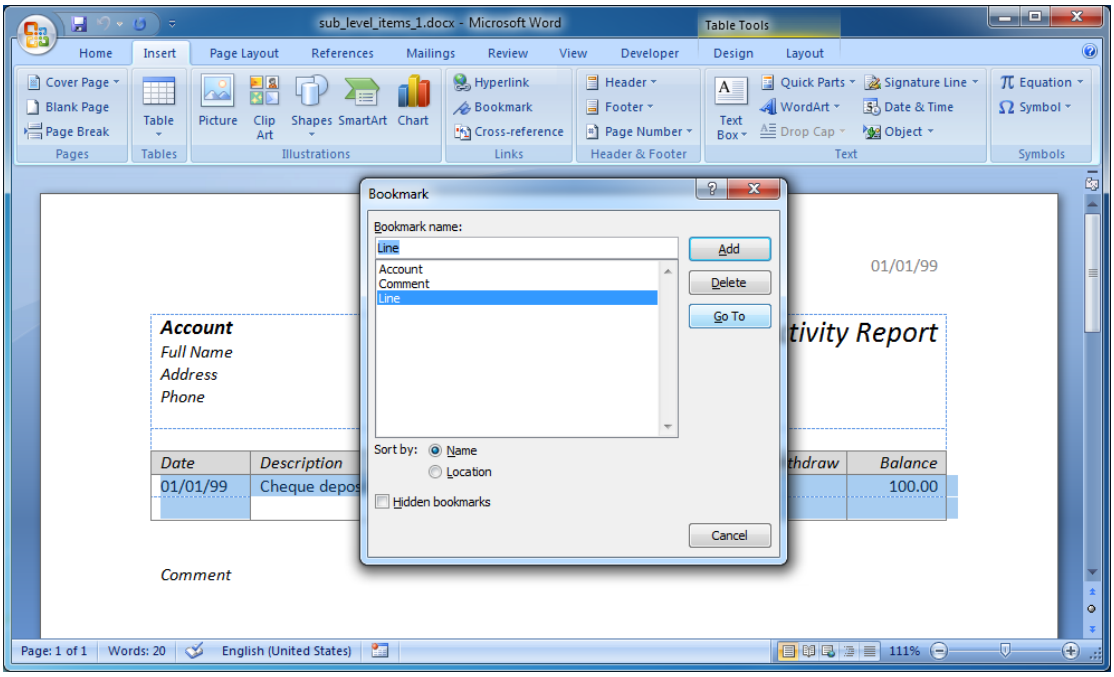
Sub Level Items 1 Exercise

1. Open sub_level_items_1.docx in the DocxFactory/exercises/templates/ directory (see picture below).



2. Open the Bookmark dialog box to view the items in the template.

   There are 3 bookmarks for the items in the template. Highlight each bookmark and press the Go To button to view the highlighted area (see picture below).



3. Compile the template. You can also compile the template from the command-line.

See example below.

```
$ word-processing-compiler <source file> <target file>
```

Note: There is also a word-processing-merger command line tool to merge templates and XML or JSON. To show the tools options, launch the tools with the --help option.

4. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;

int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/sub_level_items_1.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Line");
            l_merger.paste("Comment");
        }

        l_merger.save("/tmp/sub_level_items_1.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

In general, when pasting an item the item is added to the end of the document. When a top level item is pasted it is simply added to the end of the document. When a sub level item is pasted it is added inside its parent in the end of the document because a sub level item must be added inside its parent and cannot be added by itself.

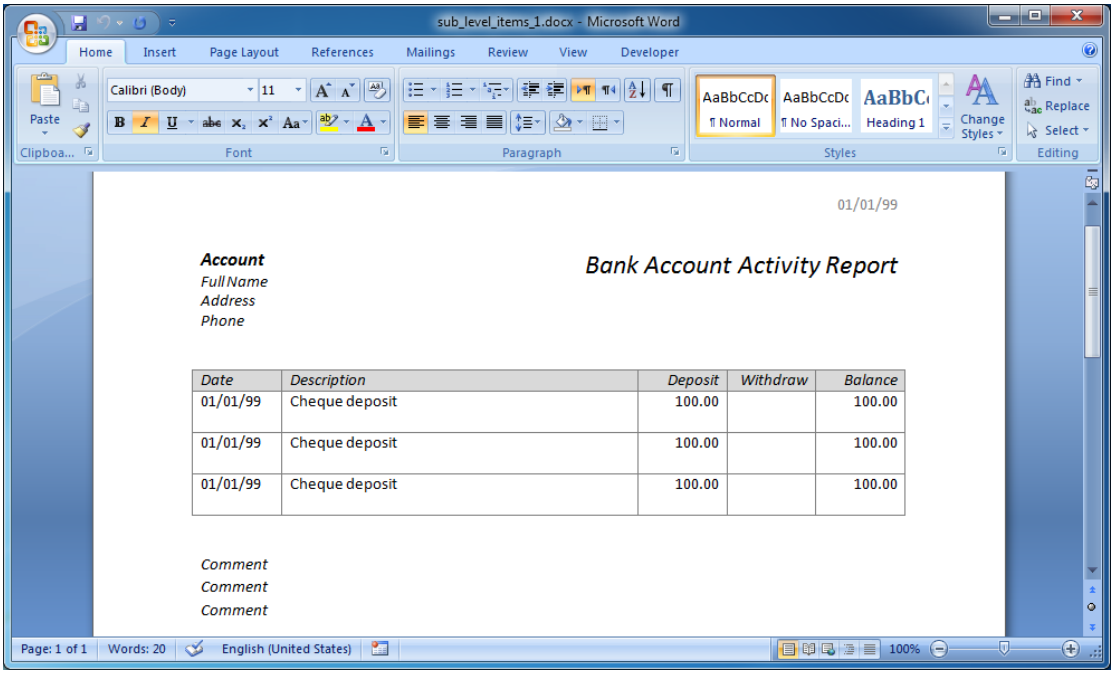
More specifically, when a sub level item is pasted DocxFactory looks for the parent of the item being pasted starting from the last item pasted and going up through its parents, when the parent is found, the item is added inside the parent.

With the following exceptions, if the item is not a direct child of the parent then all the item missing parents are first pasted before pasting the item. Similarly, if no parent was found except for the document then all the item parents upto the top level item are first pasted before pasting the item.

In our example, if you pasted the Line item without first pasting its Account parent item then DocxFactory would first paste its missing Account parent item and then paste the Line item.

5. Open the created .DOCX file.

   As you can see, every sub level item is repeated in its place (see picture below).



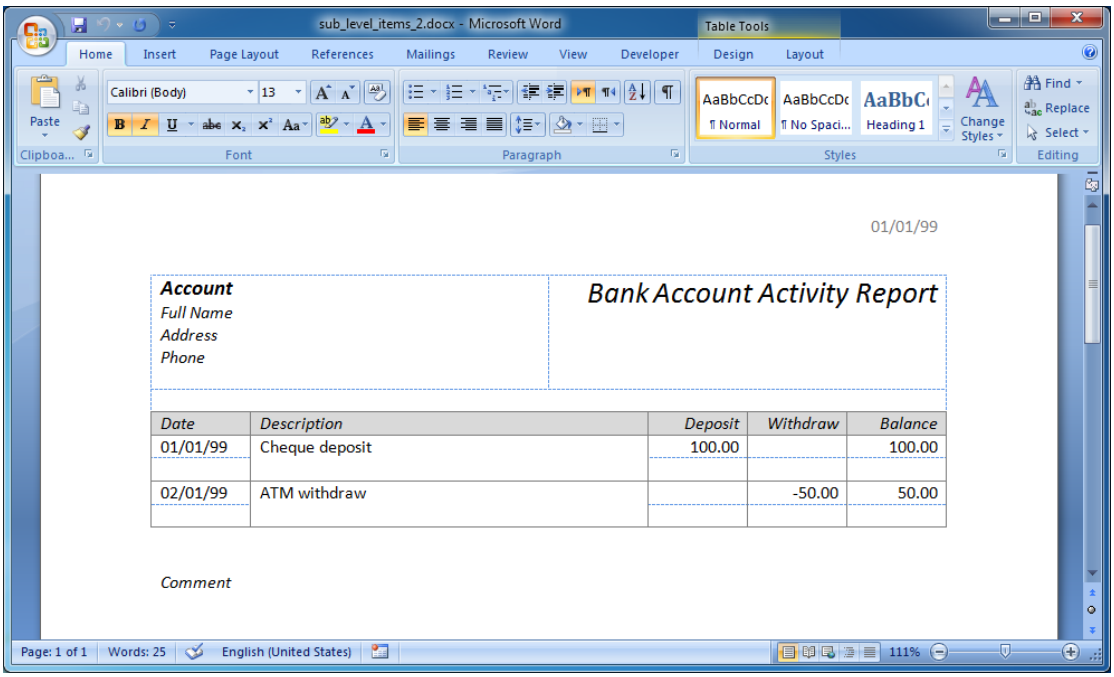
Sub Level Items 2 Exercise

Item Groups

In the previous exercise, every sub level item was repeated in its own place separately from each other but what if you want to mix them together?

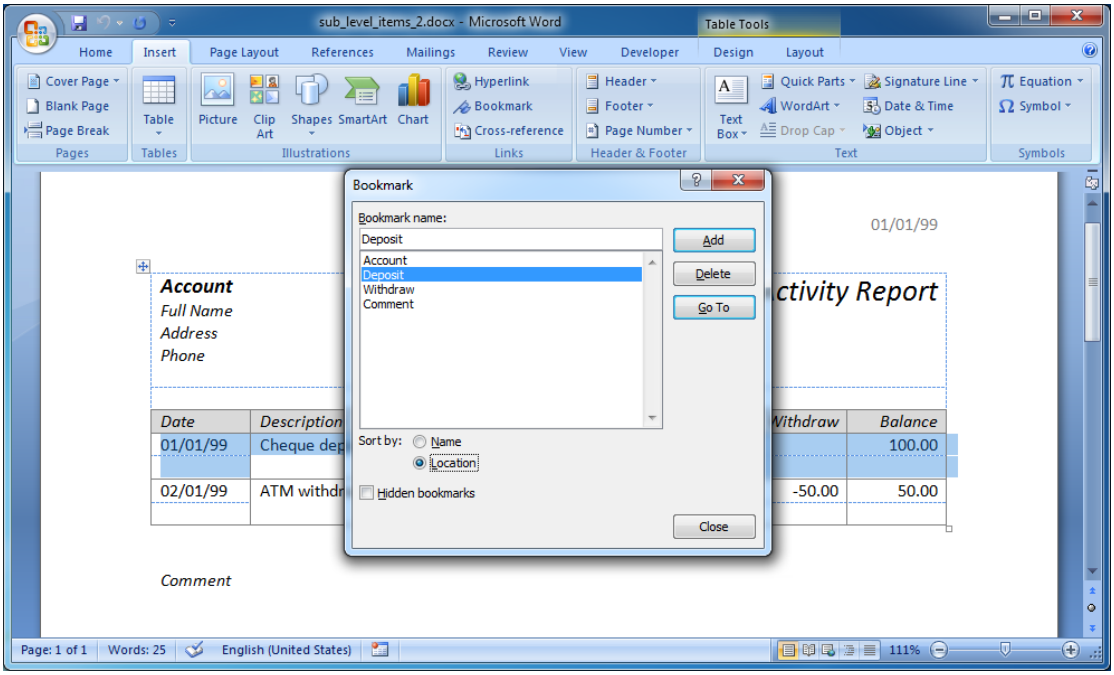
Items placed together one right after the other with nothing between them are called an Item Group. Items in the same group can be mixed together.

1. Open sub_level_items_2.docx in the DocxFactory/exercises/templates/ directory (see picture below).



2. Open the Bookmark dialog box to view the items in the template.

   Select sort by Location at the bottom of the dialog box and highlight the Deposit, Withdraw and Comment bookmarks. You can see that the Deposit, Withdraw highlighted areas are placed one right after the other with nothing between them. The Comment bookmark is placed separately with a space separating it from the other bookmarks (see picture below).



3. Compile the template.

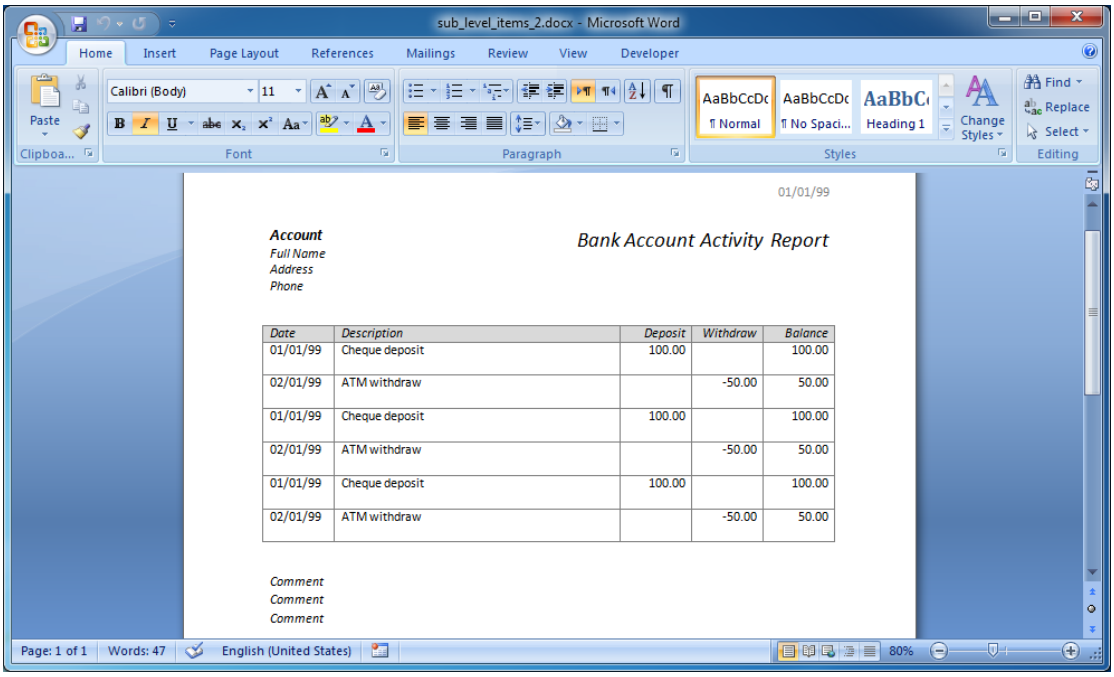
4. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/sub_level_items_2.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Deposit");
            l_merger.paste("Withdraw");
            l_merger.paste("Comment");
        }

        l_merger.save("/tmp/sub_level_items_2.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

5. Open the created .DOCX file.

   As you can see, the Deposit and Withdraw items are mixed together in the order they were pasted and the Comment items are still separate from the other items (see picture below).

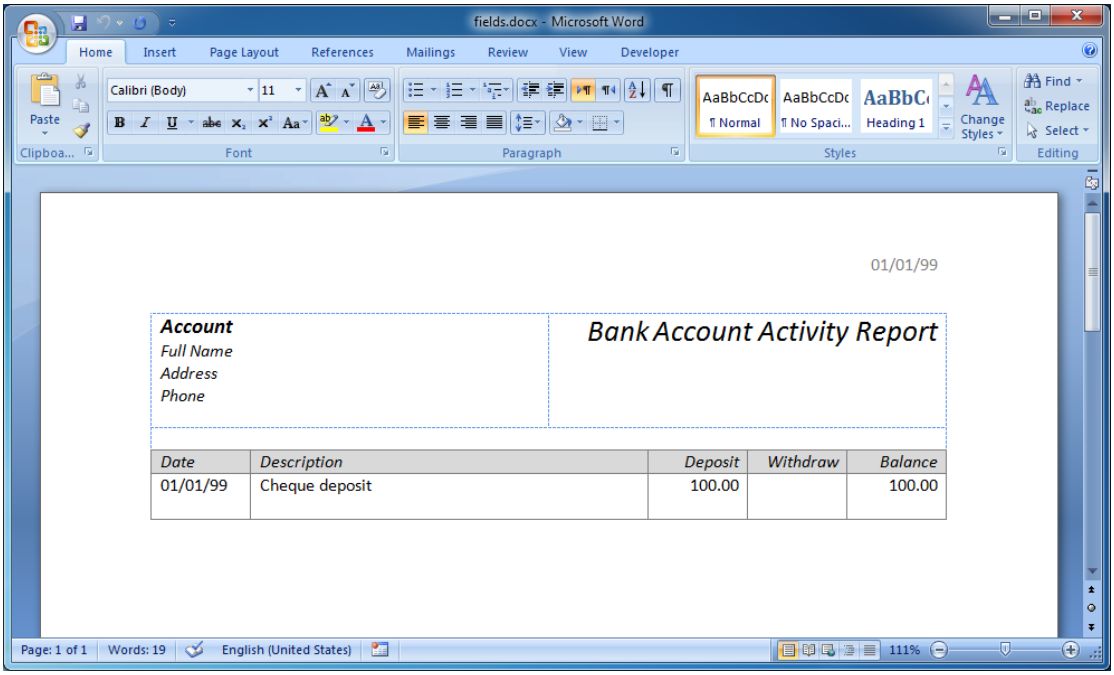


Fields

Fields are place holders in items for inserting values.

Fields Exercise

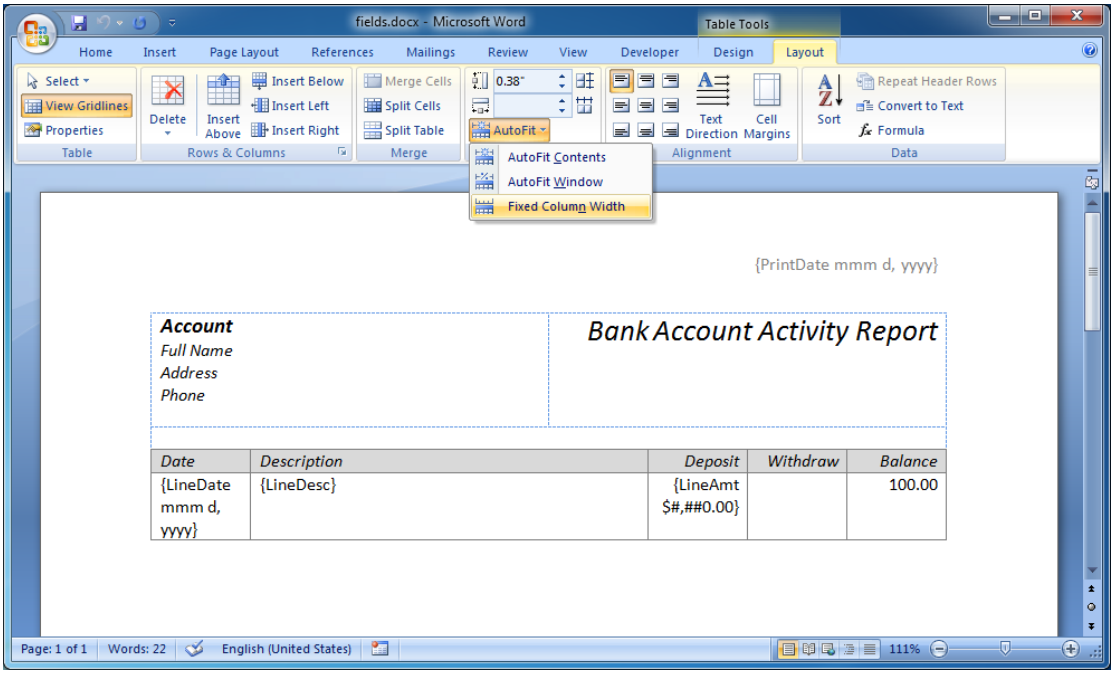
1. Open fields.docx in the DocxFactory/exercises/templates/ directory (see picture below).



2. To add fields, type the field name in squiggly brackets. You can also add an optional format after the field name separated by a space. For example: “{MyNumber #,##0.00}”.

   Add the fields below to your template (see picture below):

   1. {LineDate MMM d, yyyy} in the Date column.
   2. {LineDesc} in the Description column.
   3. {LineAmt $#,##0.00} in the Deposit column.
   4. {PrintDate MMM d, yyyy} in the header of the document.



Note: The same field name can be used more than once in the same document and even in the same item. When setting the field values, if there is more than one field with the specified field name then all the fields with that name are set.

Note: If your fields do not fit in the table columns and you do not want the columns width to change while you are designing your template then you can select Fixed Column Width in the AutoFit menu in the Layout ribbon (see picture above).

Note: If you want to use squiggly brackets in your template then escape the brackets by entering double brackets (“{{“ or “}}”). The brackets will show up only once in the created .DOCX file.

There are 3 basic field types. The field type is decided according to the field format entered in the template.

Text Format

Text fields have no format and are the default field type if no format is entered.

Note: At the moment, text fields have no format but text field format is planned to be added in future versions of DocxFactory.

Number Format

Number fields use the Microsoft Excel number formats.

For example:

- $#,##0.00
- #0%
- [GREEN]#,##0.0#;[RED]-#,##0.0#;-

In the template the number format should always be entered using the American number format (using a comma "," for the thousands separator and a dot "." for the decimal point). In the created .DOCX file the number values are displayed in the current DocxFactory number format. This approach allows for sharing templates created in different countries with different number formats.

Note: You can also enter colors using the six digits hexadecimal format prefixed with a #. For example: "[#FF00FF]".

Datetime Format

Like number formats, datetime formats also use the Microsoft Excel datetime formats.

For example:

- MMM d, yyyy
- [h]:mm:ss.sss AM/PM
- MM/dd/yyyy hh:mm:ss

In the template you can also enter a "99/99/99" or "99/99/9999" in the date format. In the created .DOCX file the day, month and year places in the date value are displayed according to the current DocxFactory date format.

Note: You can also escape characters that are used by any of the formats using the "\" sign. For example: "\M\D\Y MMM d, yyyy".

*Note: You can also use comments in your field definition using the C/C++ multiline comment syntax. For example: "{MyField /*MMM d, yyyy*/ 99/99/99}".*

3. Compile the template.

4. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;

int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        // DocxFactory default input codepage is UTF8.
        // Use setCodePage() to change the codepage.
        // l_merger.setCodePage("ISO8859-1");

        time_t l_start = clock();

        l_merger.load("/opt/DocxFactory/exercises/templates/fields.dfw");

        l_merger.setClipboardValue("_header", "PrintDate",(double) l_start);

        l_merger.paste("Account");

        for (int i = 0; i < 3; i++)
        {
            l_merger.setClipboardValue("Line", "LineDate",(double) l_start);
            l_merger.setClipboardValue("Line", "LineAmt", (double) i);
            l_merger.setClipboardValue("Line", "LineDesc",
                string("Desc") +(char) ('0' + i));

            l_merger.paste("Line");
        }

        l_merger.save("/tmp/fields.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code sets the field values of an item in the clipboard and pastes the item, copying the item with the set field values to the new document. Because of that, field values must be set before the item is pasted otherwise if the item is pasted before the values are set then the item will be pasted with empty values to the new document. Every time after a paste function is run all the fields in the clipboard are automatically cleared so you would not need to clear old values manually.

The code sets the values for all the basic field types:

- Text fields are set with a string value.
- Number fields are set with a double value.
- Datetime fields are set with a double value of UNIX time (seconds from 1/1/1970 at 00:00:00).

In addition, the PrintDate field in the header of the document is set. The "_Header" item name is used to refer to fields that are both in the header or footer of the document. The same item name is used to refer to both the header and footer to help separate between the design and code. For example: If you move a field in the header to the footer in your template then no changes in your code will be required. Unlike regular items, the header and footer do not need to be pasted and are always part of the document.

The code introduces the setClipboardValue function in the WordProcessingMerger singleton (see details below).

DocxFactory::WordProcessingMerger::setClipboardValue Function

Sets the field value of an item in the DocxFactory clipboard.

1. Sets the field with a string value.
2. Sets the field with a number value.

Declaration**:**

```
1. void setClipboardValue(
        const string& p_itemName,
        const string& p_fieldName,
        const string& p_str);

2. void setClipboardValue(
        const string& p_itemName,
        const string& p_fieldName,
        double        p_val);
```

Parameters:

```
p_itemName   - Item name.
p_fieldName  - Field name.
p_str        - String value.
p_val        - Number value.
```

Notes:

- The item names are not case sensitive. For example: “Account” and “account” refer to the same item.
- The field names are not case sensitive. For example: “LineDate” and “linedate” refer to the same field.
- If there is more than one field with the specified field name then all the fields with that name are set.
- If the item name is not specified (“”) then the fields anywhere in the document with the specified field name are set. It is recommended to specify an item name to avoid setting unintended fields.
- Use the “_Header” item name to refer to items that are in the header or footer of the document.
- If a text field is set with a number value then the number value is converted to a string value using the current DocxFactory number format.
- If a number field is set with a string value then the string value is converted to a number value using the current DocxFactory number format.
- Datetime fields are set with a double value of UNIX time (seconds from 1/1/1970 at 00:00:00). Fractions of a second values are also supported.

  Datetime fields can also be set with a string value of ISO datetime or just a date string in the current DocxFactory date format or just a time string.

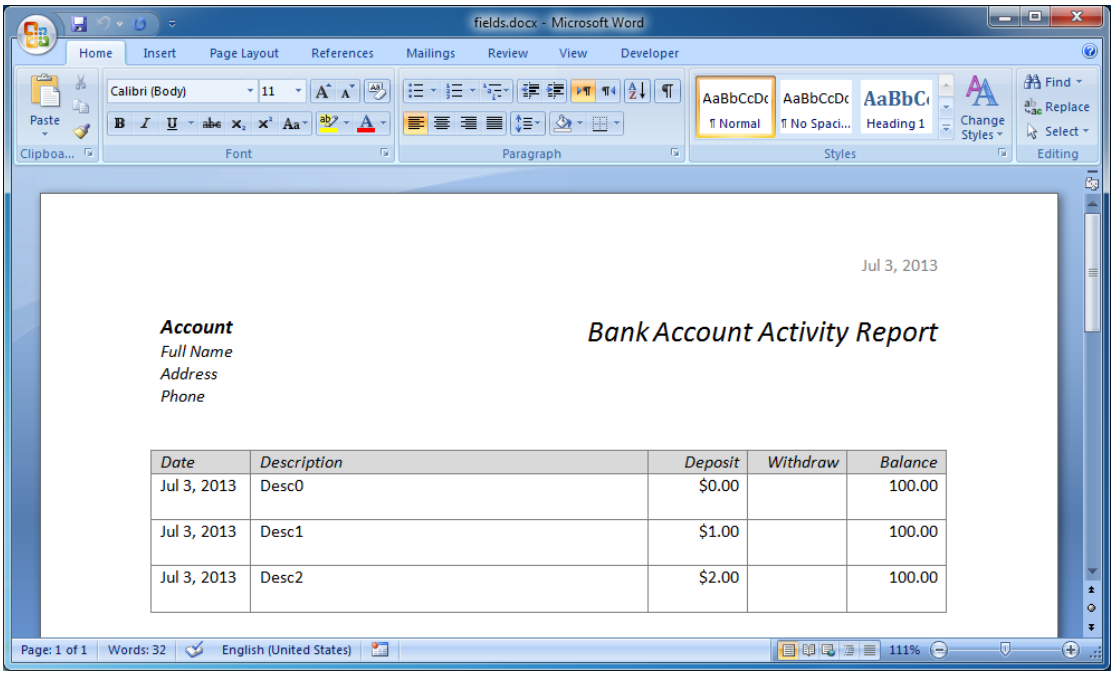
  For example:

  - 1999-12-31T23:59:59.123+11:30 (“T” can be replaced with a blank “ “)
  - 1999-12-31T23:59:59
  - 1999-12-31
  - 31/12/1999
  - 31/12/99
  - 23:59:59.999
  - 23:59:59

  The only way of setting the timezone for a datetime field is with an ISO datetime string value.

5. Open the created .DOCX file (see picture below).

   The code in this exercise contains all the main components for creating a .DOCX file: create a new document from a template, set field values, paste items and save the .DOCX file. As you can see, there are only 4 main functions needed to create a .DOCX file.



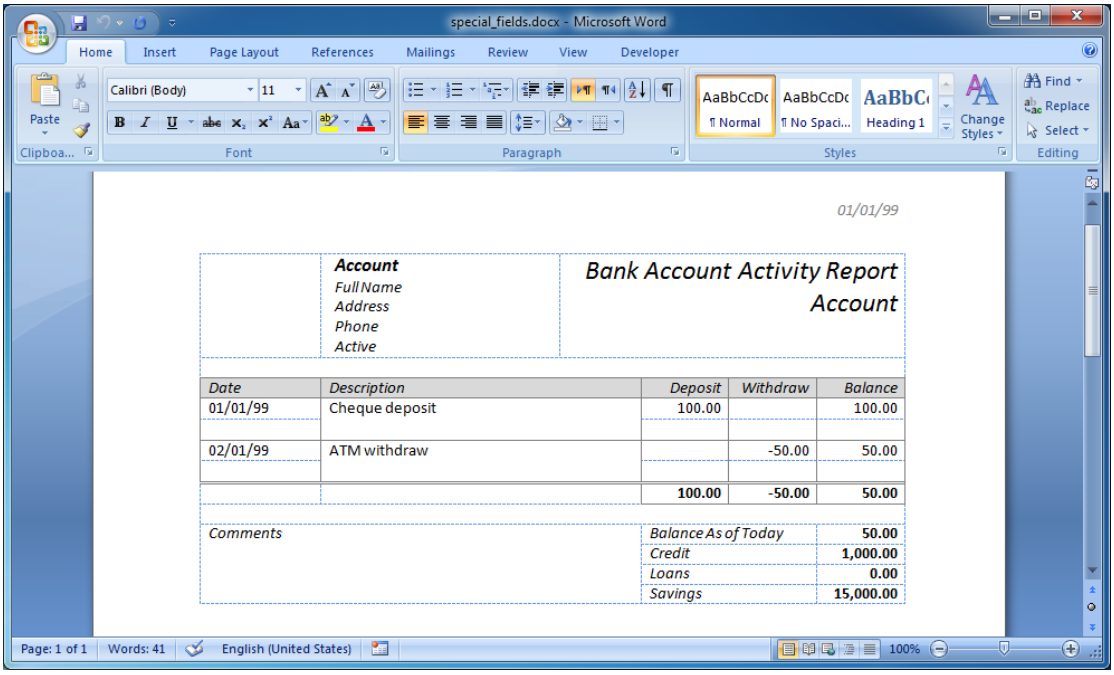
Special Fields

Besides the basic fields there are additional special fields:

1. Picture fields to insert pictures.
2. Barcode fields to insert text values displayed as barcodes. DocxFactory supports an extensive list of 1D and 2D barcodes including Code39, Code128, EAN, UPC, ISBN, Databar, Postal Codes, PDF417, Data Matrix, QR Code, Maxi Code and many more.
3. HTML/RTF fields to insert HTML or RTF values instead of plain text that can be used to add a little formatting like fonts, bolding, colors etc. all the way up to inserting mini documents with numbered or bullet lists, tables etc.
4. Boolean fields which are a cross between fields and items. Boolean fields have items for yes and no values (which just like regular items can have text, pictures etc.). When setting the field value, the appropriate item for that value is pasted.

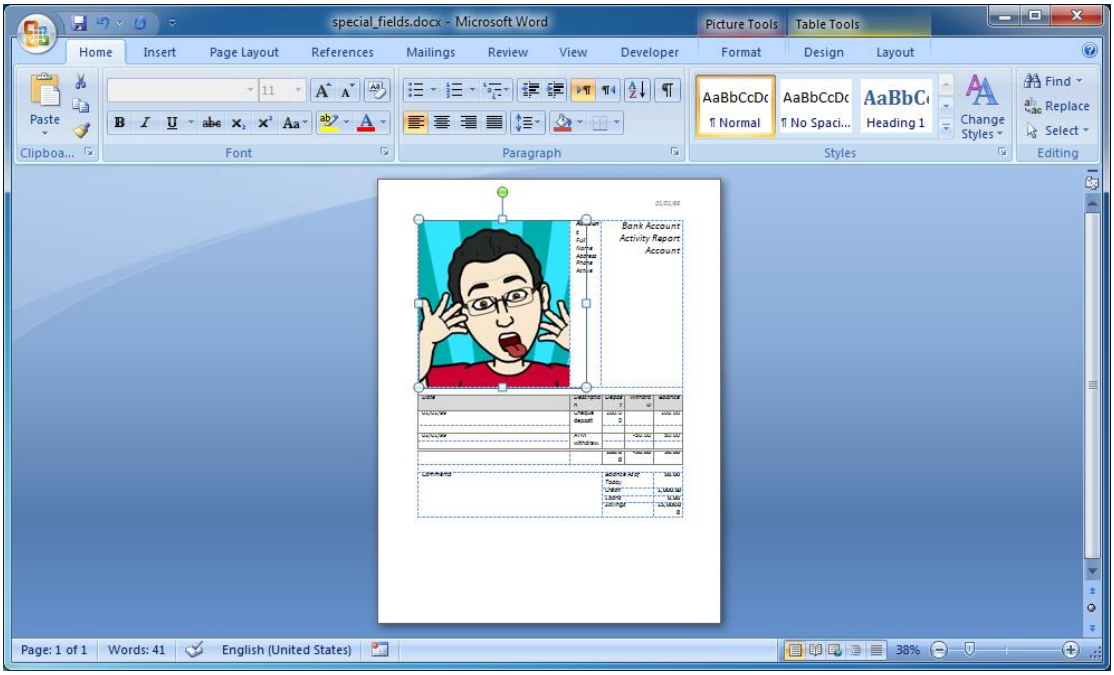
Special Fields Exercise

1. Open special_fields.docx in the DocxFactory/exercises/templates/ directory (see picture below).



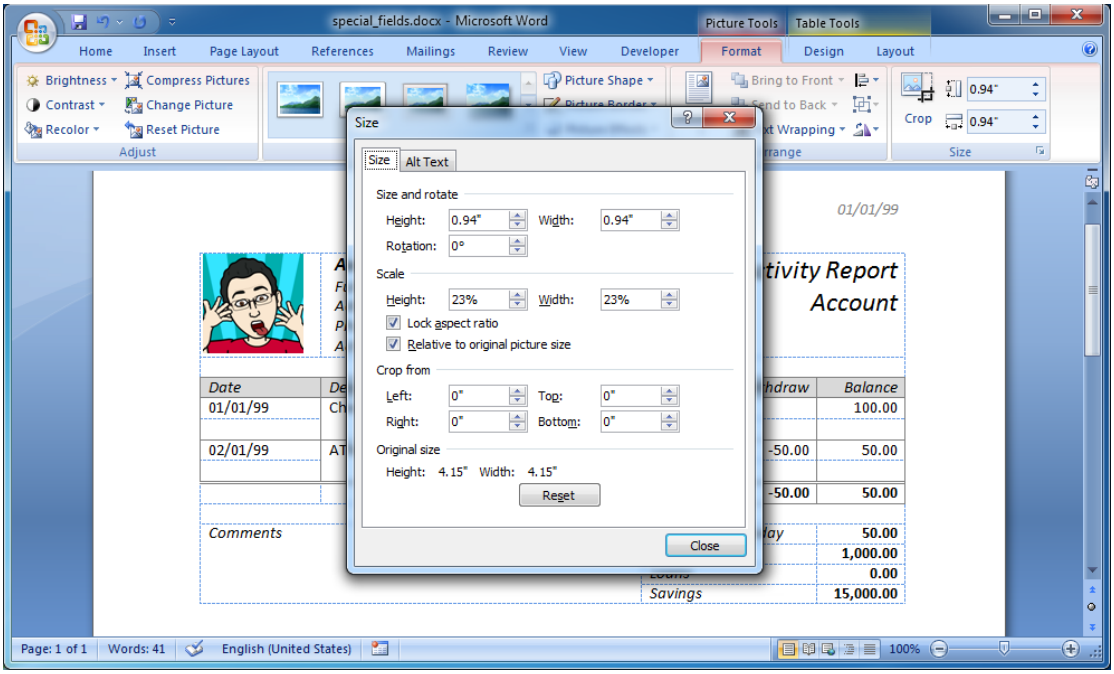
2. To add a picture field, first insert a picture to the template. The picture acts as a place holder that will be replaced with your pictures when creating a new document.

   First click in the box in the top left corner of the document to place the cursor in the box. Then select Picture in the Insert ribbon and choose customer0.png from the DocxFactory/images/ directory (see picture below).



3. Size the picture.

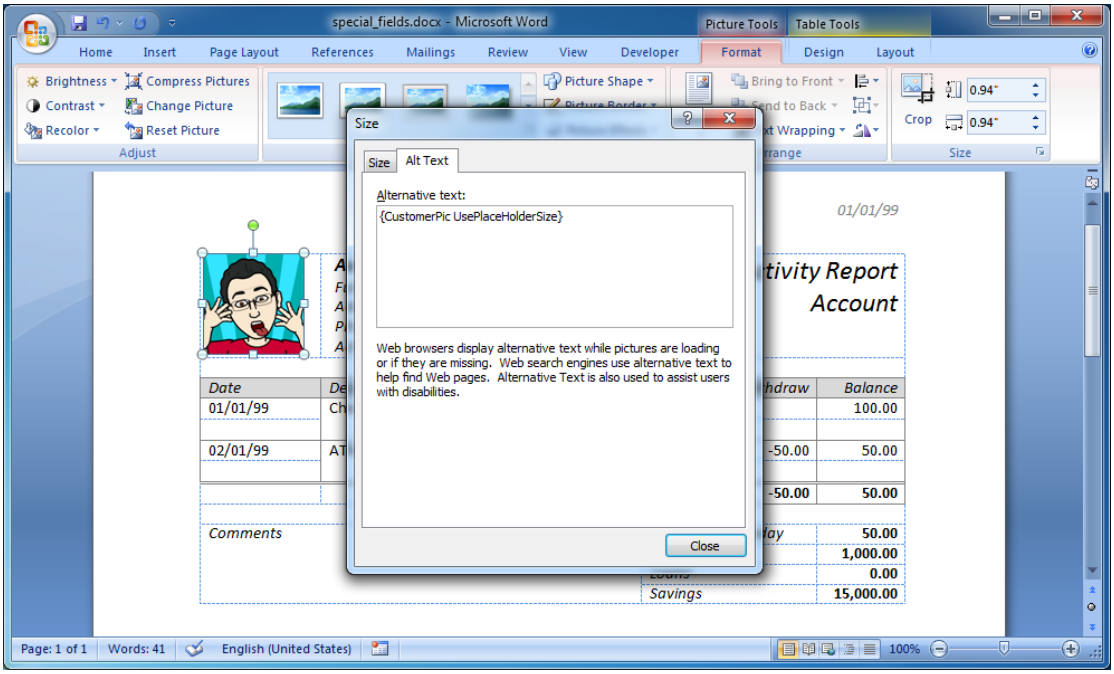
   Right click the picture and select Size from the drop down menu. In the size dialog box make sure the Lock Aspect Ratio is selected, enter 0.94" in the height and press the Close button (see picture below).



4. Enter the picture field name and optional formatting.

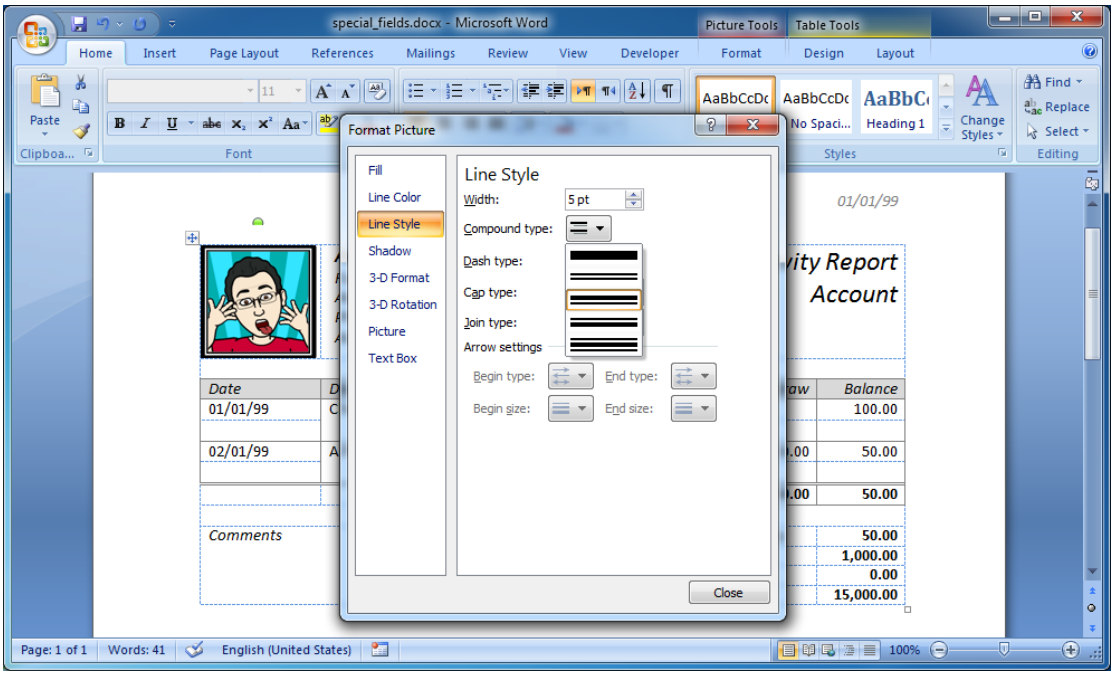
   If you are using Microsoft Word 2007 then right click the picture and select Size from the drop down menu. Select the Alt Text tab in the Size dialog box and enter "{CustomerPic UsePlaceHolderSize}" in the Alternative text box (see picture below).

   If you are using Microsoft Word 2010 then right click the picture and select Format Picture from the drop down menu. Select Alt Text from the menu and enter the field name and optional formatting in the Description box.



5. Add a border around the place holder image.

   Right click the picture and select Format Picture. In the Format Picture dialog box, select the Line Style category, select the third option from the Compound type combo box, enter 5pt Width and close the dialog box (see picture below).

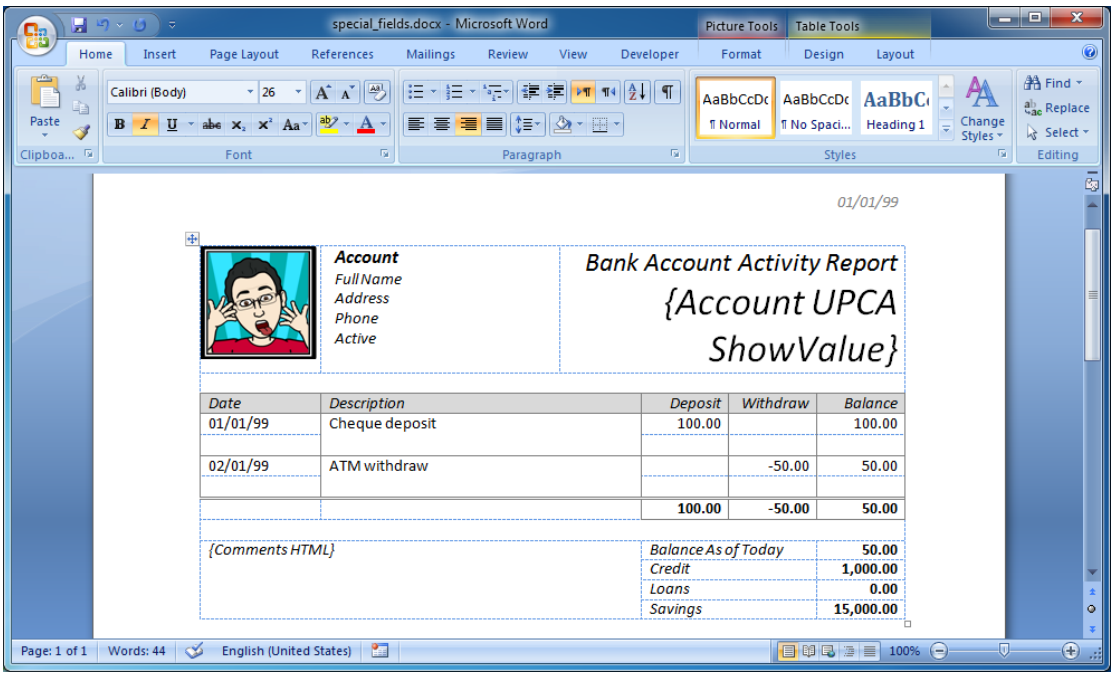


6. To add a barcode field, enter the field name, the barcode type and any other barcode optional formatting in squiggly brackets just like adding a basic field.

   Replace the Account under Bank Account Activity Report with “{Account UPCA ShowValue}” and increase the font size to 26 (see picture below).

7. To add an HTML/RTF field, enter the field name and “HTML” or “RTF” in the optional formatting in squiggly brackets just like adding a basic field.

   Replace the Comments at the bottom of the document with “{Comments HTML}” (see picture below).



The HTML tag styles like fonts, colors etc. are taken from document named styles for similar parts (in the Styles group in the Home ribbon). For example: “<h1>” tag uses the Heading 1 style, “ ” tag uses the Normal style and “” tag uses the Strong style. Just like in HTML, tag styles can also be combined. For example: “ ” combines the Normal and Strong styles.

You can also create your own named style and apply it to an HTML tag by setting the class property to the style name. For example: create a style named MyStyle and insert the HTML “ ” (note that the HTML class must be in lowercase). Although HTML can also be styled with CSS styles combined in the HTML it is recommended to insert plain HTML in the program and manage the HTML styles using named styles in the template to help keep the data and design separate.

The inserted HTML/RTF values are like paragraphs which have a line break before and after them and cannot be inserted in the middle of a line.

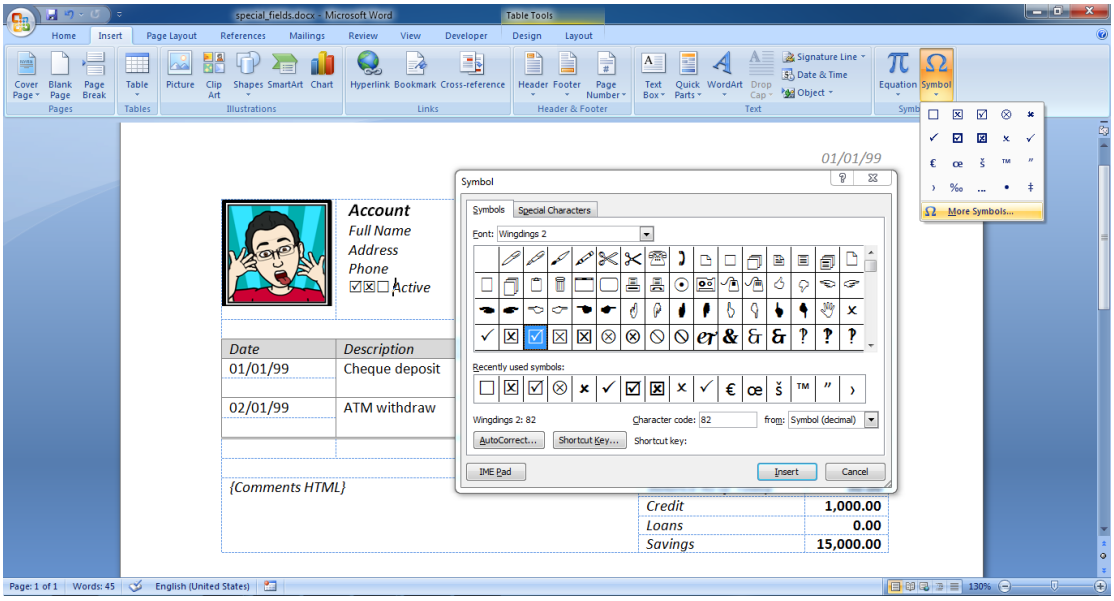
8. To add a boolean field, create items for the boolean field values:

    1. <field name>_yes - The item name for the field yes value.

    2. <field name>_no - The item name for the field no value.

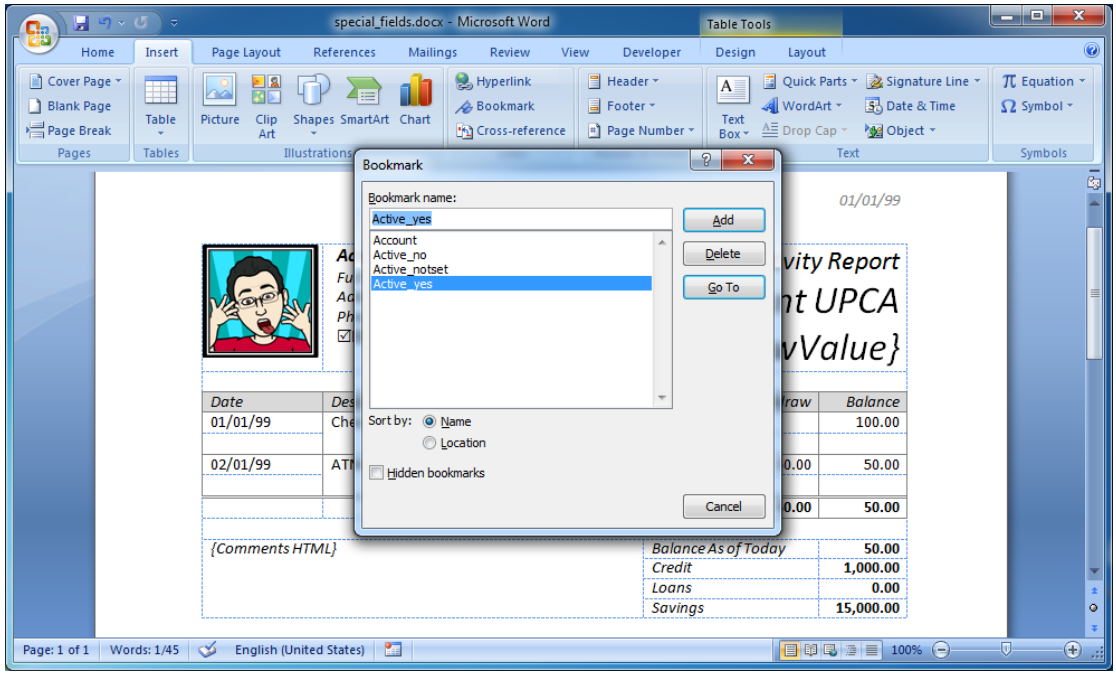
    3. <field name>_notset - The item name for the field if the value was not set.

    First, select More Symbols from the Symbol drop down menu in the Insert ribbon. Make sure the "Windings 2" font is selected and insert the symbols: "☑", "☒" and "☐" before the "Active" text and deselect the *Italicize* (see picture below).



Then create the items below (see picture below):

1. Active_yes - That marks the "☑" symbol.
2. Active_no - That marks the "☒" symbol.
3. Active_notset - That marks the "☐" symbol.



Note: All the items for the boolean field values are optional. If there is no item for the set field value then no item is pasted.

Note: The field name and value in the boolean field item names are not case sensitive. For example: "MyField_Yes" and "myfield_yes" refer to the same field name and value.

Just like basic fields, special fields also have optional formatting.

Picture Format

- UseImageFileSize - Resizes the inserted picture to the image file width and height and is the default format if no format is entered.
- UsePlaceHolderSize - Keeps the place holder picture width and height.
- UsePlaceHolderWidth - Keeps the place holder picture width and calculates the height according to the image file width/height ratio.
- UsePlaceHolderHeight - Keeps the place holder picture height and calculates the width according to the image file width/height ratio.

Barcode Format

- <Barcode type>

1 Dimensional Barcodes

Code11	- Code 11
Code25	- Code 2 of 5 Standard
Code25IATA	- Code 2 of 5 IATA
Code25Inter	- Code 2 of 5 Interleaved
Code25Logic	- Code 2 of 5 Data Logic
Code25Ind	- Code 2 of 5 Industrial
ITF14	- ITF-14
Code39	- Code 3 of 9 (Code 39)
Code39Ext	- Extended Code 3 of 9 (Code 39+)
Code93	- Code 93
PZN	- PZN
LOGMARS	- LOGMARS
Code32	- Code 32
Code128	- Code 128 (automatic subset switching)
Code128B	- Code 128 (subset B)
EAN128	- GS1-128 (UCC.EAN-128)
EAN14	- EAN-14
NVE18	- NVE-18
UPCA	- UPC-A
UPCE	- UPC-E
EAN	- EAN (2, 5, 8, 13)
ISBN	- ISBN (EAN-13 with verification stage)
Codabar	- Codabar
Pharma	- Pharmacode
Pharma2Track	- Pharmacode 2 Track
Plessey	- Plessey Code
MSIPlessey	- MSI Plessey
Telepen	- Telepen Alpha
TelepenNum	- Telepen Numeric
RSS14	- GS1 Databar-14
RSS14Stack	- GS1 Databar-14 Stacked
RSS14StackOmni	- GS1 Databar-14 Stacked Omni Directional
RSSLtd	- GS1 Databar Limited
RSSExp	- GS1 Databar Extended
RSSExpStack	- GS1 Databar Expanded Stacked
Channel	- Channel Code
FIM	- FIM
Flat	- Flattermarken
DAFT	- DAFT Code

PostNet	-	PostNet
PLANET	-	PLANET
OneCode	-	USPS OneCode
RM4SCC	-	Royal Mail 4 State (RM4SCC)
DPLeit	-	Deutshe Post Leitcode
DPIdent	-	Deutshe Post Identcode
AUSPost	-	Australia Post Standard Customer
AUSReply	-	Australia Post Reply Paid
AUSRoute	-	Australia Post Routing
AUSRedirect	-	Australia Post Redirection
KIX	-	Dutch Post KIX Code
JapanPost	-	Japanese Postal Code
KoreaPost	-	Korea Post

2 Dimensional Barcodes

Code16K	-	Code 16K
Code49	-	Code 49
PDF417	-	PDF417
PDF417Trunc	-	PDF417 Truncated
MicroPDF	-	MicroPDF417
DataMatrix	-	Data Matrix
QRCode	-	QR Code
MicroQR	-	Micro QR Code
MaxiCode	-	Maxicode
Aztec	-	Aztec Code
AzRune	-	Aztec Runes
CodeOne	-	Code One
GridMatrix	-	Grid Matrix

HIBC Version Barcodes

HIBCCode39	-	HIBC Code 39
HIBCCode128	-	HIBC Code 128
HIBCPDF417	-	HIBC PDF417
HIBCMicroPDF	-	HIBC MicroPDF417
HIBCDataMatrix	-	HIBC Data Matrix
HIBCQRCode	-	HIBC QR Code
HIBCAztec	-	HIBC Aztec Code

Composite Barcodes

CCEAN	- Composite Symbol with EAN Linear Component
CCEAN128	- Composite Symbol with GS1-128 Linear Component
CCUPCA	- Composite Symbol with UPC-A Linear Component
CCUPCE	- Composite Symbol with UPC-E Linear Component
CCRSS14	- Composite Symbol with GS1 Databar-14 Linear Component
CCRSS14Stack	- Composite Symbol with GS1 Databar-14 Stacked
CCRSS14StackOmni	- Composite Symbol with GS1 Databar-14 Stacked Omni
CCRSSLtd	- Composite Symbol with GS1 Databar Limited Component
CCRSSExp	- Composite Symbol with GS1 Databar Extended Component
CCRSSExpStack	- Composite Symbol with GS1 Databar Expanded Stacked

- Scale: <scale> - Used to set the barcode print size. The barcode size is multiplied by the scale, where the height of the barcode in scale 1 is equal to the height of a paragraph in font size 30. The default scale, if no scale is entered is calculated according to the font size of the barcode field definition in the template, where font size 30 is equal to scale 1.
- Height: <height> - The height for 1 dimensional barcodes. The default height, if no height is entered is 50.
- ShowValue - Shows the barcode text value.
- Angle: <angle> - Turns the barcode in an angle.
- Font: <name> - The show value font name. The default font, if no font is entered is the font “Calibri”.
- FontSize: <size> - The show value font size. The default font size, if no font size is entered is 11.
- FGColor: <color> - The barcode foreground color (bars and text color). The default foreground color, if no foreground color is entered is the foreground color of the barcode field definition in the template.
- BGColor: <color> - The barcode background color. The default background color, if no background color is entered is the background color (or table cell shade) of the barcode field definition in the template.
- WhiteSpace: <width> - The whitespace width on the barcode sides.
- Border: <width> - The barcode border width.
- Bind - Draws a binding border with bars on the top and bottom of the barcode.
- Box - Draws a box border around the barcode.

Barcode Specific Formats

Code39 and Code39Ext

- CheckDigit - Adds a modulo 10 check digit.

UPCA, UPCE, EAN, ISBN and CCEAN

Note: An EAN2 or EAN5 barcode can be added to the right of the barcode by adding "+<value>" with the EAN value to the barcode value. For example: "1234567890+12345".

MSIPlessey

- CheckDigit: <option number> - Adds a check digit. Options:

  1. Modulo 10
  2. Modulo 10 & Modulo 10
  3. Modulo 11
  4. Modulo 11 & Modulo 11

RSSExp

Notes: GS1 Application Identifiers (AIs) should be entered using [square brackets] notation. These will be converted to (round brackets) when the value is shown.

RSSExpStack and CCRSSExpStack

- Cols: <cols> - Number of columns from 1 to 9.

Notes: GS1 Application Identifiers (AIs) should be entered using [square brackets] notation. These will be converted to (round brackets) when the value is shown.

Channel

- Channels: <channels> - Number of channels from 3 to 8.

OneCode

Notes: Input data consists of a 20 digit tracking code, followed by a dash ("-"), followed by a delivery point zip-code which can be 0, 5, 9 or 11 digits in length.

Code16K and Code49

- GS1 - GS1 input mode.

PDF417, PDF417Trunc and HIBCPDF417

- Cols: <cols> - Number of columns from 1 to 20.
- Security: <n> - Error correction value from 0 to 8. Number of code words equals $2^{(n+1)}$.

MicroPDF and HIBCMicroPDF

- Cols: <cols> - Number of columns from 1 to 4.

DataMatrix

- Size: <option number> - Resolution size. Options:

1. 10 x 10	16. 64 x 64
2. 12 x 12	17. 72 x 72
3. 14 x 14	18. 80 x 80
4. 16 x 16	19. 88 x 88
5. 18 x 18	20. 96 x 96
6. 20 x 20	21. 104 x 104
7. 22 x 22	22. 120 x 120
8. 24 x 24	23. 132 x 132
9. 26 x 26	24. 144 x 144
10. 32 x 32	25. 8 x 18
11. 36 x 36	26. 8 x 32
12. 40 x 40	27. 12 x 26
13. 44 x 44	28. 12 x 36
14. 48 x 48	29. 16 x 36
15. 52 x 52	30. 16 x 48

- Square - Adjusts the size using only square sizes.
- GS1 - GS1 input mode.

Note: If no size is entered or the data does not fit in the entered size then the size is adjusted automatically.

QRCode

- Size: <option number> - Resolution size. Options:

1.	21 x 21	21.	101 x 101
2.	25 x 25	22.	105 x 105
3.	29 x 29	23.	109 x 109
4.	33 x 33	24.	113 x 113
5.	37 x 37	25.	117 x 117
6.	41 x 41	26.	121 x 121
7.	45 x 45	27.	125 x 125
8.	49 x 49	28.	129 x 129
9.	53 x 53	29.	133 x 133
10.	57 x 57	30.	137 x 137
11.	61 x 61	31.	141 x 141
12.	65 x 65	32.	145 x 145
13.	69 x 69	33.	149 x 149
14.	73 x 73	34.	153 x 153
15.	77 x 77	35.	157 x 157
16.	81 x 81	36.	161 x 161
17.	85 x 85	37.	165 x 165
18.	89 x 89	38.	169 x 169
19.	93 x 93	39.	173 x 173
20.	97 x 97	40.	177 x 177

- Security: <option number> - Error correction. Options:

1. Approx. 20% of symbol.
2. Approx. 37% of symbol.
3. Approx. 55% of symbol.
4. Approx. 65% of symbol.

Note: If no size is entered or the data does not fit in the entered size then the size is adjusted automatically.

MicroQR

- Cols: <cols> - Number of columns from 1 to 4.

MaxiCode

- Mode: <mode> - Maxicode mode from 2 to 6. The default mode, if no mode is entered is mode 4.

Note: Modes 2 and 3 require a primary and secondary value. The value is divided into a primary and secondary value by a newline character ("\n"). The primary value required format is 9 digits for mode 2 and 6 alpha numeric characters for mode 3 of post code data (the remaining characters should be filled with spaces), followed by a 3 digit country code (ISO3166), followed by a 3 digit service code depending on your parcel courier.

Aztec

- Size: <option number> - Resolution size. Options:

1.	15 x 15	19.	79 x 79
2.	19 x 19	20.	83 x 83
3.	23 x 23	21.	87 x 87
4.	27 x 27	22.	91 x 91
5.	19 x 19	23.	95 x 95
6.	23 x 23	24.	101 x 101
7.	27 x 27	25.	105 x 105
8.	31 x 31	26.	109 x 109
9.	37 x 37	27.	113 x 113
10.	41 x 41	28.	117 x 117
11.	45 x 45	29.	121 x 121
12.	49 x 49	30.	125 x 125
13.	53 x 53	31.	131 x 131
14.	57 x 57	32.	135 x 135
15.	61 x 61	33.	139 x 139
16.	67 x 67	34.	143 x 143
17.	71 x 71	35.	147 x 147
18.	75 x 75	36.	151 x 151

- Security: <option number> - Error correction. Options:

1. >10% + 3 code words.
2. >23% + 3 code words.
3. >36% + 3 code words.
4. >50% + 3 code words.

- GS1 - GS1 input mode.

Note: If no size is entered or the data does not fit in the entered size then the size is adjusted automatically.

CodeOne

- Size: <option number> - Resolution size. Options:

  1. 16 x 18
  2. 22 x 22
  3. 28 x 32
  4. 40 x 42
  5. 52 x 54
  6. 70 x 76
  7. 104 x 98
  8. 148 x 134
  9. 8X height
  10. 16X height

- GS1 - GS1 input mode.

Note: If no size is entered or the data does not fit in the entered size then the size is adjusted automatically.

GridMatrix

- Size: <option number> - Resolution size. Options:

  1. 18 x 18
  2. 30 x 30
  3. 42 x 42
  4. 54 x 54
  5. 66 x 66
  6. 78 x 78
  7. 90 x 90
  8. 102 x 102
  9. 114 x 114
  10. 126 x 126
  11. 138 x 138
  12. 150 x 150
  13. 162 x 162

- Security: <option number> - Error correction. Options:

  1. Approx. 10% of symbol.
  2. Approx. 20% of symbol.
  3. Approx. 30% of symbol.
  4. Approx. 40% of symbol.
  5. Approx. 50% of symbol.

Note: If no size is entered or the data does not fit in the entered size then the size is adjusted automatically.

Composite Barcodes

- Mode: <option number> - The 2D component of the composite symbol. Options:

  1. CC-A
  2. CC-B
  3. CC-C

Note: Composite barcodes require a primary and secondary value for the linear and 2D component respectively. The value is divided into a primary and secondary value by a newline character ("\n").

HTML/RTF Format

- HTML - For inserting HTML values.
- RTF - For inserting RTF values.

Boolean Format

Booleans fields have no additional optional formatting.

Note: All optional formatting are not case sensitive. For example: "HTML" and "html" refer to the same optional formatting.

9. Compile the template.

10. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/special_fields.dfw");

        l_merger.setClipboardValue("Account", "Account", "12345678901");
        l_merger.setClipboardValue("Account", "Active",  "no");
        l_merger.setClipboardValue("Account", "CustomerPic",
            "/opt/DocxFactory/exercises/images/customer1.png");

        l_merger.setClipboardValue("Account", "Comments",
            "<h3>An unordered list:</h3>"
            "<ul>"
            "<li>List item</li>"
            "<li>List item</li>"
            "<li>List item</li>"
            "</ul>");

        l_merger.paste("Account");

        l_merger.save("/tmp/special_fields.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code sets the value for all the special field types:

1. Picture fields are set with a string value of the image file path.

   Note: Relative paths are resolved relative to the current working directory. For example: "dir/image.jpg" is resolved to "[working directory]/dir/image.png".

   Note: The picture inserted as a place holder is only used to define the picture field in the template. In fact, the picture is removed when compiling the template, leaving the place holder blank. If the picture field is not set when creating a new document then no picture is inserted in the place holder and the picture field remains blank.

   Note: DocxFactory does not support images in compatibility mode on purpose because of their many drawbacks, most notably .DOCX files with even a hundred images may take many minutes and even more to open. To find if the .DOCX file is in compatibility mode, check if the title bar displays [Compatibility Mode]. To remove compatibility mode, select Save as from the menu, Word Document and in the Save as dialog-box make sure the Maintain compatibility with Word 97-2003 checkbox is not selected.

   Note: In most cases, the image files are saved inside the .DOCX file (also called embedded images) but it is also possible to save only the image files URL instead (also called linked images) to reduce the .DOCX file size but if the .DOCX file is opened on a network with no access to the image files then the images will not be displayed. DocxFactory uses embedded images by default. To use linked images pass an image file "file://<host>/<directory>/<file>" URL instead. It is recommended to use embedded images. Use linked images only when necessary.

2. Barcode fields are set with a string value to display as a barcode.

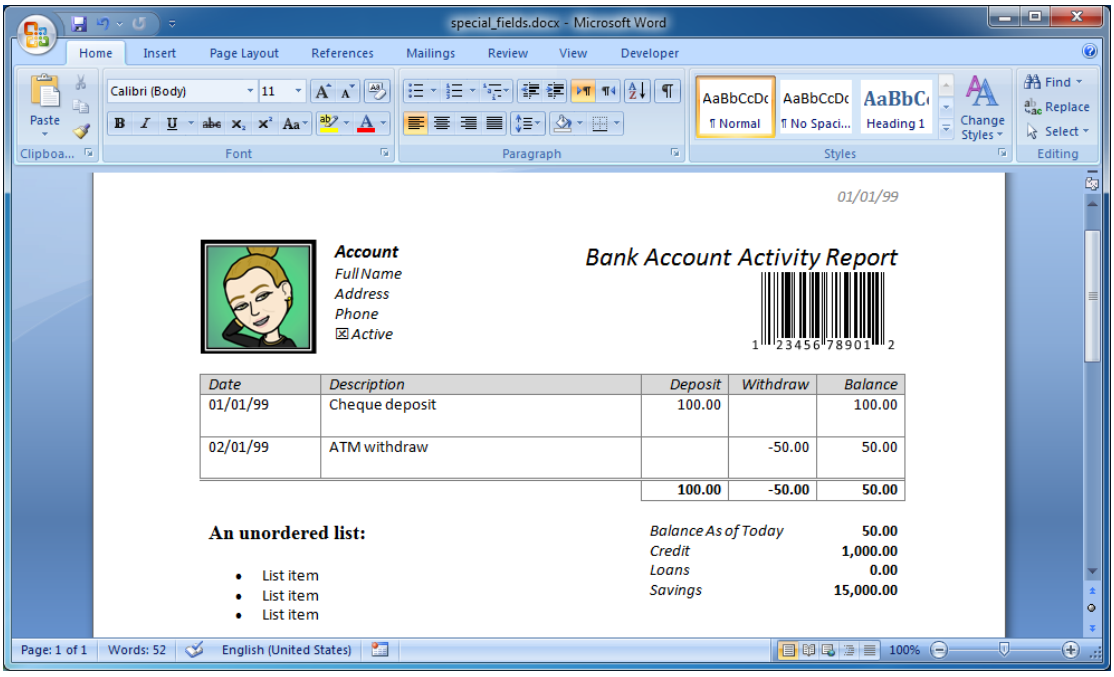
3. HTML/RTF field are set with a string value of the HTML or RTF.

   Note: The HTML "<html><body></body></html>" tags (including DTD and other headers) or the RTF "{\rtf1\ansi\deff0}" definitions are optional. If they are missing then DocxFactory will add them automatically.

   Note: Similar to picture fields, DocxFactory automatically embeds <img src="<file>"> tag image files into the document so the document can be sent out and the images can still be displayed. To force DocxFactory to use linked images, enter an "http://<host>/<directory>/<file>" or "file://<host>/<directory>/<file>" URL.

4. Boolean fields are set with a number value of 0 or string value of "0", "0.0", "no", "false" for no and all other values for yes.

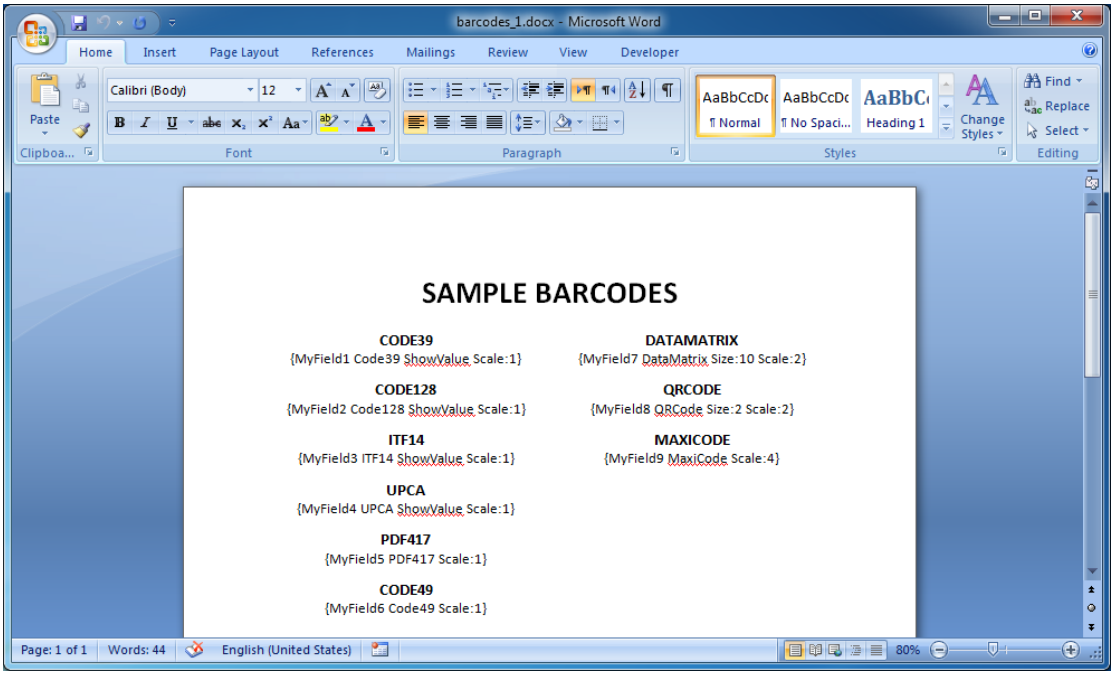
10. Open the created .DOCX file (see picture below).



Barcodes 1 Exercise

The following exercise creates a document with sample barcodes.

1. Open barcodes_1.docx in the DocxFactory/exercises/templates/ directory (see picture below).



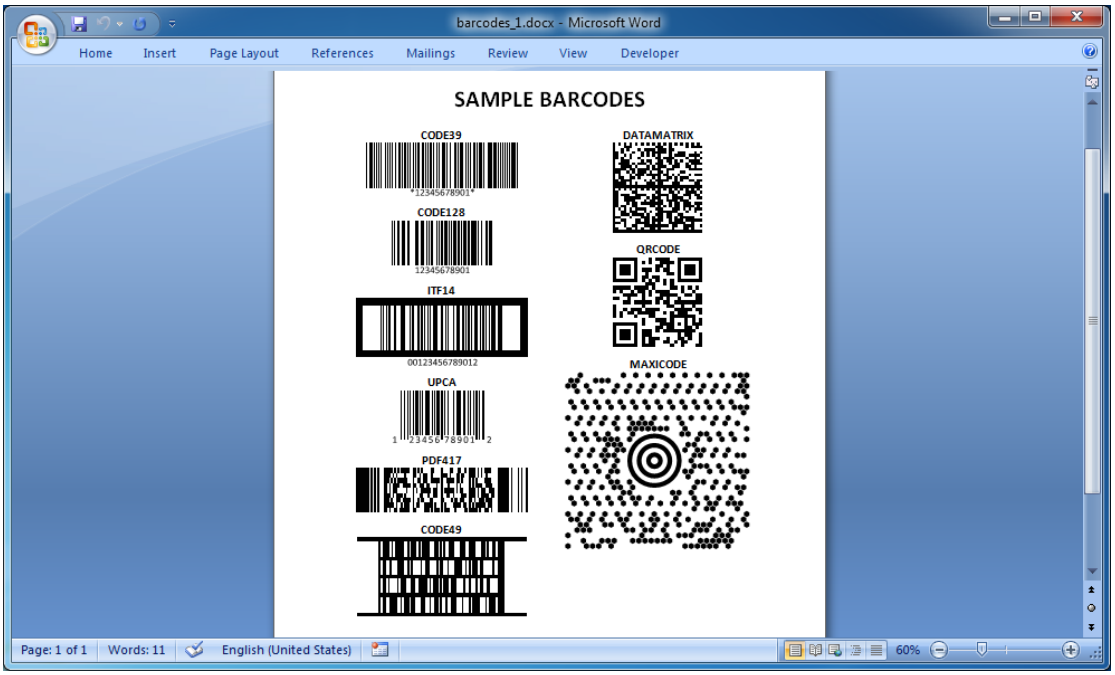
2. Compile the template.

3. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/barcodes_1.dfw");

        l_merger.setClipboardValue("Main", "MyField1", "12345678901");
        l_merger.setClipboardValue("Main", "MyField2", "12345678901");
        l_merger.setClipboardValue("Main", "MyField3", "12345678901");
        l_merger.setClipboardValue("Main", "MyField4", "12345678901");
        l_merger.setClipboardValue("Main", "MyField5", "Hello World");
        l_merger.setClipboardValue("Main", "MyField6", "Hello World");
        l_merger.setClipboardValue("Main", "MyField7", "Hello World");
        l_merger.setClipboardValue("Main", "MyField8", "Hello World");
        l_merger.setClipboardValue("Main", "MyField9", "Hello World");

        l_merger.paste("Main");

        l_merger.save("/tmp/barcodes_1.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

4. Open the created .DOCX file (see picture below).

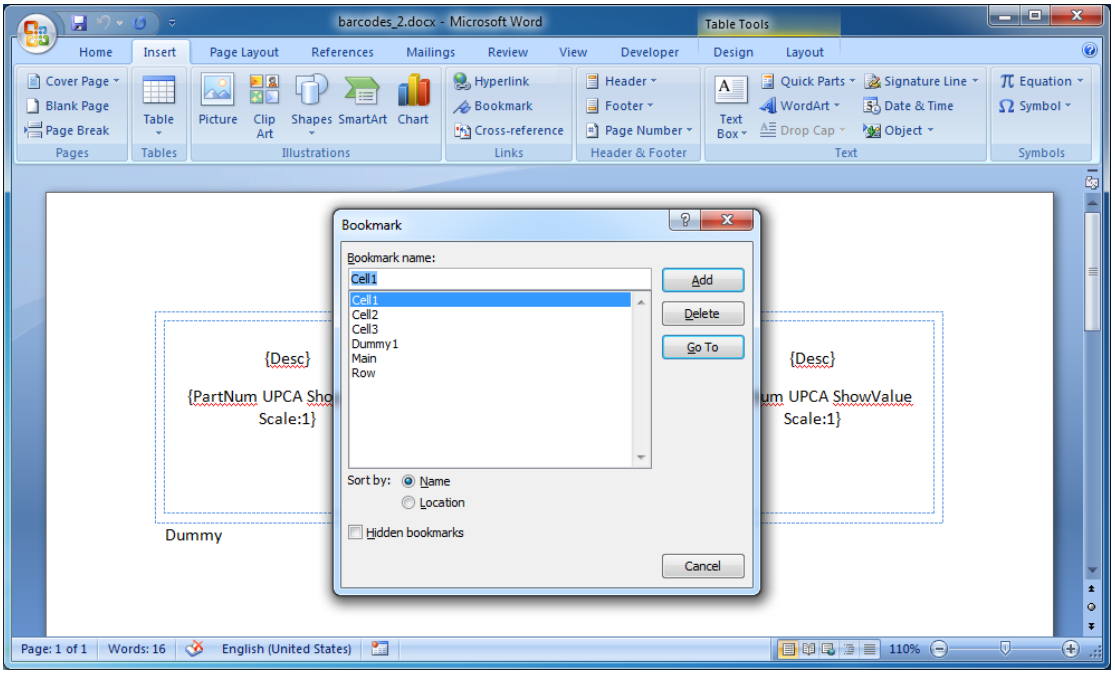


Barcodes 2 Exercise

The following exercise creates a document for printing stickers.

1. Open barcodes_2.docx in the DocxFactory/exercises/templates/ directory.

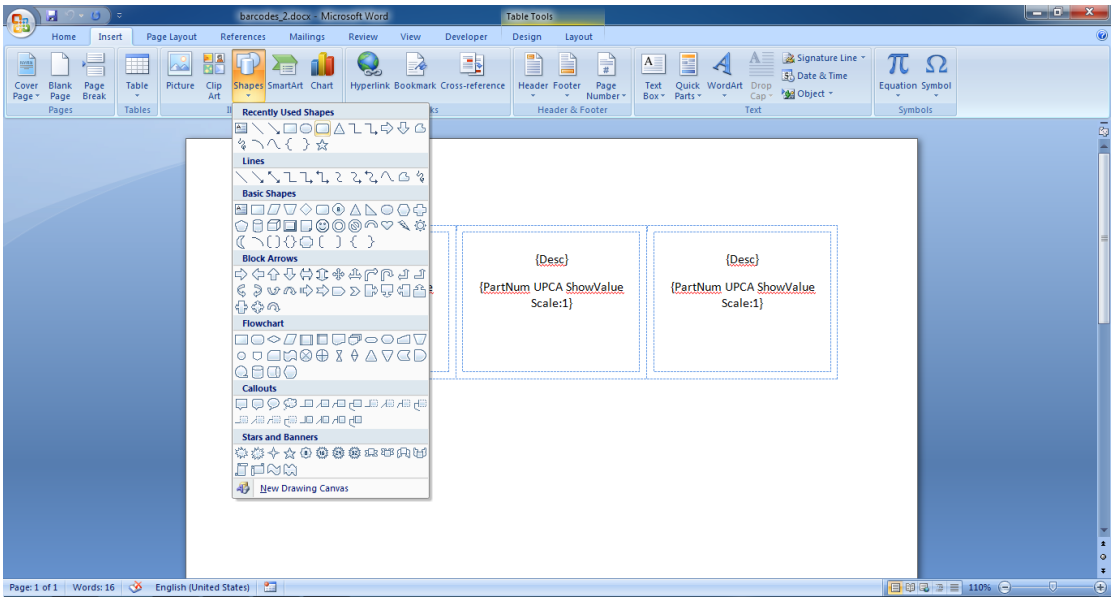
   Open the Bookmark dialog box and go through and highlight the bookmarks (see picture below).



The template has a Row and Cell items like a table and you can paste the Row and Cell items like you would for creating a table. There is also a Main top level item because if the Row item was a top level item then it would start in a new page every time it was pasted and a dummy item which is never pasted and its only purpose is to help differentiate between the Row and Main items.

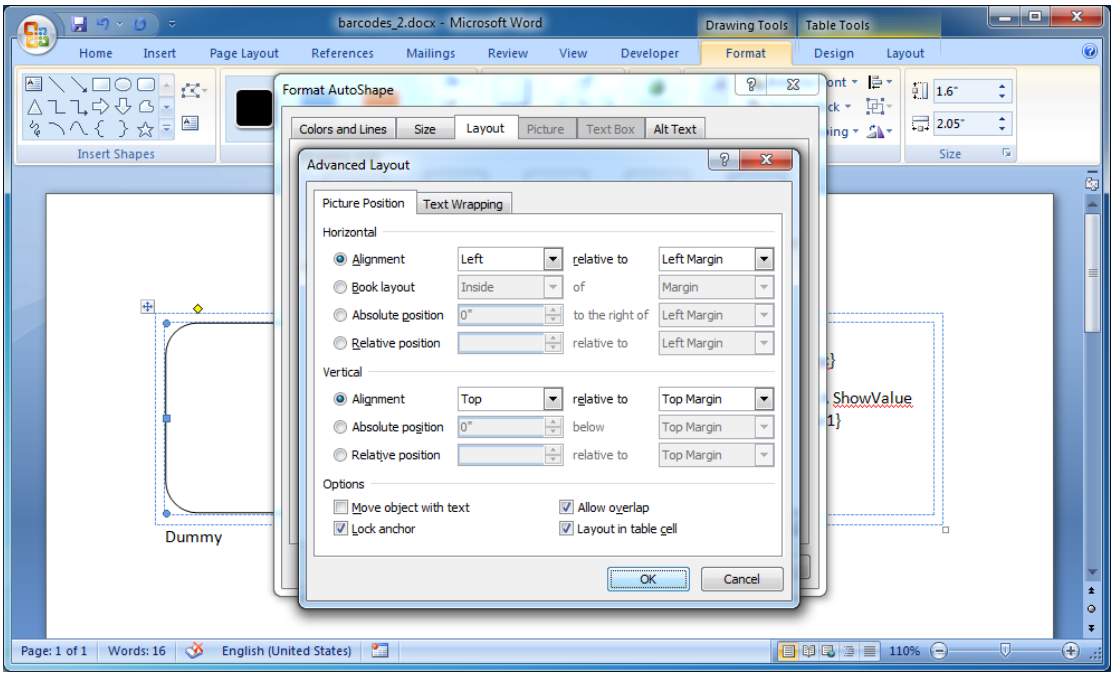
2. Draw a rounded rectangle over the first Cell item.

   Place the cursor before “{Desc}” field. Select rounded rectangle from the Shapes drop down menu in the Insert ribbon and draw a rounded rectangle over the first cell item (see picture below).



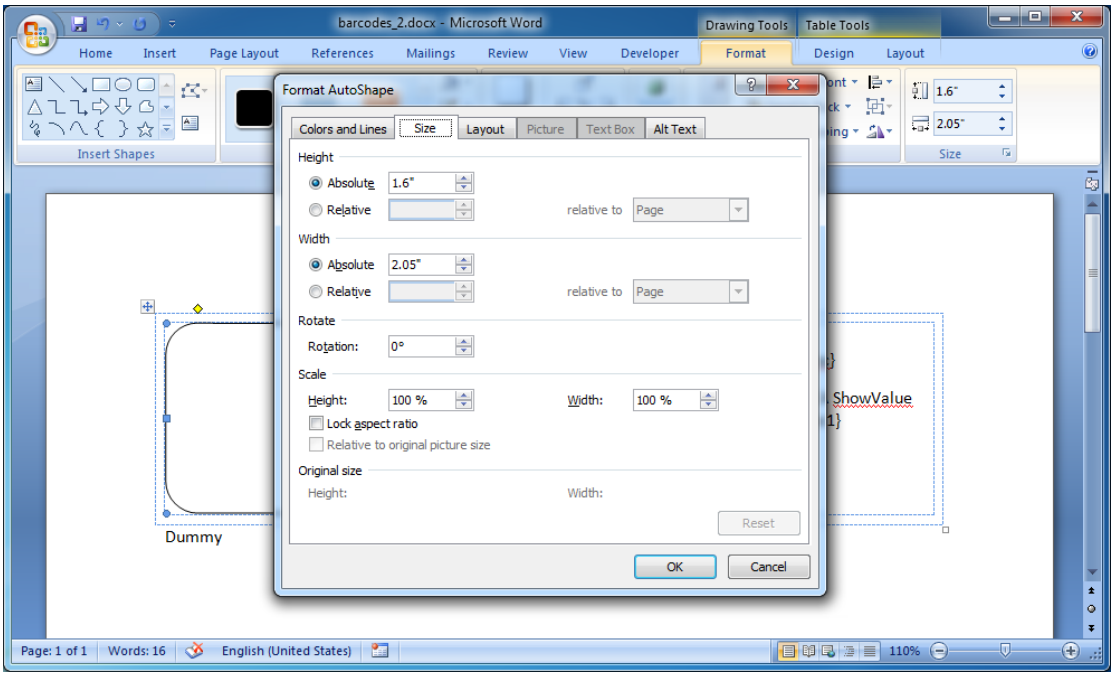
3. Position the rounded rectangle.

   Right click the rounded rectangle and select Format AutoShape from the drop down menu. Click the Advanced button in the Layout tab. Set Horizontal Alignment: Left relative to: Left Margin and Vertical Alignment: Top relative to: Top Margin in the Picture Position tab and Select Layout in table cell (see picture below).



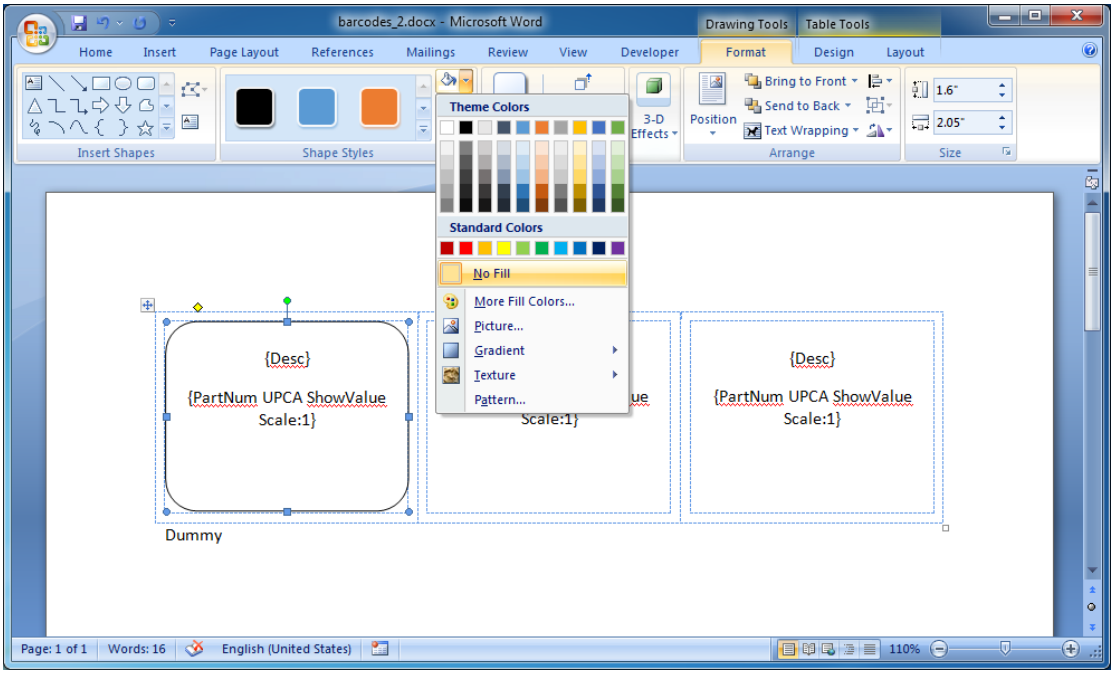
4. Set the rounded rectangle size.

   Select the Size tab in the Format AutoShape dialog box. Set the Height to Absolute: 1.6" and Width to Absolute: 2.05" (see picture below).



5. Remove the rounded rectangle fill color and keep only the shape outline.

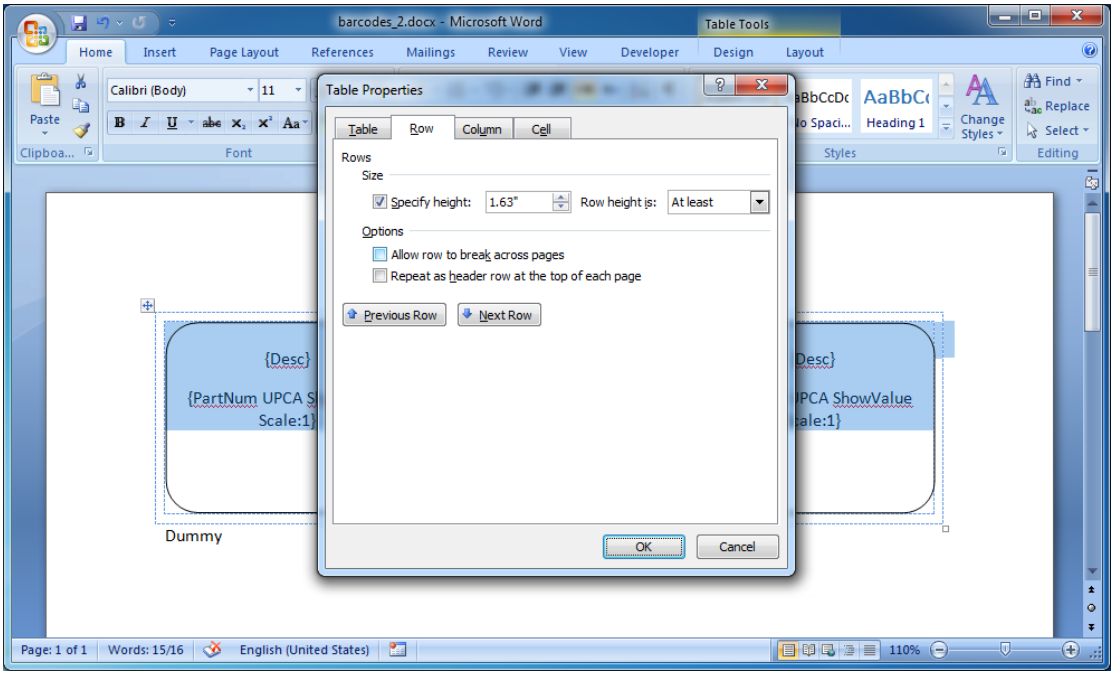
   Select No Fill in the Fill drop down menu in the Format ribbon (see picture below).



6. Repeat steps 2 to 5 for the second and third Cell items.

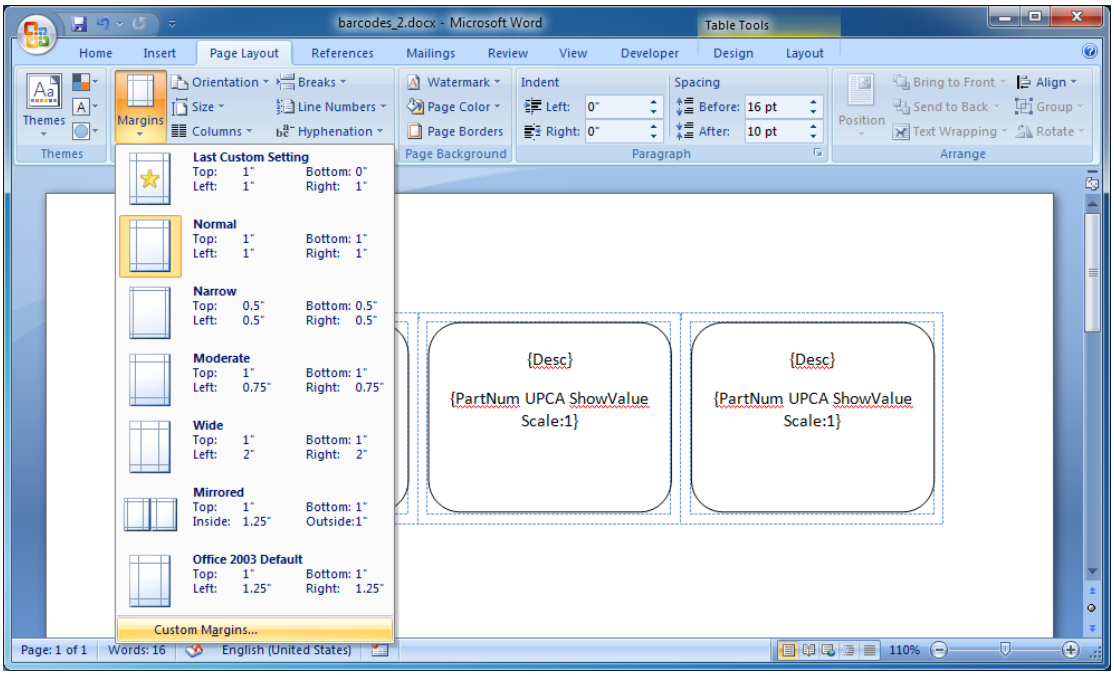
7. Unselect the Allow row to break across pages for the Row item so either the entire row fits in the page or the row is moved to the next page.

   Select the Row bookmark from the Bookmarks dialog box. Right click the highlighted area and select Table Properties from the drop down menu and unselect the Allow row to break across pages in the Row tab (see picture below).



8. Remove the page bottom margin. Word adds a paragraph after a table. If the table fills the page then the paragraph will overflow adding an empty page. You can remove or shorten the page bottom margin to make space for the paragraph.

   Select the Custom Margins from the Margins drop down menu in the Page Layout ribbon. Set the Bottom Margins: 0" in the Margins tab (see picture below).



9. Compile the template.

10. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;

int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/barcodes_2.dfw");

        l_merger.paste("Main");

        for (int i = 0; i < 20; i++)
        {
            int j = (i + 1) % 3;
            if (j == 0) j = 3;

            if (j == 1)
                l_merger.paste("Row");

            l_merger.setClipboardValue("", "Desc",      "Hello World");
            l_merger.setClipboardValue("", "PartNum",   "12345678901");

            l_merger.paste(string("Cell") + (char) ('0' + j))
        }

        l_merger.save("/tmp/barcodes_2.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

11. Open the created .DOCX file (see picture below).

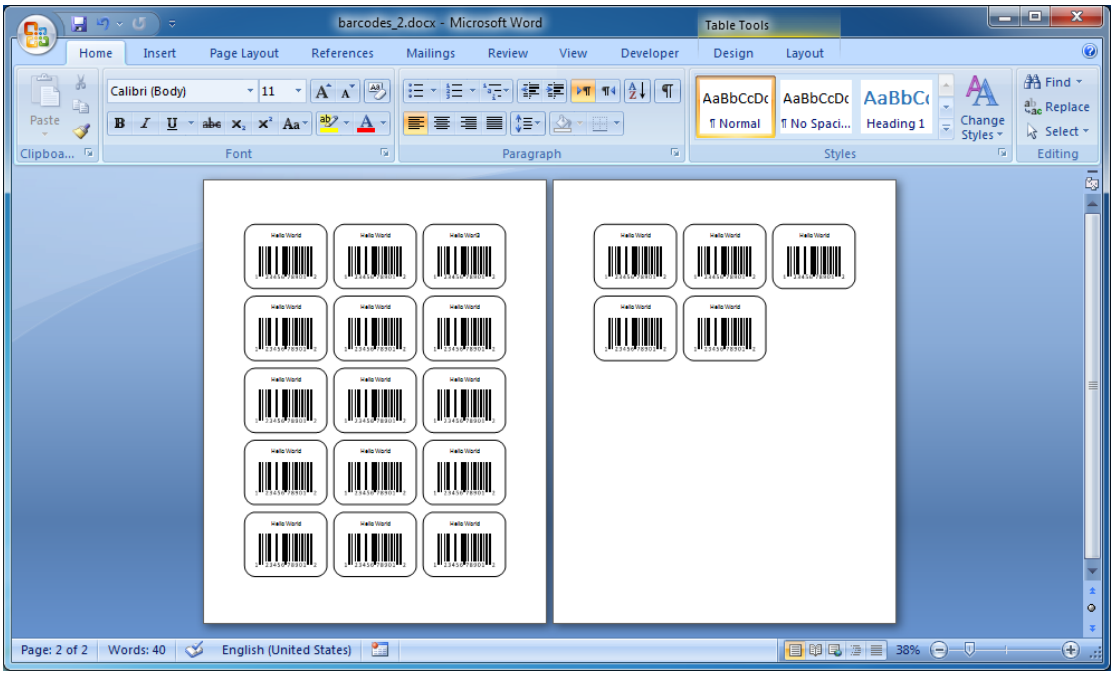


Chart Fields

Charts are another special field. A big difference between charts and other fields is that unlike other fields which only hold a single value, charts hold multiple values. In most cases, charts display a value as a column at the height of the value for a sequence of categories. Other series of values (for the same categories) can be displayed in a different color. So charts hold a matrix of values by categories and series.

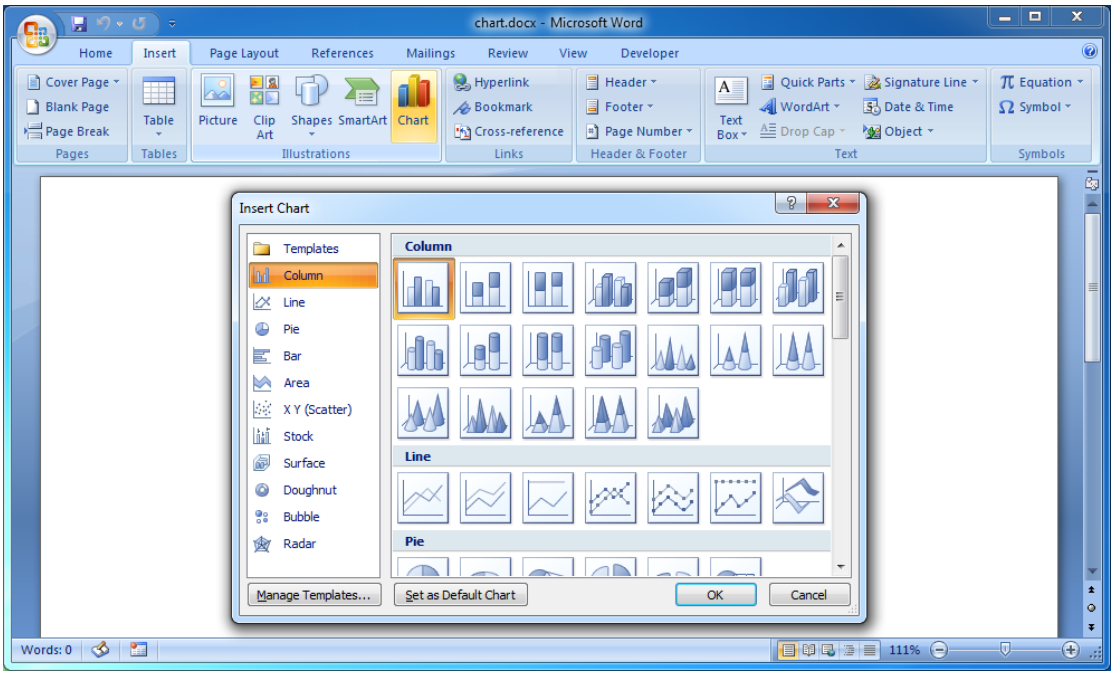
There are several exceptions:

1. Single series charts - That can only have a single series like Pie charts (Doughnut charts are not a single series charts like Pie charts because they can have multiple series represented by different rings).
2. Fixed series charts - That have a fixed number of series like Stock and Composite charts.
3. X, Y, Size charts - That hold a list of X, Y, Size values like Scatter and Bubble charts instead of a values matrix by series and categories.

Chart Fields Exercise

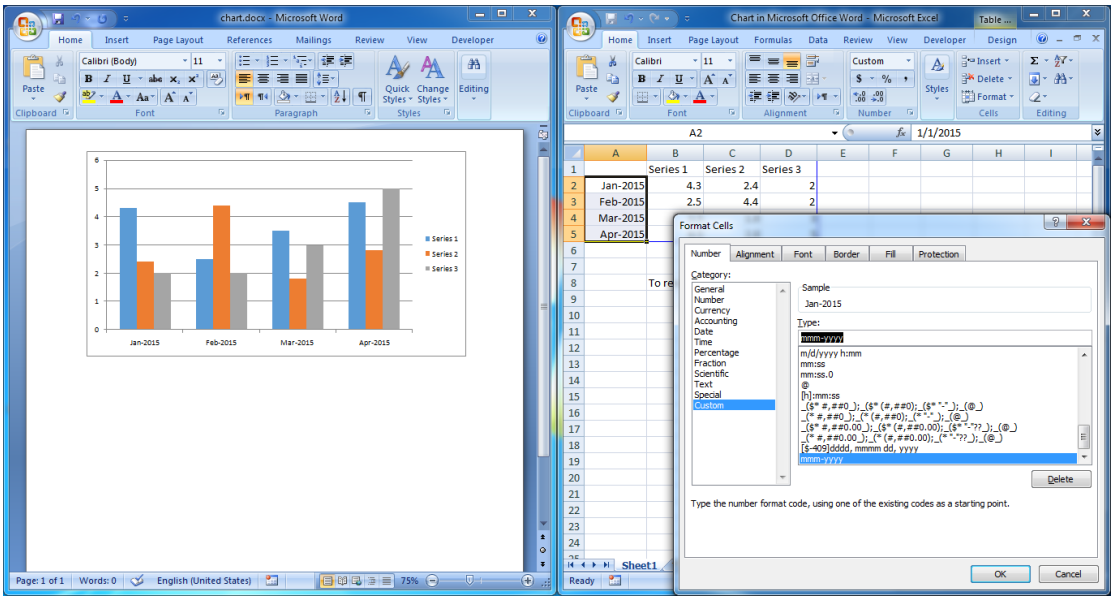
1. Open a new blank document and insert a column chart.

   Open a new blank document. Select Chart in the Insert ribbon to open the Insert Chart dialog box and select Clustered Column Chart from the Insert Chart dialog box (see picture below).



2. Change the categories data type and format.

   After you insert a chart the values matrix by categories and series is opened and can be edited in Excel. Insert date values for the categories and set the format to month and year (see picture below).

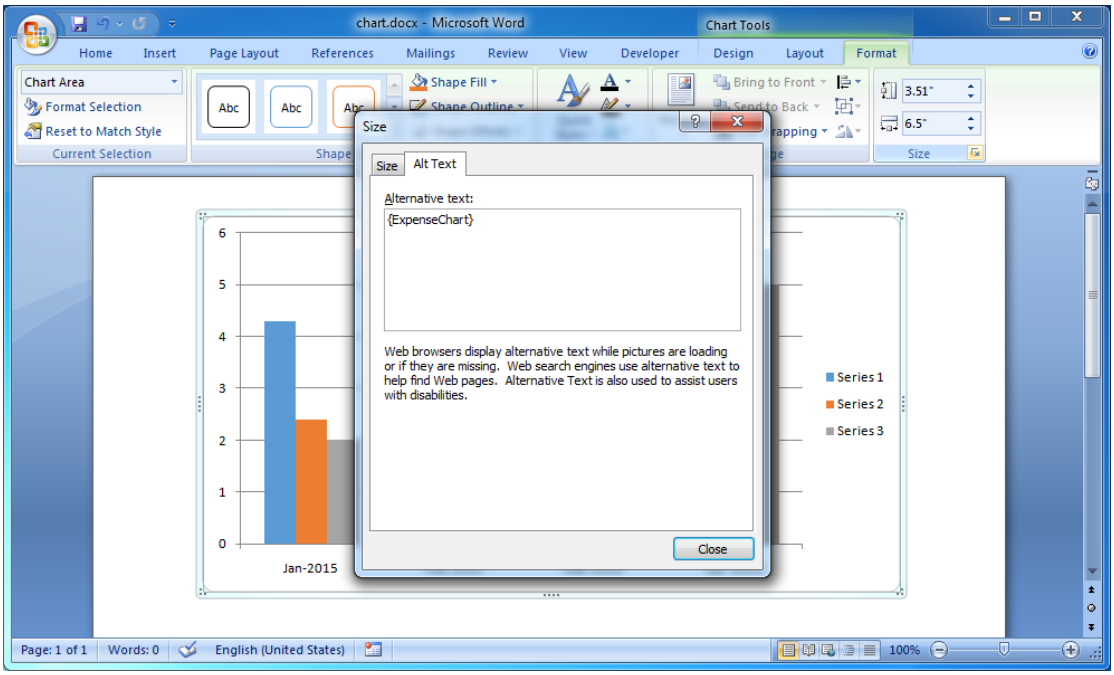


Just like all other fields, you set the data types and formats in the template and insert the values when creating a new document so all the values and number of series, categories and values will be removed and replaced with the values you insert but the data types and formats will remain.

The series are the names of series of values by categories so they can only be text values. The sequence of categories can be date, number or text values. The values are represented by column heights so they can only be number values. Except for X, Y, Size charts which hold a list of X, Y, Size number values.

3. Enter the chart field name.

   Select the chart. Click the corner of the Size group in the Format ribbon to open the Size dialog box. Select the Alt Text tab in the Size dialog box and enter "{ExpenseChart}" in the Alternative Text box (see picture below).



4. Compile the template.

5. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load("/opt/DocxFactory/exercises/templates/chart.dfw");

l_merger.setChartValue("", "ExpenseChart", "Income", "2015-01-01", 3600.0);
l_merger.setChartValue("", "ExpenseChart", "Income", "2015-02-01", 3600.0);
l_merger.setChartValue("", "ExpenseChart", "Income", "2015-03-01", 3600.0);
l_merger.setChartValue("", "ExpenseChart", "Income", "2015-04-01", 4200.0);
l_merger.setChartValue("", "ExpenseChart", "Income", "2015-05-01", 4200.0);

l_merger.setChartValue("", "ExpenseChart", "Expense", "2015-01-01", 1000.0);
l_merger.setChartValue("", "ExpenseChart", "Expense", "2015-02-01", 1200.0);
l_merger.setChartValue("", "ExpenseChart", "Expense", "2015-03-01", 1300.0);
l_merger.setChartValue("", "ExpenseChart", "Expense", "2015-04-01", 1100.0);
l_merger.setChartValue("", "ExpenseChart", "Expense", "2015-05-01", 2200.0);

        l_merger.save("/tmp/chart.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

Note: That no items were created in the template and no items were pasted in the code. DocxFactory supports templates that have only fields for such cases as simple letters.

The code introduces the setChartValue function in the WordProcessingMerger singleton (see details below).

DocxFactory::WordProcessingMerger::setChartValue Function

Sets a value in the chart field values matrix by category and series.

There are several exceptions:

1. Single series charts - For charts that can only have a single series like Pie charts, the setChartValue function ignores the series name parameter. The series name in the created document is the same series name in the template.

2. Fixed series charts - For charts that have a fixed number of series like Stock and Composite charts, the setChartValue function only accepts values for the fixed series that were in the template. Values for other series are ignored and are not set.

3. X, Y, Size charts - For charts that hold a list of X, Y, Size values like Scatter and Bubble charts, the setChartValue function treats the 1st parameter as the X value, the 2nd parameter as the Y value and the 3rd parameter as the Size value regardless of their data type.

Declaration**:**

```
1. void setChartValue(
        const string& p_itemName,
        const string& p_fieldName,
        const string& p_series,
        const string& p_category,
        double        p_value);

2. void setChartValue(
        const string& p_itemName,
        const string& p_fieldName,
        const string& p_series,
        double        p_category,
        double        p_value);

3. void setChartValue(
        const string& p_itemName,
        const string& p_fieldName,
        double        p_x,
        double        p_y,
        double        p_size);
```

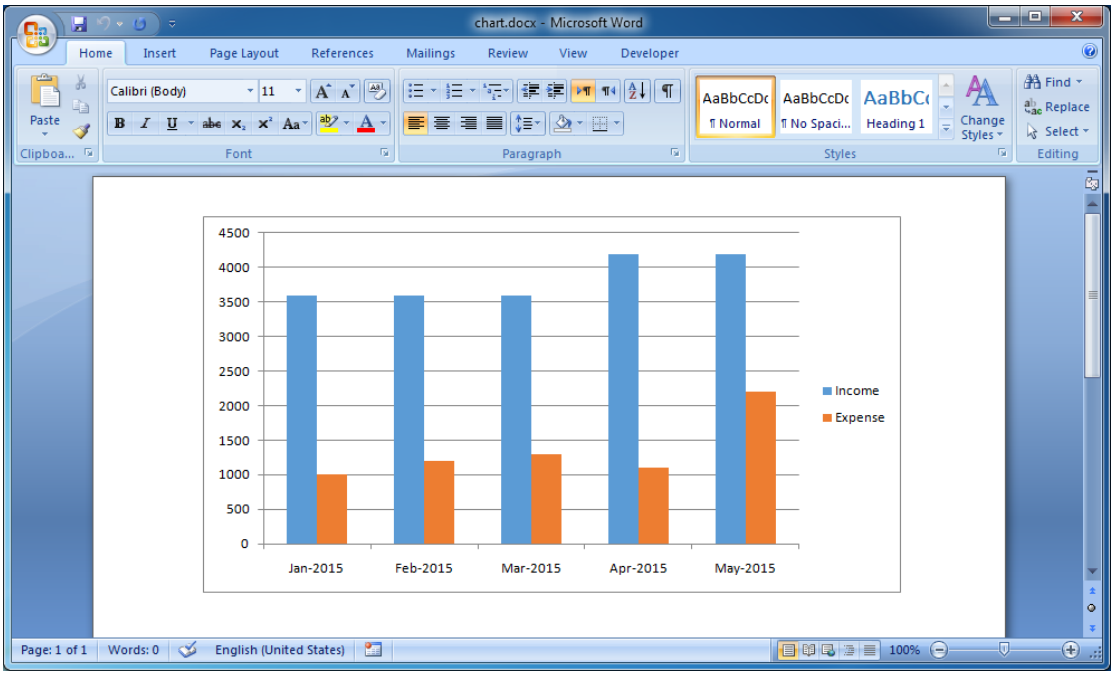
Parameters:

```
p_itemName   - Item name.
p_fieldName  - Field name.
p_series     - Series name.
p_category   - Category.
p_value      - Value.
p_x          - X.
p_y          - Y.
p_size       - Size.
```

Notes:

- The sort order of the chart series and categories in the created document is the order you add them to the values matrix.
- The series and categories (if the categories values are text) are not case sensitive. For example: the series "Income" and "income" refer to the same series but the chart series and categories in the created document use the series and categories casing when they were first added with the exception of single series charts and fixed series charts that keep the series casing in the template.
- Every time you add a new column or row to the values matrix by adding a new series or category the new values are initialized with a value of zero. For example: if you add a new series and not set one of the values for a category for that series then its value will be zero.
- An empty chart with no values cannot be shown so if no values are set for a chart field then the chart will not be in the created document.
- If a value for the same category and series is set more than once then the value is accumulated so you can set a chart field from a detailed list that will be summarized by categories and series.
- The setChartValue function is the equivalent setClipboardValue function for chart fields and has the same rules for item and field casing, setting date and time values and converting between different field and parameter data types.

6. Open the created .DOCX file (see picture below).



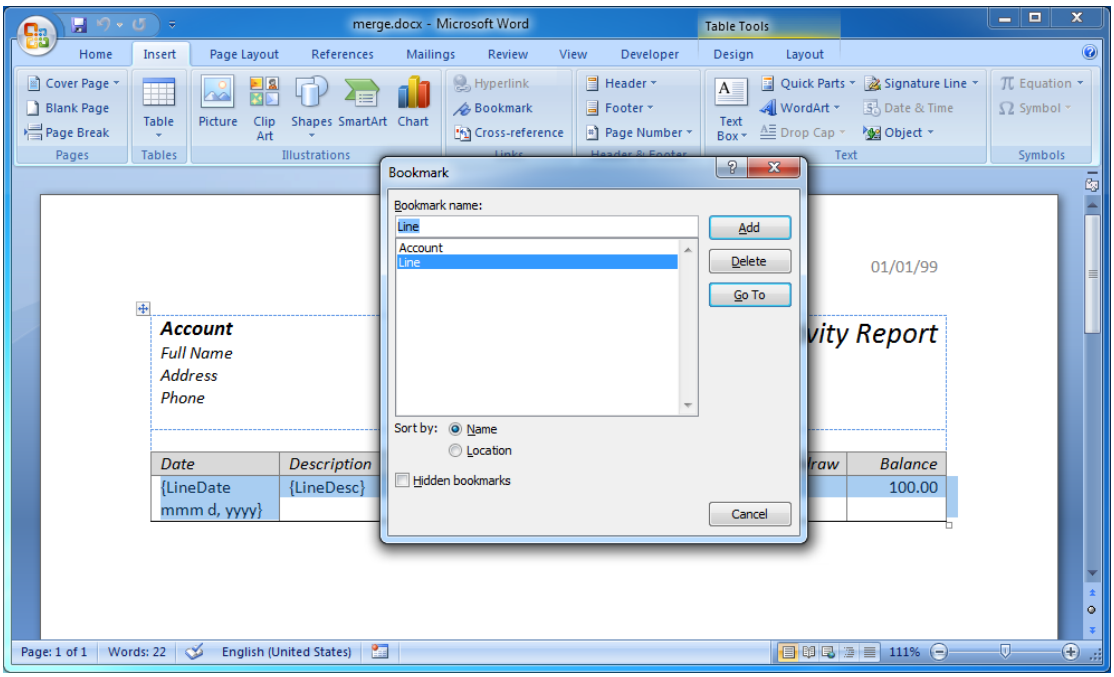
Merge XML and JSON

DocxFactory allows you to merge XML and JSON (and any data structure that you can convert to XML and JSON) in a single merge function instead of using multiple setClipboardValue and paste functions.

Merge XML and JSON Exercise

1. Open merge.docx in the DocxFactory/exercises/templates/ directory.

   Open the Bookmark dialog box to view the bookmarks. The template continues the fields exercise (see picture below).



2. Compile the template.

3. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load("/opt/DocxFactory/exercises/templates/merge.dfw");

        l_merger.merge(
"<Accounts>"
"<Account>"
"<Line>"
"<LineDate>1/1/99</LineDate><LineAmt>0</LineAmt><LineDesc>Desc0</LineDesc>"
"</Line>"
"<Line>"
"<LineDate>1/1/99</LineDate><LineAmt>1</LineAmt><LineDesc>Desc1</LineDesc>"
"</Line>"
"</Account>"
"</Accounts>");

        l_merger.merge(
"{\"Account\": [{"
    "\"Line\": [{"
        "\"LineDate\": \"1/1/99\", \"LineAmt\": 2, \"LineDesc\": \"Desc2\""
    "}, {"
        "\"LineDate\": \"1/1/99\", \"LineAmt\": 3, \"LineDesc\": \"Desc3\""
    "}]"
"}]}");

        l_merger.save("/tmp/merge.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }
```

```
    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code introduces the merge function in the WordProcessingMerger singleton (see details below).

DocxFactory::WordProcessingMerger::merge Function

Merges XML or JSON with the template. The merge function recognizes if the string passed to the function is XML or JSON from the beginning of the string.

When merging XML, DocxFactory drills down the entire XML document. When it reaches a node with the same name as an item, it reads all the node children with the same name as the item fields and sets the field clipboard value with the node content, it then pastes the item and continues drilling the rest of the children (that are not field nodes).

When merging JSON, DocxFactory drills down the entire JSON document. When it reaches an object property with the same name as an item or an object in an array property with the same name as an item, it reads all the value properties (string, number, boolean or null values) with the same name as the item fields and sets the field clipboard value with the property value, it then pastes the item and continues drilling the object.

Declaration**:**

```
void merge(const string& p_data);
```

Parameters:

```
p_data – XML or JSON string.
```

Merge XML Notes:

- To make the merge simple and flexible, DocxFactory finds item nodes only if the node name matches an item name and does not take into consideration the node parents so you can put the node in another container node (for example: <items><item/>…</items>) or not put the node in any other node (except the document root node) and it would not change the result.

- When reading an item node, if a child node has the same name of both an item field and another item, DocxFactory will treat the node as a field because there is no way of knowing if the node is meant to be used as a field or an item. To avoid confusion between items and fields, do not name items and fields with the same name.

- To set a chart field, in the field node insert a list of point nodes with a series, category and value nodes and their values. For example: <chart><point><series>Amount</series><category>01/01/1999</category><value>100</value></point>…</chart>. You can also use <x>, <y> and <size> tags instead of <series>, <category> and <value> tags for X, Y and Size charts.

- DocxFactory supports CDATA sections for entering data that is not parsed and may contain illegal XML characters (like "<" and "&") without the need to escape these characters. You can use CDATA to set HTML and RTF field values or any field values that may contain illegal XML characters without the need to escape these characters.

- When setting a field value from the content of a node, the complete inner node content is used including XML tags not just text which allows you to enter HTML tags for HTML fields. Note that even though HTML in some cases may not be well formed, the node content must be well formed XML otherwise you will have to use a CDATA section.

- To enter new lines in field values, use a new line character (ASCII #10) or escape the character using the "
" XML entity. "\n" is not a character escape sequence in XML.

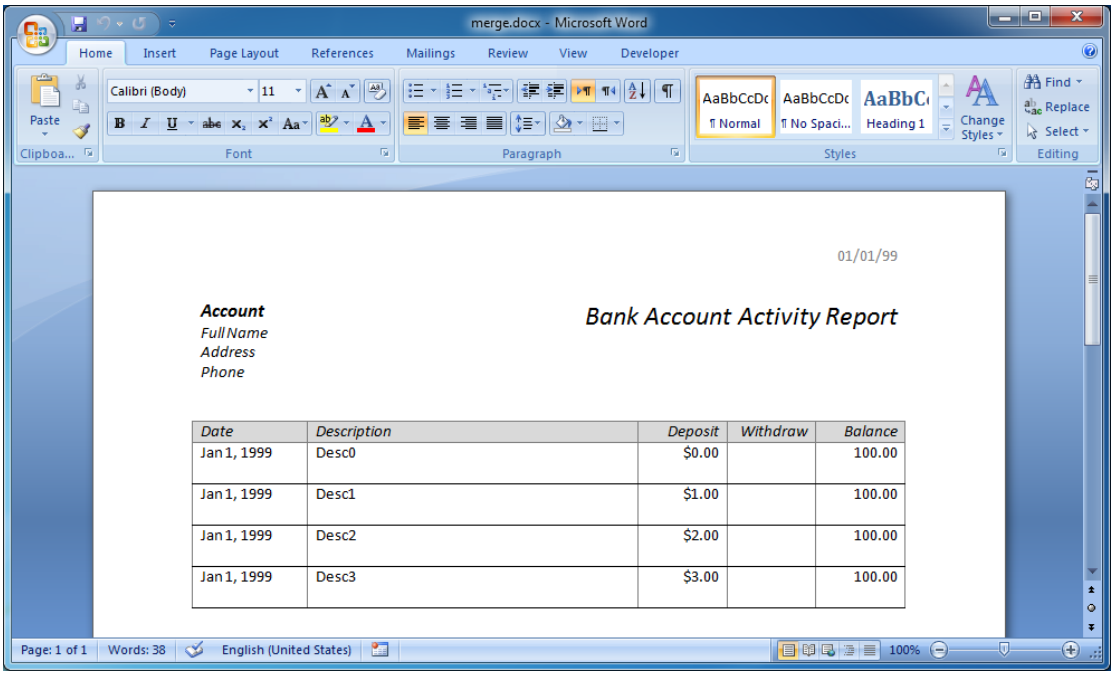
- The item names are not case sensitive. For example: the tag name “<Account>” and “<account>” refer to the same item.

- The field names are not case sensitive. For example: the tag name “<LineDate>” and “<linedate>” refer to the same field.

Merge JSON Notes:

- Just like merging XML, DocxFactory finds item objects only if the object property name matches an item name and does not take into consideration the object parents so you can put the object in another container object or even put all the items in object properties in an array of objects in the document root object (for example: {"items": [{"item": {}}, ...]}) and it would not change the result.
- To set a chart field, in the field property value enter an array of objects with a series, category and value properties and their values. For example: "chart": [{"series": "Amount", "category": "01/01/1999", "value": 100}, …]. You can also use "x":, "y": and "size": properties instead of "series":, "category": and "value": properties for X, Y and Size charts.
- The JSON property names must be enclosed in quotes according to the JSON specification otherwise a JSON parsing error will be returned.
- The item names are not case sensitive. For example: the property name "Account": and "account": refer to the same item.
- The field names are not case sensitive. For example: the property name "LineDate": and "linedate": refer to the same field.

4. Open the created .DOCX file (see picture below).

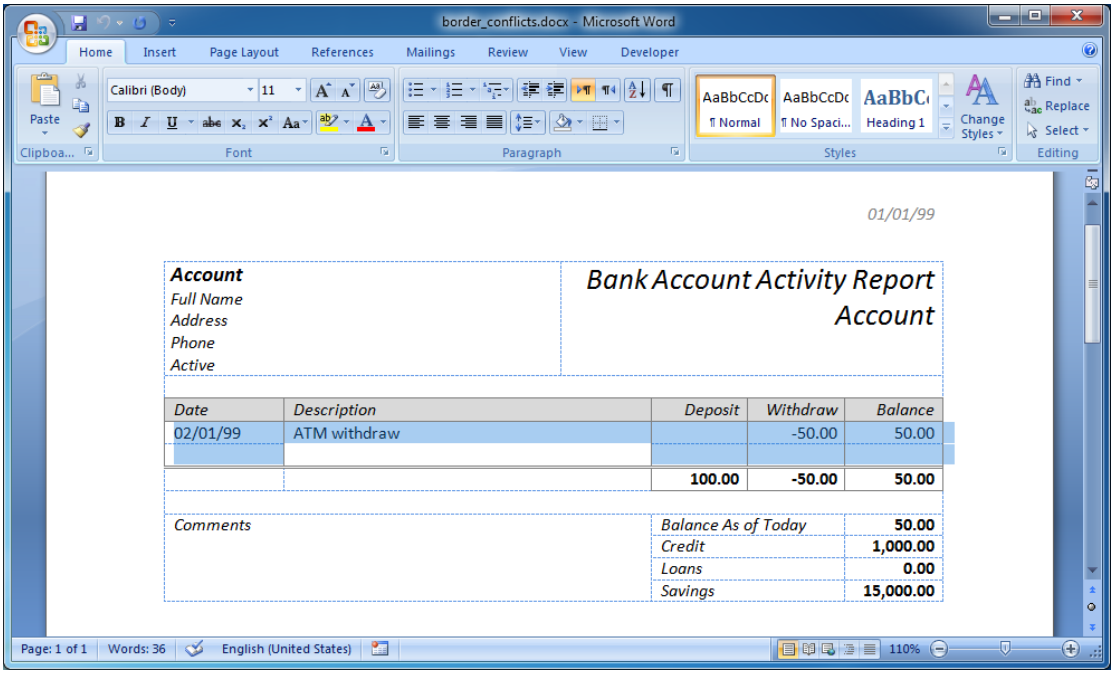


Border Conflicts

This chapter shows a common problem and its solution.

Border Conflicts Exercise

2. Open border_conflicts.docx in the DocxFactory/exercises/templates/ directory, open the Bookmarks dialog box and highlight the Line item (see picture below).



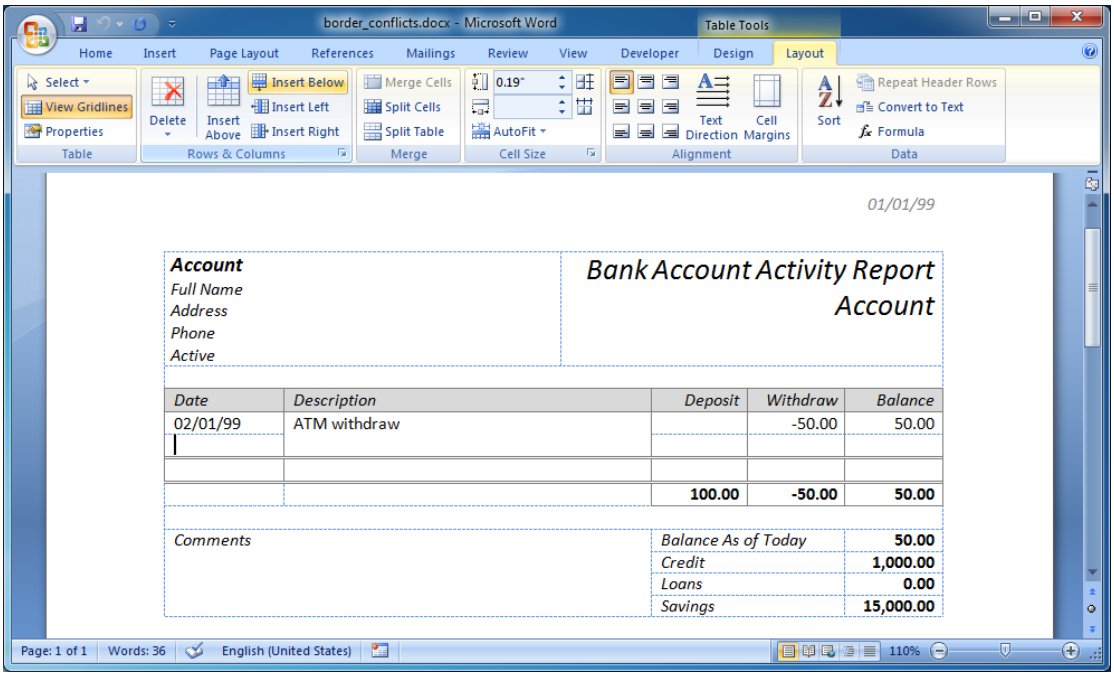
The problem is that the Line item shares a border with the report summary and there is no separate bottom border for the Line item and a separate top border for the report summary, to set with different border styles.

To fix this problem create a dummy item between them. The dummy item is not pasted and is only used to separate the items. After the dummy item is added there is a separate bottom border for the Line item and a separate top border for the report summary that can be set with different border styles.

When creating a new document and pasting items, if items meet on the same border with different border styles then, in general the bigger border will overtake the smaller border (the thicker border, the border with the most lines etc.).

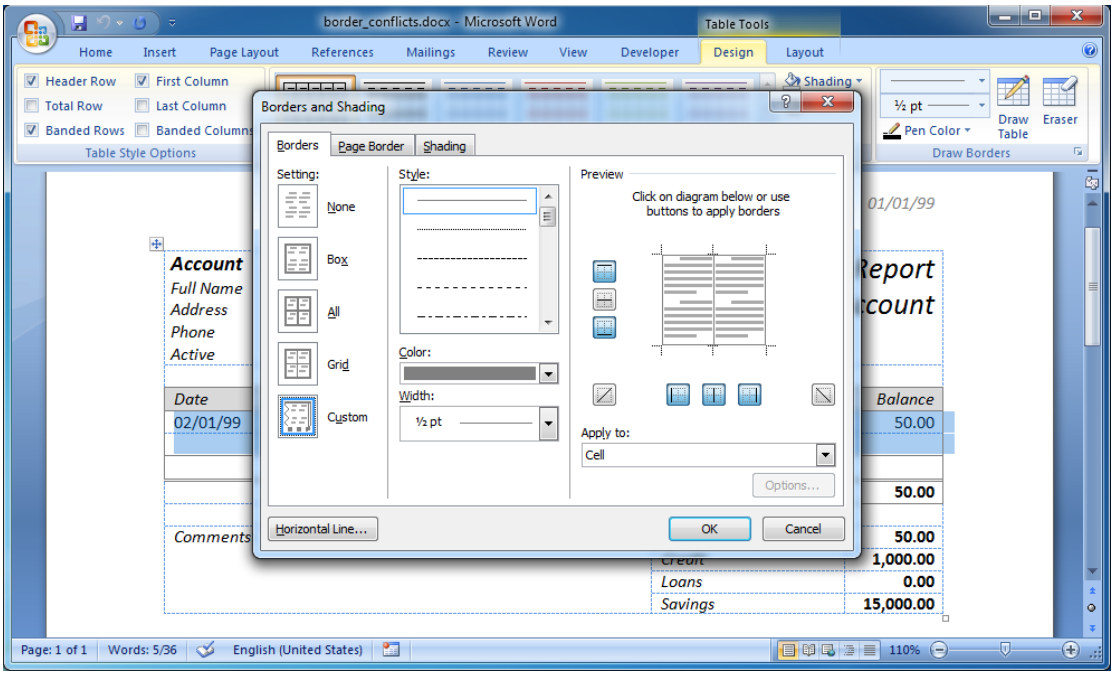
3. First, insert a new row between the items.

   Place the cursor on the bottom of the Date column and select Insert Below in the Layout ribbon (see picture below).



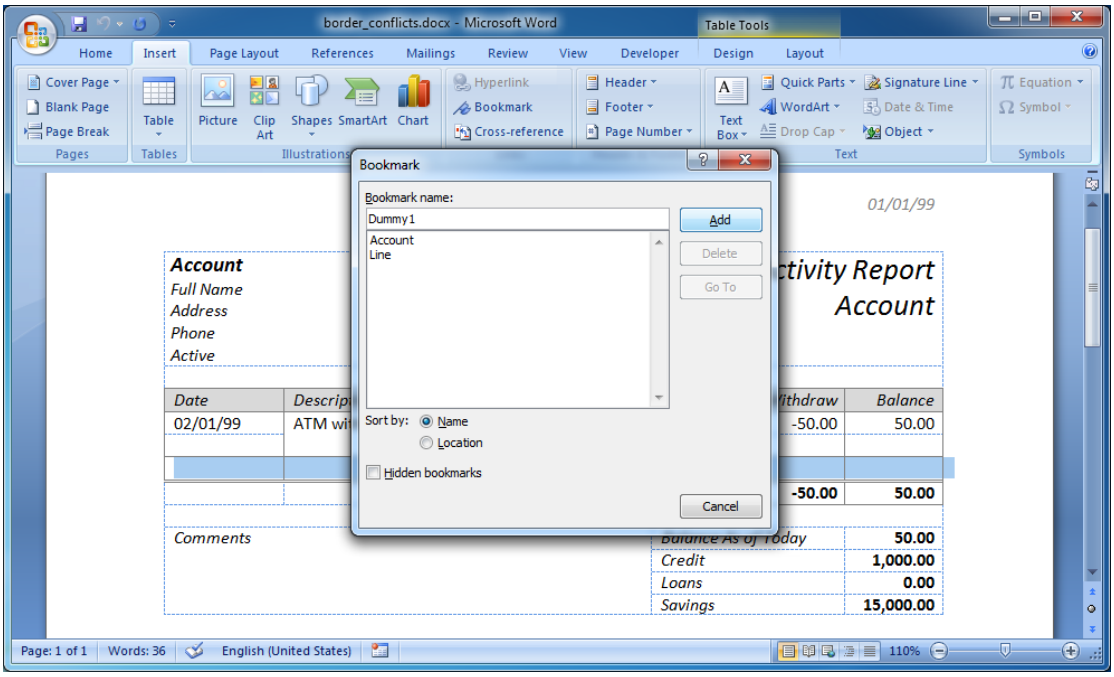
4. Then change the Line item bottom border.

   Highlight the Line item, right click the highlighted area and select Borders and Shading from the drop down menu. In the Borders and Shading dialog box, change the bottom border to a single line (see picture below).



5. Finally, create a bookmark for the dummy item (see picture below).

   Dummy items are not pasted or referenced and there is no need to give them meaningful names. Dummy items are named “Dummy<n>” by convention.



6. Compile the template.

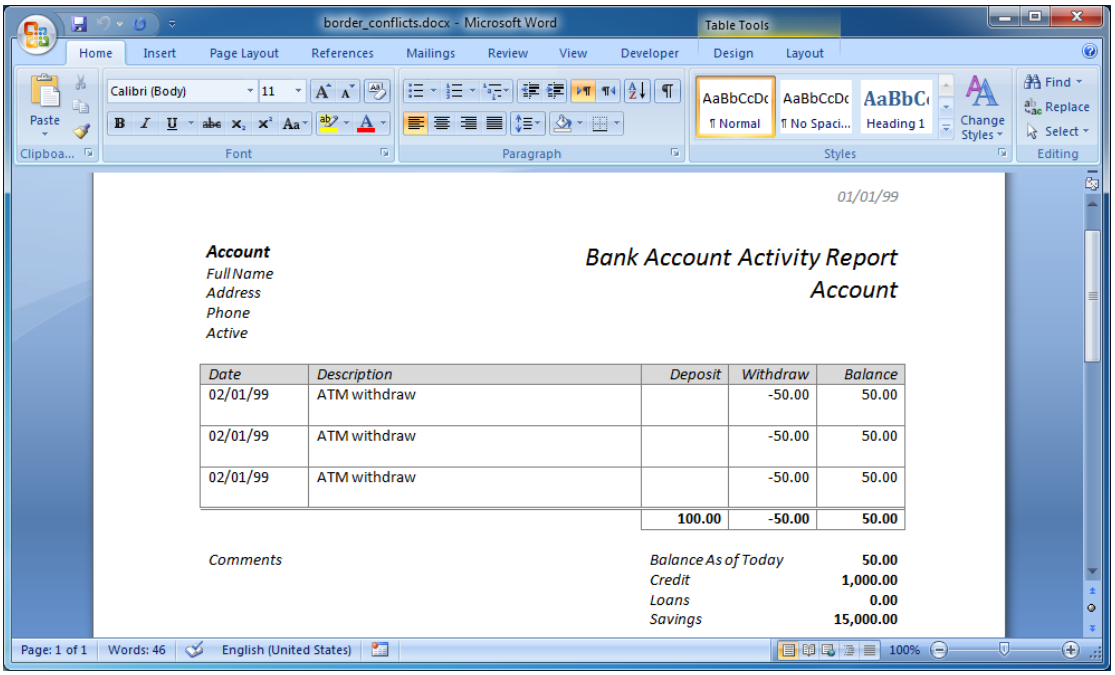
7. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/border_conflicts.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Line");
        }

        l_merger.save("/tmp/border_conflicts.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

8. Open the created .DOCX file.

   As you can see, the Line item bottom border has a different border style than the report summary top border style and on the border where the Line item and the report summary meet the report summary overtakes the Line item border style (see picture below).



Alternating Colors

Items marked as alternating color in the template will alternate between two colors in the new document.

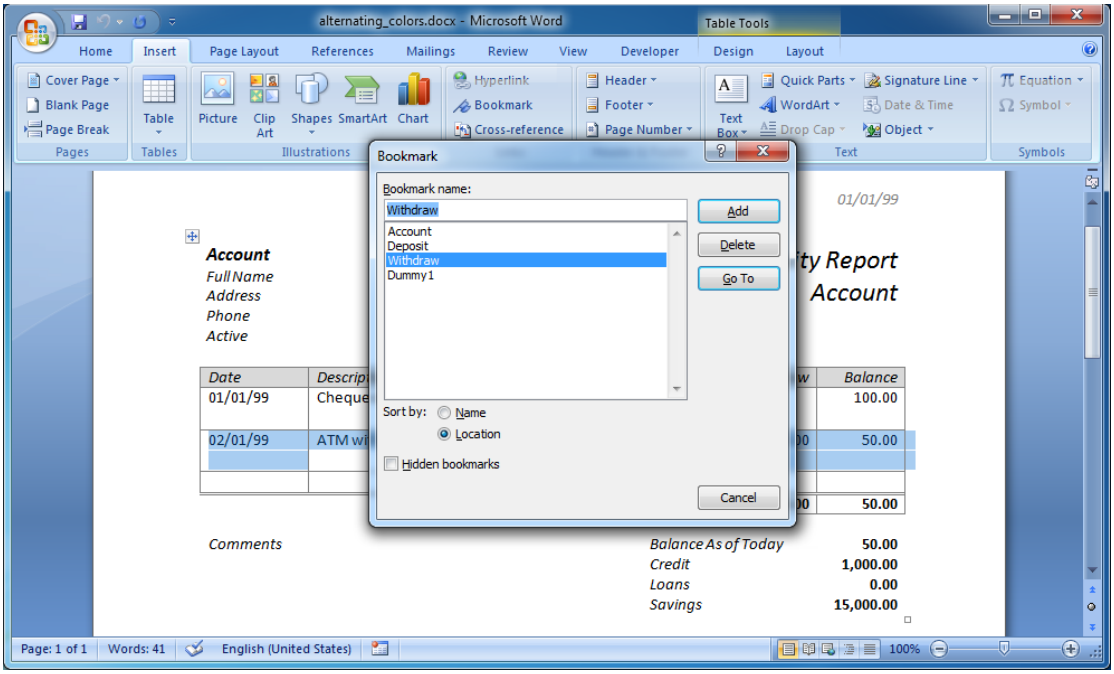
The two alternating colors are taken from the color of the first table cell in the first table row of the first and second items in the group (alternating colors is currently only supported for tables). If there is only one item then the second color is no color.

The items in the group (that are marked as alternating color) color will alternate between the two colors in the new document.

Alternating Colors Exercise

1. Open alternating_colors.docx in the DocxFactory/exercises/templates/ directory (see picture below).

   Note: Continuing from the previous chapter, the template has a dummy item to separate between the Withdraw item and the report summary.



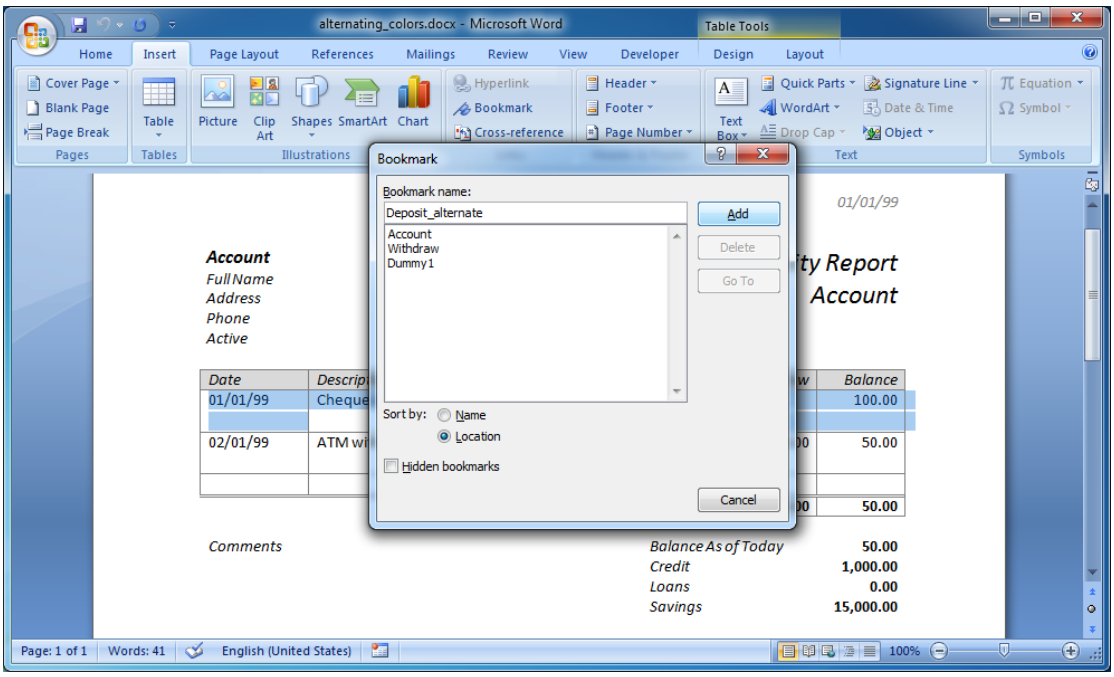
2. Add _alternate to the bookmark name to mark the item as alternating color.

   You cannot change a bookmark name so you will need to delete and re-enter the bookmark with a different name.

   1. In the bookmark Dialog box, highlight the bookmark by selecting the bookmark and pressing the Go To.

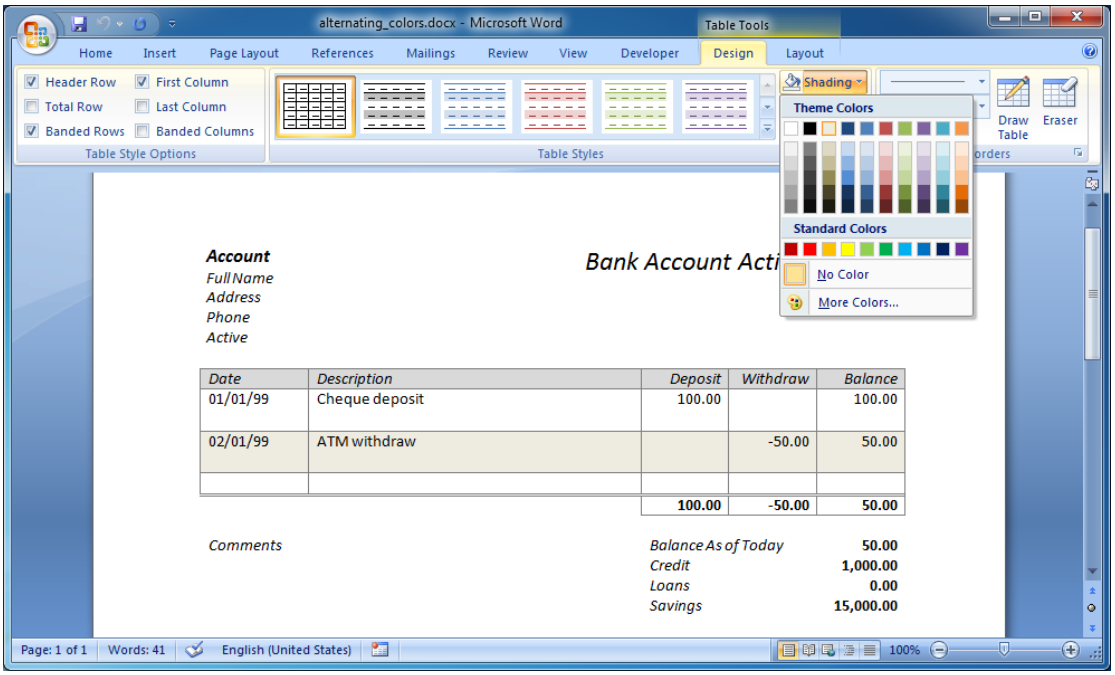
   2. Delete the bookmark with the Delete button.

   3. Re-enter the bookmark name and add _alternate to the name. Press the Add button to add the bookmark for the highlighted area (see picture below).



3. Change the color of the second item so the color of the first and second items is different.

   First, highlight the Withdraw item (you can either highlight the area manually or using the Bookmark dialog box). Then select a light gray from the Shading drop down menu in the Design ribbon (see picture below).



The exercise introduces the alternate feature (see details below).

_alternate

_alternate marks an item as alternating color.

Note: Item features are written to the end of the bookmark name.

Note: The features in the bookmark name are removed from the item name.
For example: The bookmark "MyItem_alternate" item name is "MyItem".

Note: The features in the bookmark name like the item names are not case sensitive.
For example: "_alternate" and "_Alternate" refer to the same feature.

4. Compile the template.

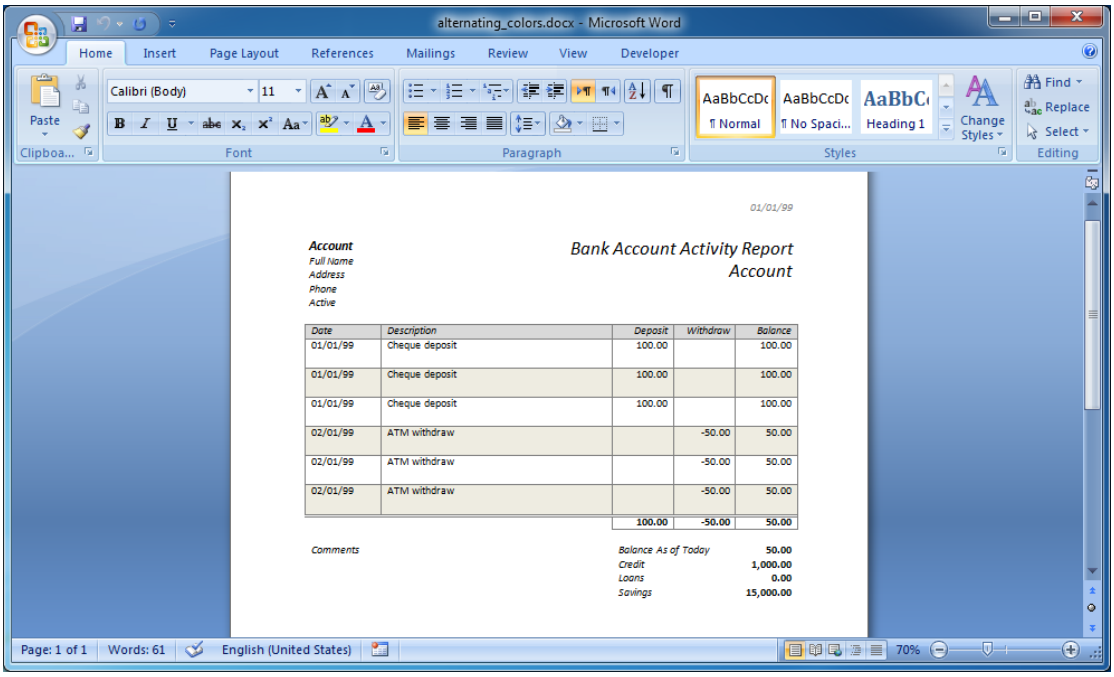
5. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/alternating_colors.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Deposit");
        }

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Withdraw");
        }

        l_merger.save("/tmp/alternating_colors.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

6. Open the created .DOCX file.

   As you can see, in the new document the color of the items in the group alternates between the two colors (see picture below).



Sections

In Microsoft Word, documents can be divided into sections and you can set a different page size and page orientation (but also other settings like header and footer) for every section. To divide a document into sections, insert a section break (which ends the previous section and starts a new section).

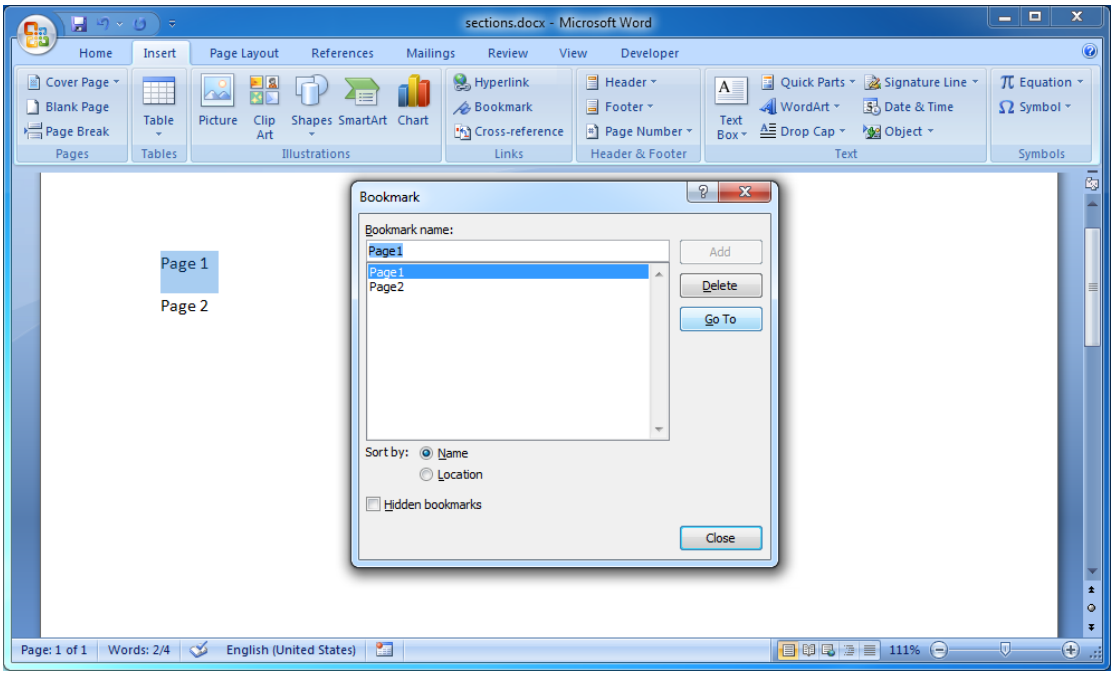
A single page cannot have two or more sections because every section has a different page type (page size and page orientation) so a section break (which ends the previous section and starts a new section) is also a page break. Just like regular page breaks, items should be placed between section breaks (or regular page breaks) and section breaks should not be placed inside items (see the chapter on top level items for a detailed explanation).

When designing a template, you set in what section the page the top level item starts is in. When creating a new document, if a new page is started when pasting a top level item that is in a different section than the last page then a section break is inserted and the same section page size and page orientation as the new page section are set. So the page the top level item starts is the same page type (page size and page orientation) in the created document and the template no matter what order the top level item is pasted.

Sections Exercise

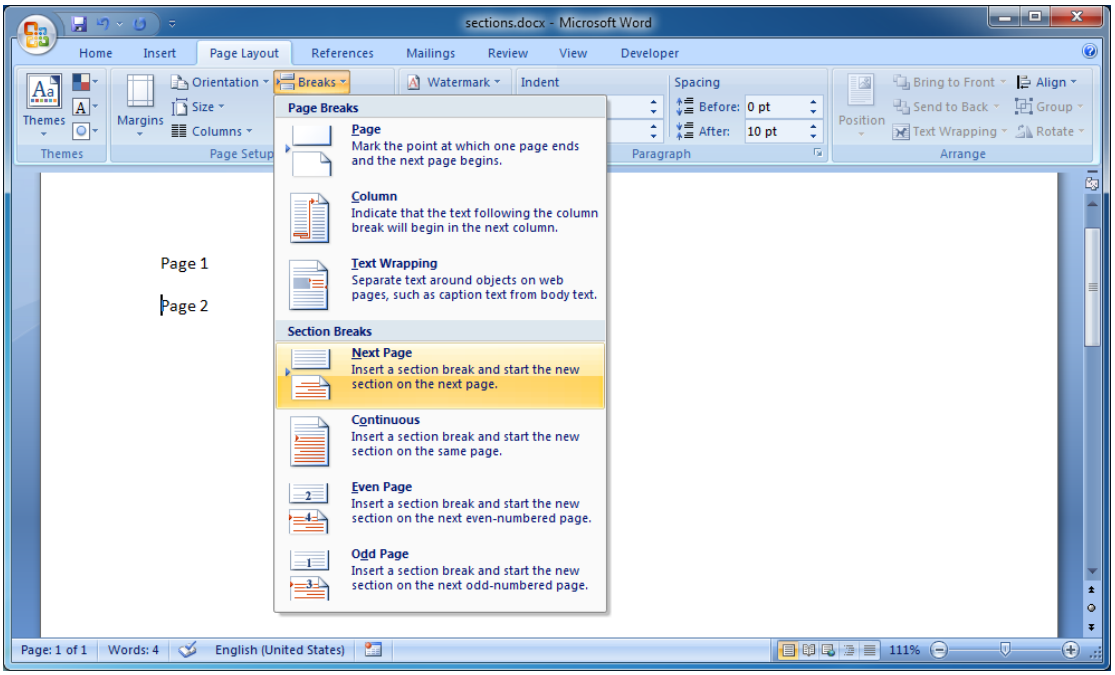
1. Open sections.docx in the DocxFactory/exercises/templates/ directory.

   Open the Bookmark dialog box to view the items in the template. There are 2 top level items: Page1 and Page2 (see picture below).



2. Insert a section break between the two items.

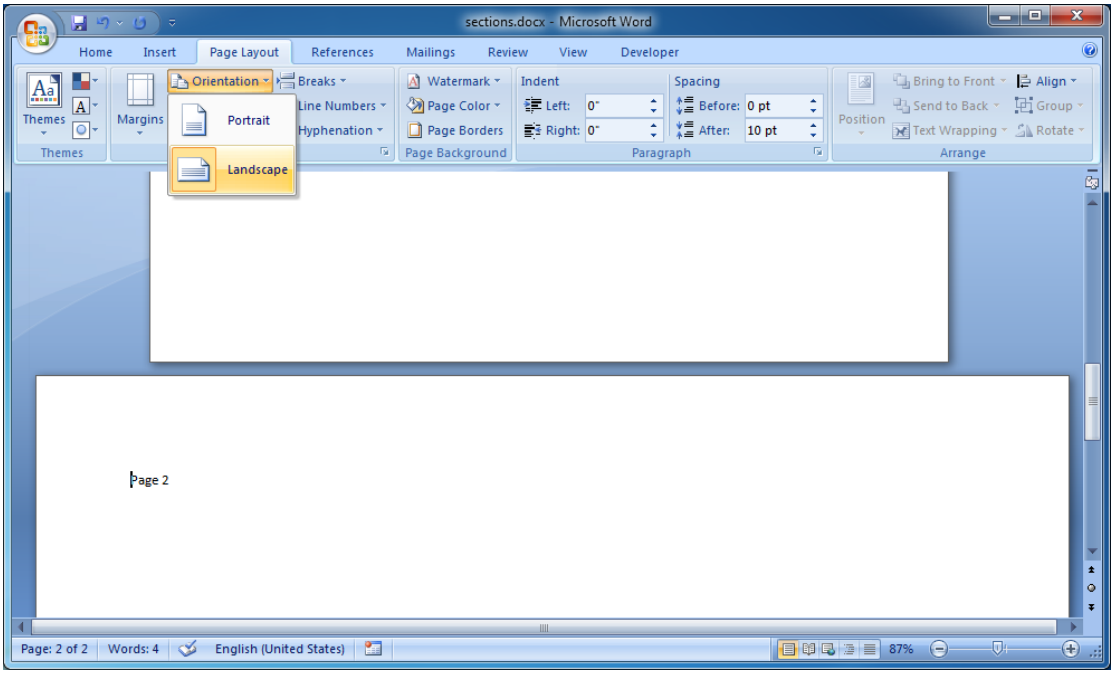
   First place the cursor at the beginning of the second item. Then select Next Page, under Section Breaks, in the Breaks drop down menu, in the Page Layout ribbon (see picture below).



As you can see, a section break is also a page break.

3. Change the section page orientation to landscape.

   First make sure the cursor is on the second page. Then select Landscape, in the Orientation drop down menu, in the Page Layout ribbon (see picture below).



4. Compile the template.

5. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/sections.dfw");

        l_merger.paste("Page2");
        l_merger.paste("Page1");
        l_merger.paste("Page2");
        l_merger.paste("Page1");

        l_merger.save("/tmp/sections.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

6. Open the created .DOCX file.

   As you can see, the page the top level item starts is the same page type (page size and page orientation) in the created document and the template no matter what order the top level item is pasted (see picture below).

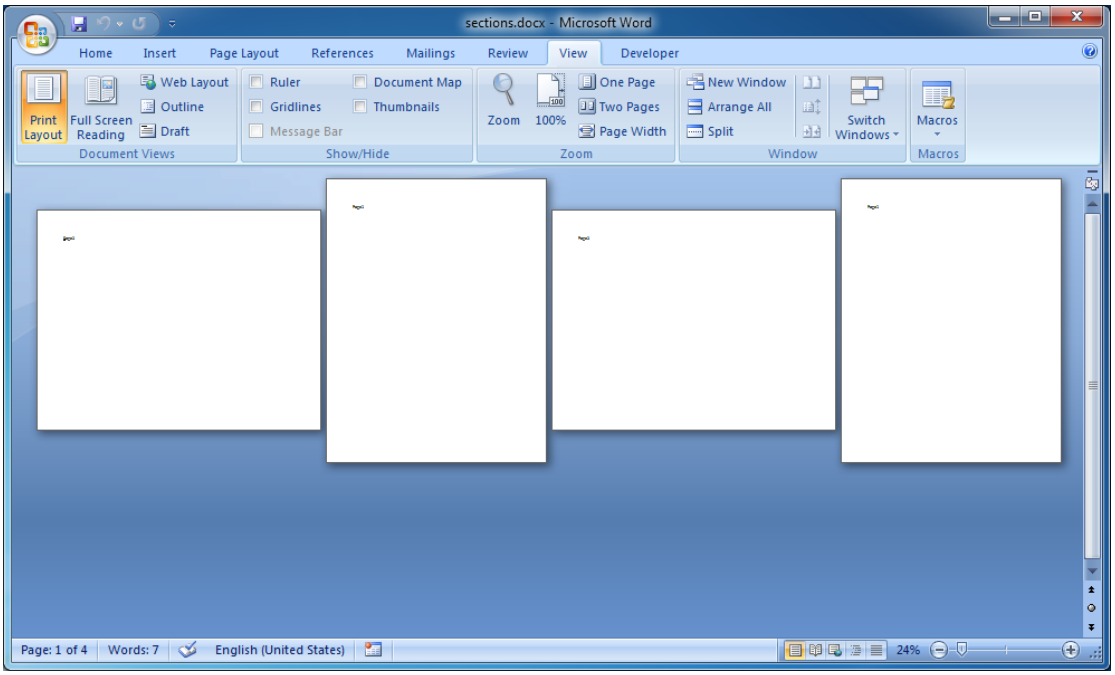


Table of Contents Automatic Update

In general, to find the page numbers in the table of contents the entire document has to be graphically rendered and the only way to do that is to open the document in Microsoft Word. Specifically, there are 4 possible ways to update the table of contents but none of them are without problems:

1. Mark the update fields flag in the document that tells Word to update the table of contents when opening the document. The problem with this approach is that a popup message will open when opening the document prompting the user to confirm the update instead of updating the table of contents automatically when opening the document.

2. Use Word Automation Services to open the document in Word, update the table of contents and save the document in the background. The problem with this approach is that opening a document, updating the table of contents and saving the document can take substantially longer than creating the document, from half a second to several seconds depending on the size of the document.

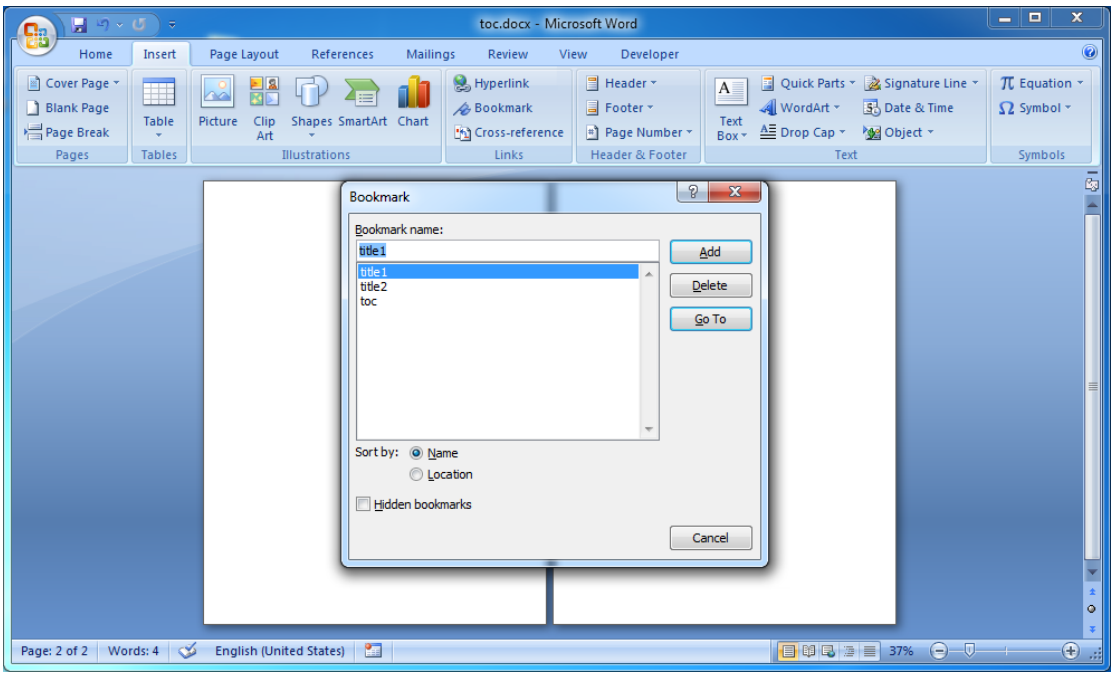
3. Add a macro to the Normal.dotm (which is the base template for new documents and is opened when Word opens) that updates the table of contents when opening a document. The problems with this approach are that: 1. The macro has to be added to the Normal.dotm of every Word installation and 2. The macro will update the table of contents of every document opened in Word.

4. Add a macro to the document that updates the table of contents when opening the document. Just like marking the update fields flag, a security popup message will also open when opening the document prompting the user if it is safe to run the macro. Because this approach basically behaves the same as marking the update fields flag, DocxFactory currently has no support for this approach.

Table of Contents Automatic Update Exercise

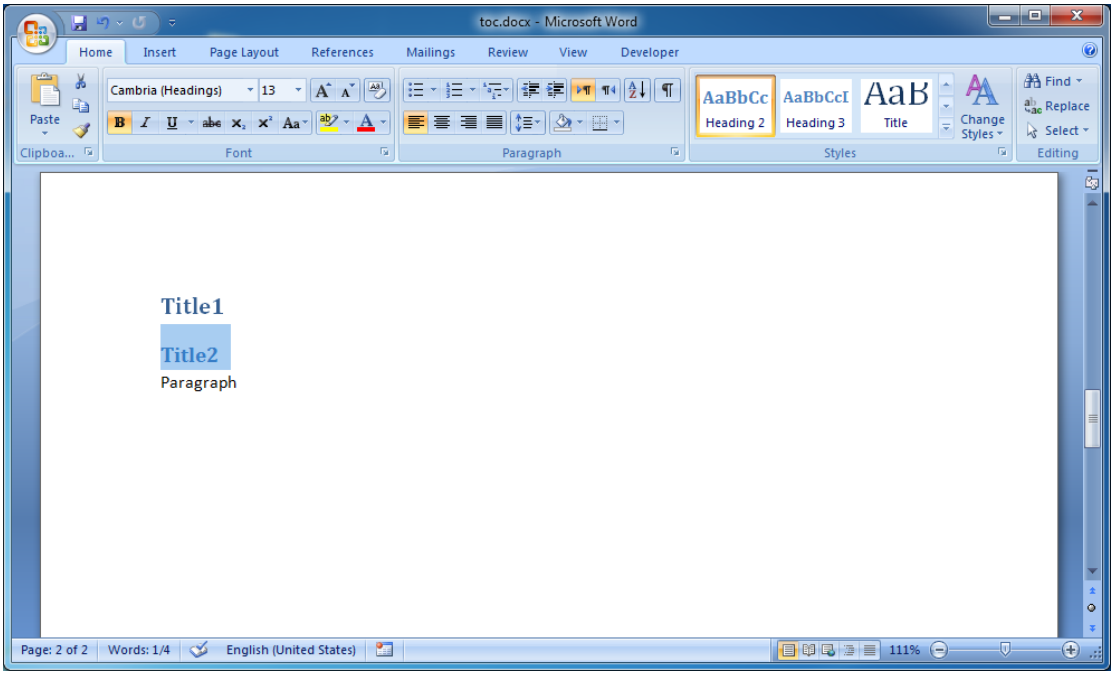
1. Open toc.docx in the DocxFactory/exercises/templates/ directory.

   Open the Bookmark dialog box to view the items in the template. There is a “toc” top level item for the table of contents page and a “title1” top level item with a “title2” sub level item with titles that will be in the table of contents (see picture below).



2. The "Heading" styles mark the titles in the document that will be in the table of contents. The biggest "Heading 1" style marks the highest level title down to the smallest "Heading 5" style that marks the lowest level title. Although the table of contents does not show titles lower than 3 levels.

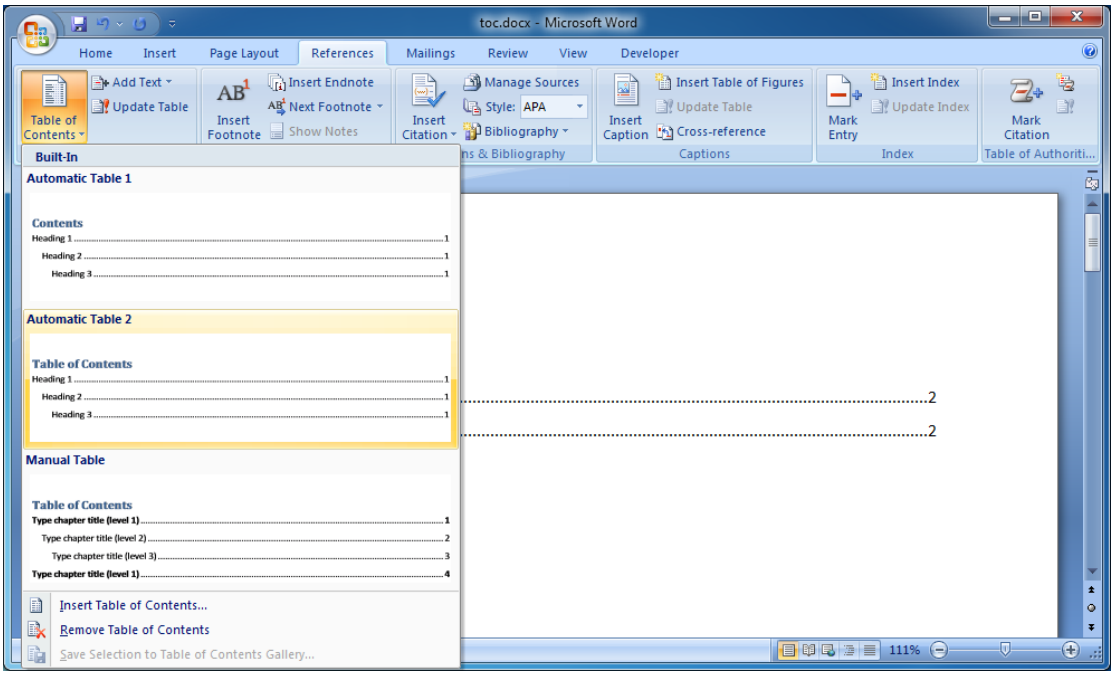
   Highlight the “Title1” paragraph and select the “Heading 1” style from the Styles group in the Home ribbon. Do the same from the “Title2” paragraph with the “Heading 2” style (see picture below).



Note: The “Heading” styles and all the styles in the document, font, size, color etc. can be modified by right-clicking the style in the Styles group and selecting Modify.

3. Insert the table of contents.

   First place the cursor in the “toc” top level item at the beginning of the document. Then select the Automatic Table 2 from the Table of Contents drop down list in the References ribbon (see picture below).



Note: If the table of contents direction is right-to-left and you would like it to be left-to-right or vice versa then open the Page Setup dialog box in the Page Layout ribbon, select the Layout tab and set the direction in the Section Direction field.

4. Compile the template.

5. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        l_merger.setUpdateTocMethod(1);

        time_t l_start = clock();

        l_merger.load("/opt/DocxFactory/exercises/templates/toc.dfw");

        l_merger.paste("toc");

        for (int i = 0; i < 2; i++)
        {
            l_merger.paste("title1");

            for (int j = 0; j < 3; j++)
            {
                l_merger.paste("title2");
            }
        }

        l_merger.save("/tmp/toc.docx");

        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

The code introduces the setUpdateTocMethod function in the WordProcessingMerger singleton (see details below).

DocxFactory::WordProcessingMerger::setUpdateTocMethod Function

Sets the method to use to update the table of contents if there is a table of contents in the document.

Declaration:

```
void setUpdateTocMethod(unsigned char p_method);
```

Values:

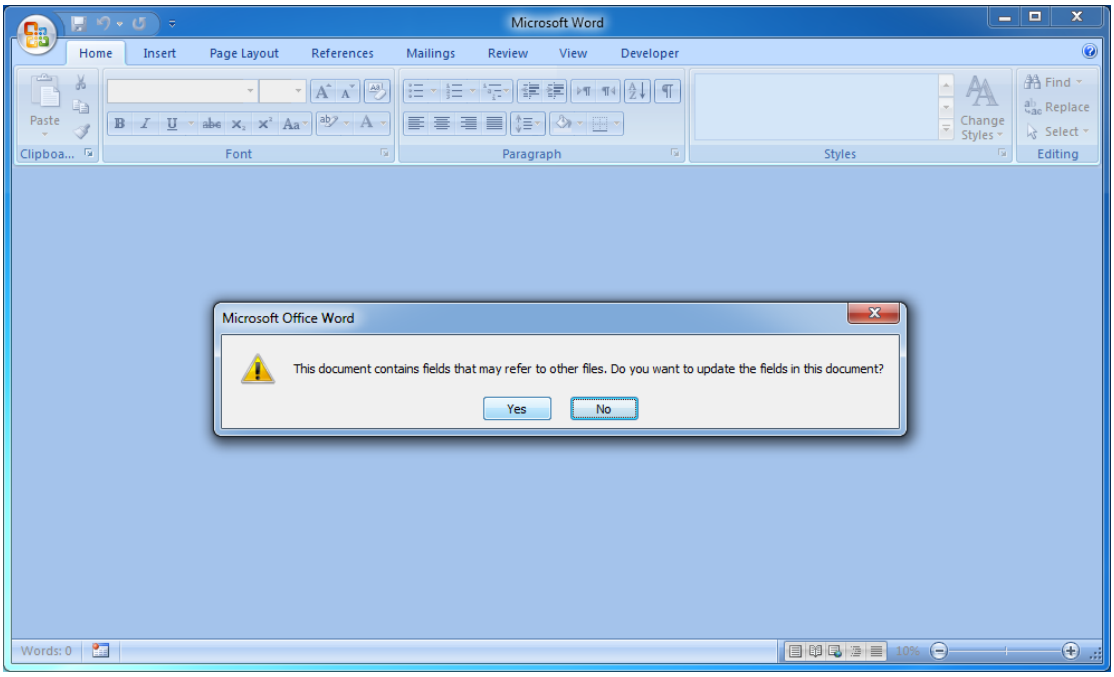
```
0 – Turns off the DocxFactory table of contents automatic update. This
    method can be used to update the table of contents with a macro in
    the Normal.dotm.

1 – Sets the update fields flag in the document that tells Word to update
    the table of contents when opening the document. This is the default
    value if no value is set.

2 – Use Word Automation Services to open the document, update the table
    of contents and save the document in the background.
```

Notes:

- The update table of contents method is a global setting that is not reset every time a new document is created.

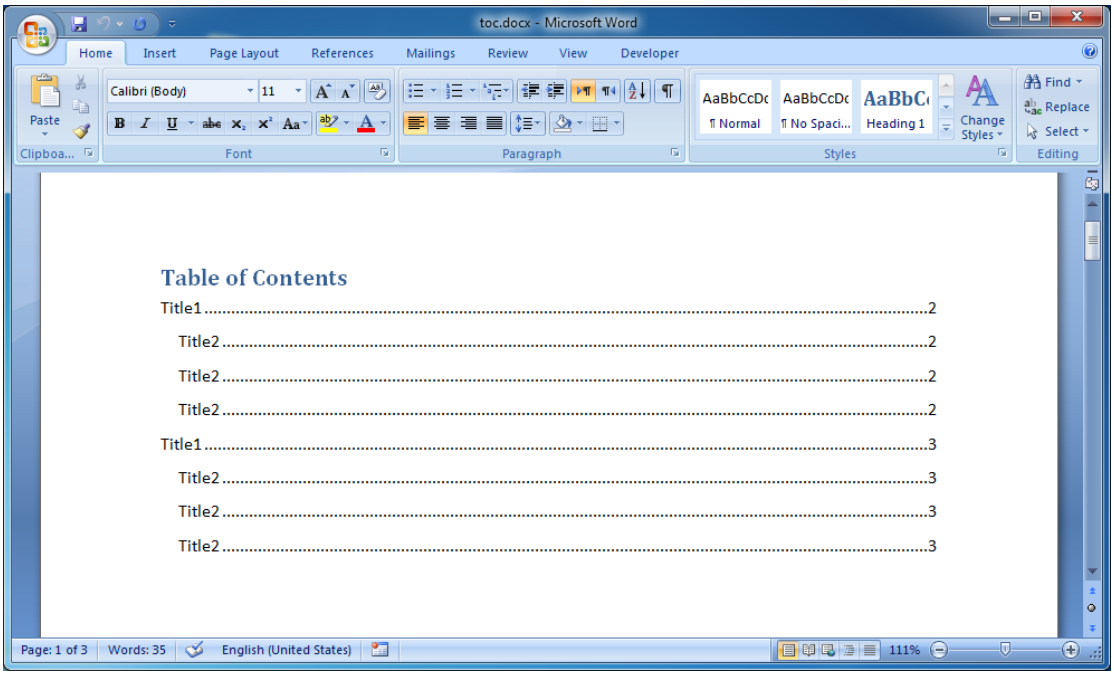
6. Open the created .DOCX file (see picture below).

   As you can see, a popup message opens when opening the document prompting the user to confirm the update.



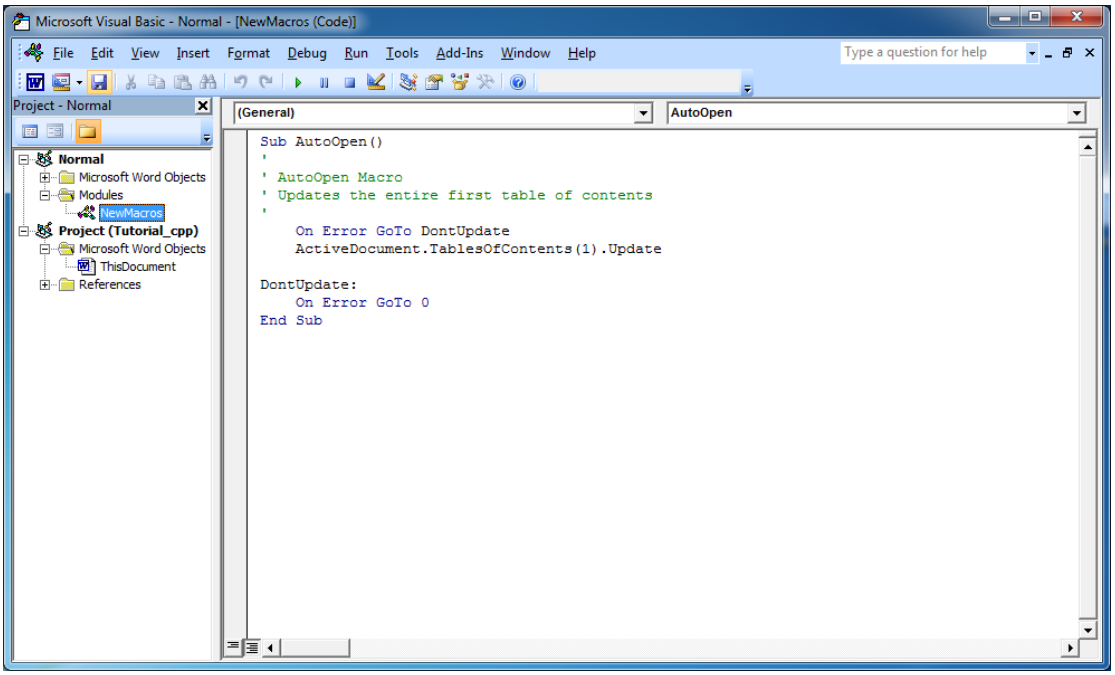
7. Set the update method to 2, run the code and open the created .DOCX file.

   As you can see, the table of contents was automatically updated and there was no popup message when opening the document but creating the document (and also updating the table of contents) took substantially longer this time.



8. Set the update method to 0 and add a macro to the Normal.dotm.

   Open the Visual Basic for Word Editor from the Code group in the Developer ribbon. Select the NewMacros section in the Normal document, copy the AutoOpen macro below and save (see picture below).



Note: Remember that 1. this macro has to be added on every computer that can open created documents and 2. this macro will update the table of contents of every document opened on this computer.

Introduction to Paging

All the exercises so far only created a single page but reports can and often have multiple pages and rules how to divide the report into pages are needed.

For example:

- If a table overflows onto the next page then repeat the column labels, header or title rows when continuing in the next page.
- Some items cannot be split and must stay whole so either the entire item fits in the page or the item is moved to the next page.
- Some items cannot be separated from their previous item so if the item is moved to the next page then the previous items are moved with him.

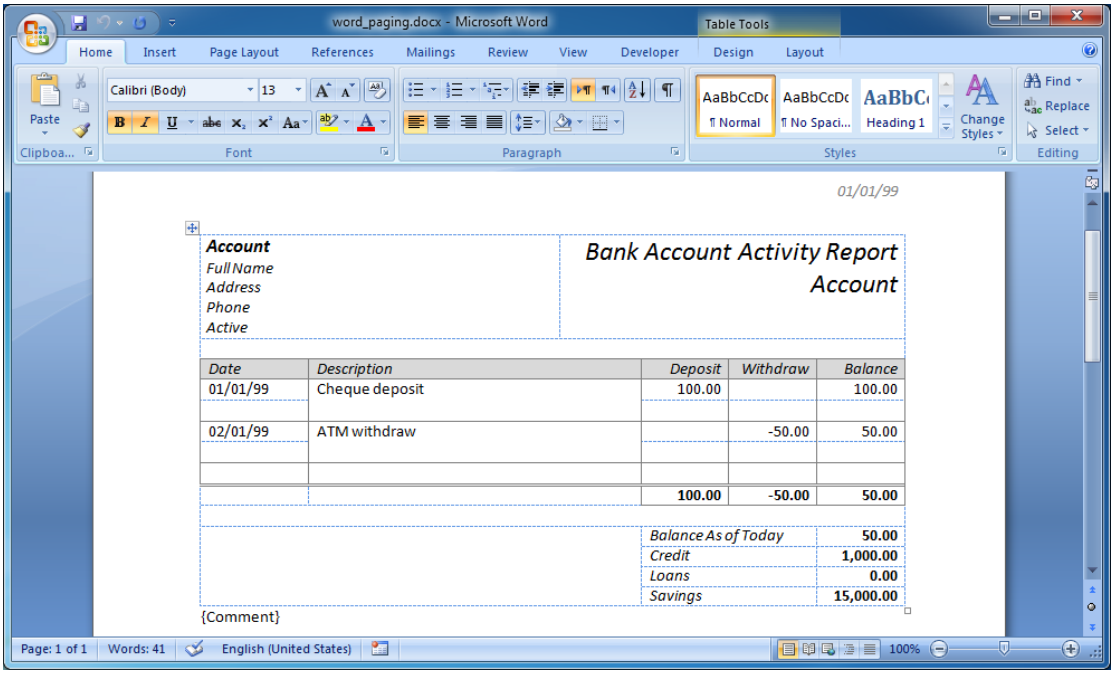
Both Word and DocxFactory have their own paging features for dividing the report into pages each with its own advantages and disadvantages but you can only use one, you cannot use both.

Word Paging Features

The Word paging features are easy to use but Word only has basic paging features.

Word Paging Exercise

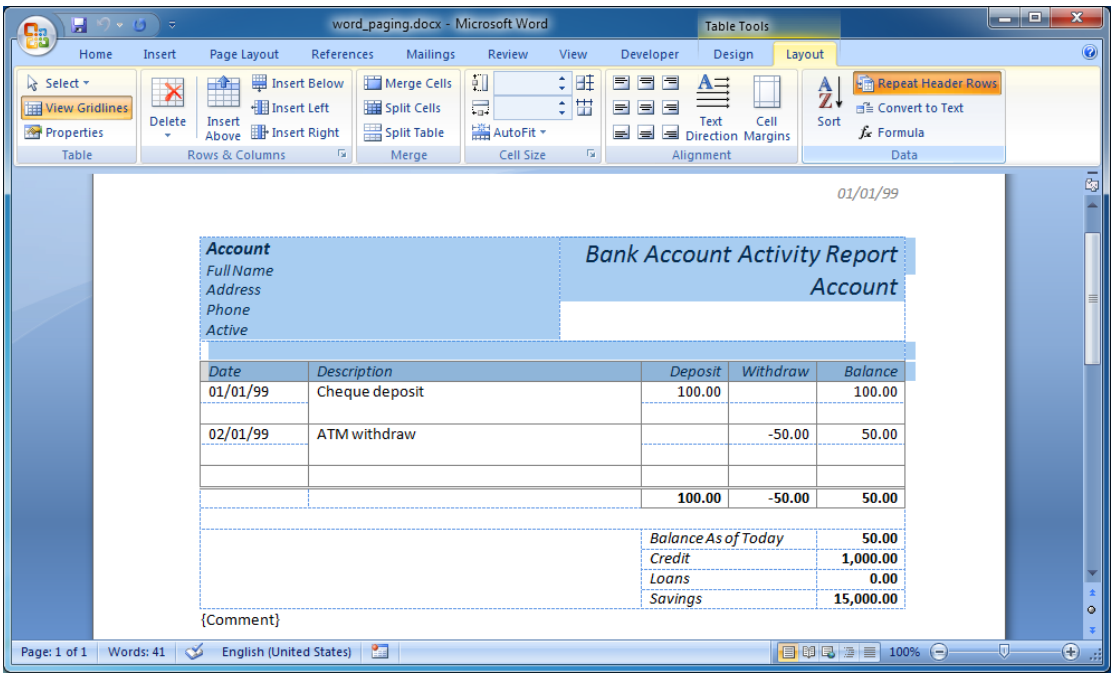
4. Open word_paging.docx in the DocxFactory/exercises/templates/ directory (see picture below).



5. Mark the table rows from the first table row to the column labels (including the column labels) in the Account item as Repeat Header so they will be repeated for every page.

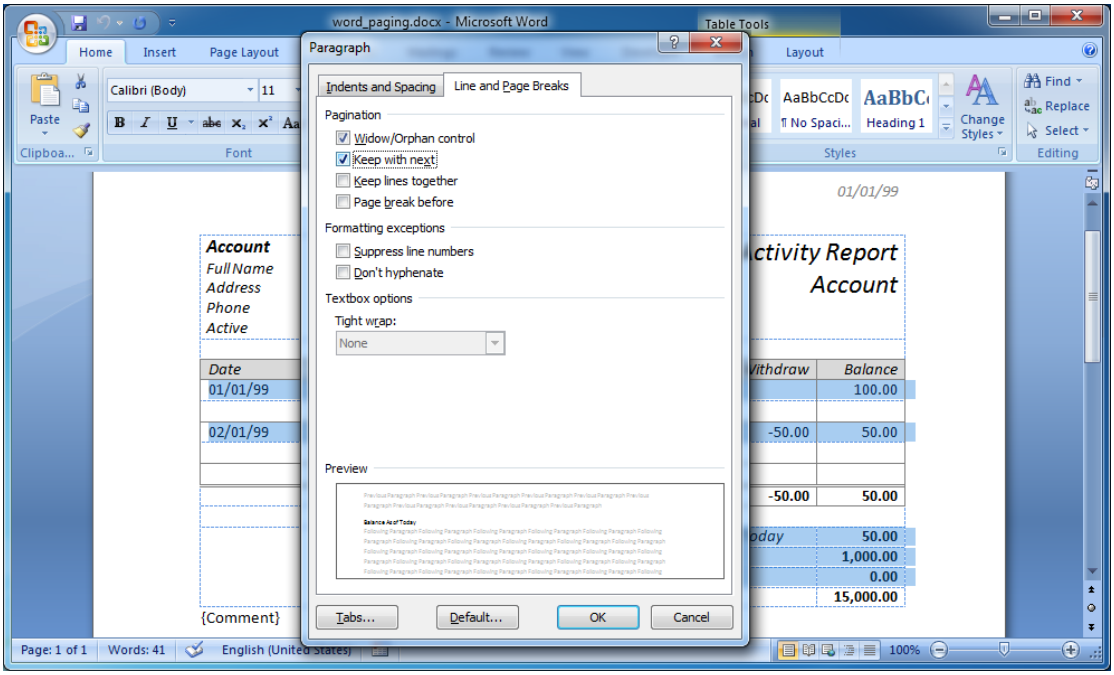
   Because you can only have one level of repeating headers in the Word paging features, you can either mark the report header and the column labels or only the column labels as Repeat Header.

   Highlight the table rows from the first table row to the column labels (including the column labels) in the Account item and select the Repeat Header Rows in the Layout ribbon (see picture below).



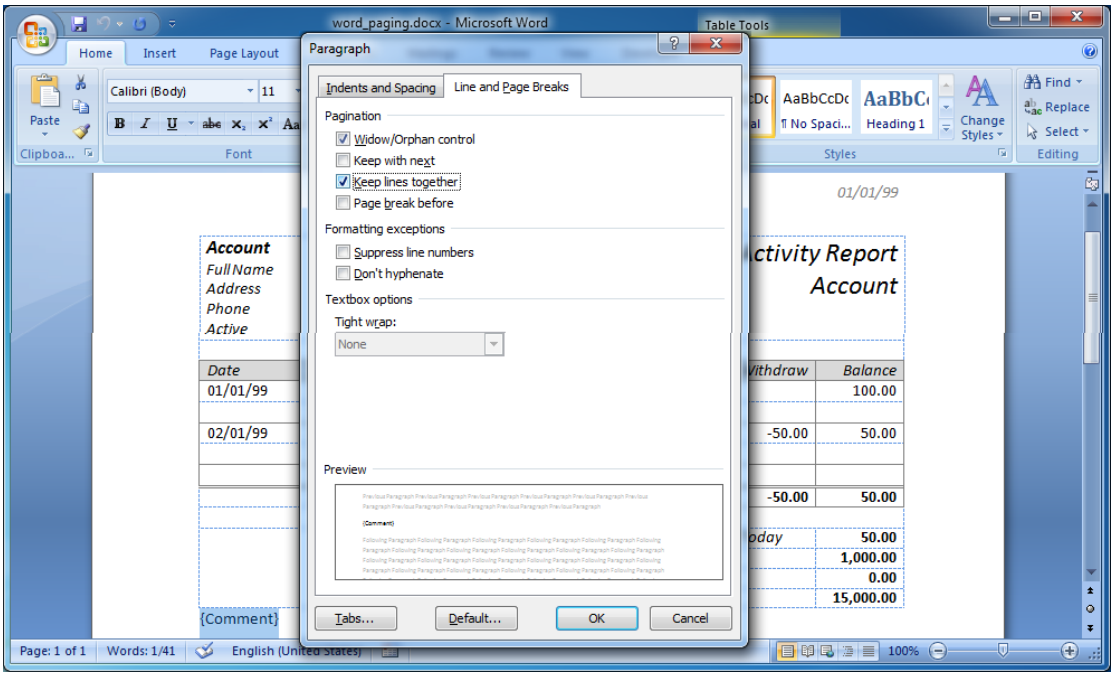
6. Chain the table rows of the Withdraw, Deposit and Summary items using Keep with next so the table rows cannot be split across pages and must stay whole.

   Highlight the first table row of the Withdraw and Deposit line items and the first 3 table rows of the Summary item, open the Paragraph dialog box in the Home ribbon and select Keep with next in the Line and Page Breaks tab (see picture below).



7. Select the Keep lines together for the Comment item so the paragraph lines cannot be split across pages and must stay whole.

   Highlight the comment item, open the Paragraph dialog box in the Home ribbon and select Keep lines together in the Line and Page Breaks tab (see picture below).



The exercise introduces the most common Word paging features.

Repeat Header Rows

If a table overflows onto the next pages then the top table rows marked as repeat header will repeat when continuing in the next pages.

Note: The repeat header rows must start from the first table row. If you want to start the repeat header after the first table row then you will need to split the table where the repeat header starts.

Note: Repeat header does not work for nested tables, which means there can only be one level of repeat header rows for the top level table starting at the first table row, which greatly limits its possible uses.

Keep with Next

Keep with next says that the paragraph cannot be separated from the next paragraph so if the next paragraph does not fit in the page and is moved to the next page then it takes the paragraph with it.

Keep with next is mostly used to chain multiple paragraphs or table rows (using the paragraphs in the table row) to a single unit that cannot be separated so either all of them fit in the page or they are all moved together to the next page.

The paragraphs are chained together by marking all the paragraphs keep with next except for the last paragraph to stop the chaining.

Keep Lines Together

Keep lines together says that the paragraph cannot be split into lines so either the paragraph fits in the page or the entire paragraph is moved to the next page.

Keep lines together only works on a single paragraph and cannot be used to keep multiple paragraphs or table rows together, use keep with next for that.

Note: Keep lines together is completely ignored and has no effect in table rows, which greatly limits its possible uses.

Allow row to break across pages

Another important paging feature especially for rows with a height of several lines is Allow row to break across pages in the Row tab, in the Table Properties dialog box.

If Allow row to break across pages is not selected then the row and its content cannot be split across pages and either the entire row fits in the page or it is moved to the next page.

8. Compile the template.

9. Create the .DOCX file.

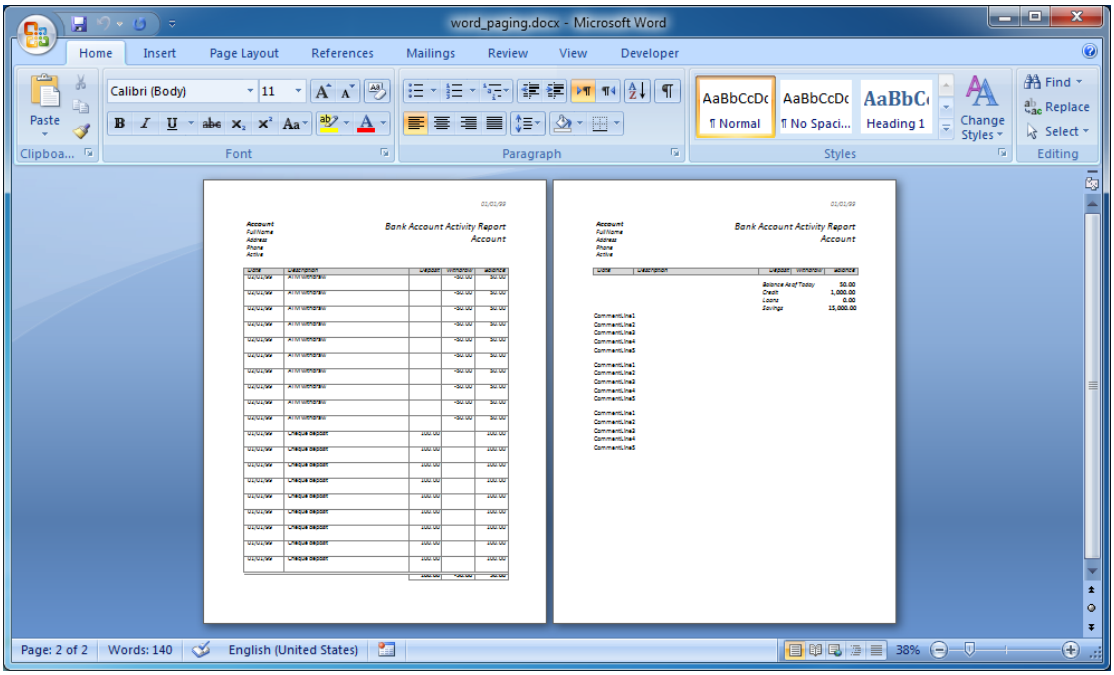
Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/word_paging.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 10; i++)
        {
            l_merger.paste("Withdraw");
        }

        for (int i = 0; i < 9; i++)
        {
            l_merger.paste("Deposit");
        }

        l_merger.paste("Total");

        l_merger.paste("Summary");

        for (int i = 0; i < 3; i++)
        {
            l_merger.setClipboardValue("Comment", "Comment",
                "CommentLine1\n"
                "CommentLine2\n"
                "CommentLine3\n"
                "CommentLine4\n"
                "CommentLine5");

            l_merger.paste("Comment");
        }

        l_merger.save("/tmp/word_paging.docx");
```

```
        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

10. Open the created .DOCX file (see picture below).

    You can change the number of line items, number of comments etc. in the code and recreate the new document to see how the report is divided into pages.



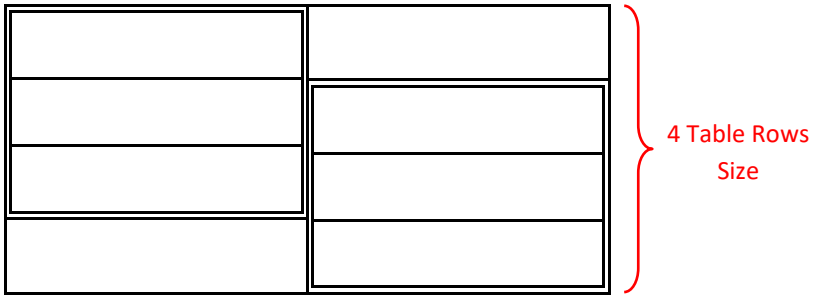
DocxFactory Paging Features

The DocxFactory paging features are more complicated to use but DocxFactory has more advanced paging features. The basic rule of thumb is that you should use the Word paging features whenever you can and use the DocxFactory paging features when you have to.

The difference between Word and DocxFactory paging is that Word renders files to printed documents so it knows the size of everything, paragraph, table row, picture etc. and if the page is filled so it can be divided using the paging rules.

But DocxFactory does not render files because rendering files to printed documents would almost be like rewriting Word so it cannot know the size of everything and if the page is filled. Instead, in DocxFactory sizes are measured in table rows. You set the page size in number of table rows and item sizes are the number of table rows in them.

For table rows with a different height than the rest of the table rows or anything other than table rows like paragraphs you can set their equivalent table row size manually. DocxFactory can also calculate item sizes with nested tables (see picture below).



A limit with measuring sizes in table rows is that if you insert a multiline text in a table row that expands the table row height, to DocxFactory it will still be one table row in height even though it might now have the height of several table rows so table rows height cannot be expanded except by other table rows inside them, as explained above.

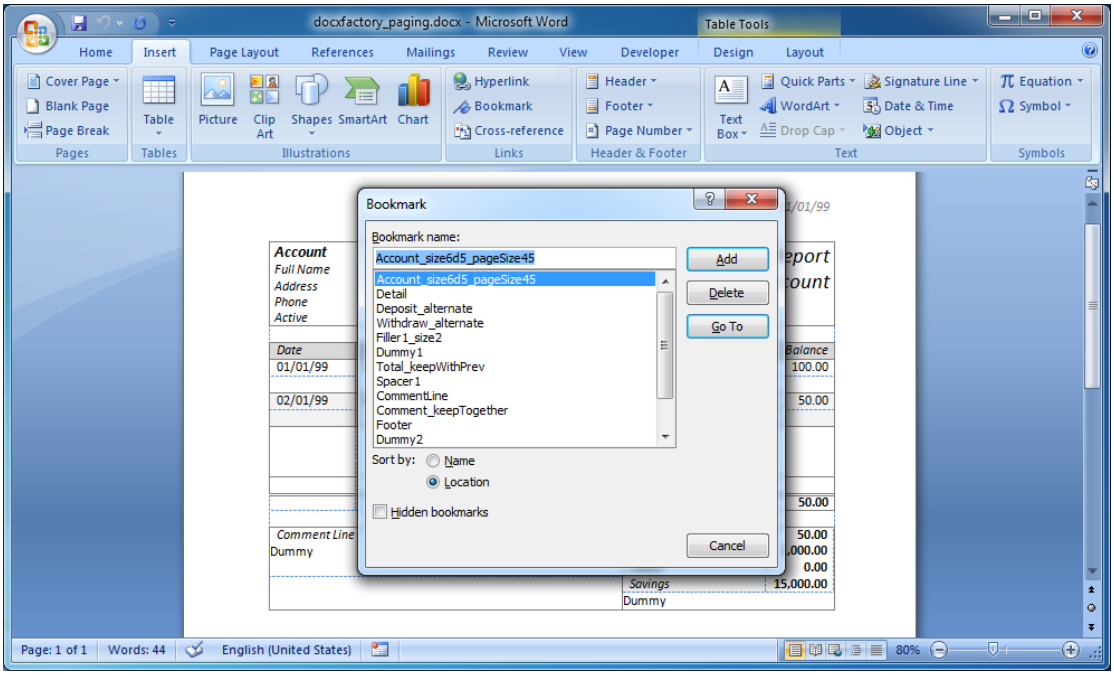
If you plan to insert multiline text values then you have 2 options: 1. Use a static table row height that is big enough to hold all possible values by either setting the table row height or merging several table rows (usually used for small 2-3 line texts). 2. Breaking the multiline text into single lines and inserting every line into its own child table row expanding the table row height dynamically (usually used for big multiline texts).

A table row border also has a size even though it is very small all the table rows borders in a page can add up to the size of another table row. Because DocxFactory measures sizes in table rows, the table rows with borders and table rows without borders should have the same size otherwise pages that use more table rows without borders will be slightly shorter. To make table rows without borders have the same size as table rows with borders add a white border (or the same color as the background) for table rows without borders.

DocxFactory Paging Exercise

9. Open docxfactory_paging.docx in the DocxFactory/exercises/templates/ directory.

   Open the Bookmark dialog box, sort the bookmarks by location and go through and highlight the bookmarks (see picture below).



The template contains the following bookmarks (see details below):

1. **Account_size6d5_pageSize45**

   Because the account item is the top level item for all the items in the report, the account item with its header at the top will be repeated for every page in the report.

   _pageSize45 sets the size of the page the top level item starts to 45 table rows.

   Because account item has table rows with a different size than the rest of the table rows, _size6d5 is used to set the size to 6.5 table rows (excluding child items and the table rows with child items in them).

2. **Detail**

   The detail item contains the column labels for the deposit and withdraw line items.
   The detail item with its column labels will be repeated if a line item does not fit in the page and is moved to the next page.

3. **Deposit_alternate**

   The deposit line item.

   _alternate marks the item as alternating color.

4. **Withdraw_alternate**

   The withdraw line item.

   _alternate marks the item as alternating color.

5. **Filler1_size2**

   Fills the page to its full size and divides the group into top items that always stay at the top and bottom items that always stay at the bottom and cannot be mixed.

   Because the filler item is made up of 3 or more table rows, _size2 says that the page needs to be filled by 2 or more table rows otherwise the page is not filled.

   Note: The filler table rows have a white border instead of having no border so the table rows size will be the same as the rest of the table rows with borders.

6. **Dummy1**

   The dummy item is never pasted and its only purpose is to separate the line and total item borders.

7. **Total_keepWithPrev**

   The lines total item.

   _keepWithPrev says that the item cannot be separated from the previous item. If the total item is moved to the next page then it will take a line item with it from the previous page so it will not be alone on the page.

8. **Spacer1**

   Adds a spacer item between items in the group.

   If the footer and detail items are in the same page then a spacer item is added between them. If the footer item starts in a new page then a spacer will not be added before it.

9. **Footer**

   The footer item contains the comment and summary items.

   The footer is put in a separate item that is only added at the end instead of being a fixed part of the account top level item that will be in every page.

10. **Comment_keepTogether**

    The report breaks multiline comments into single lines and inserts every line into its own table row instead of using a fixed table row height.

    The comment item contains all the comment lines.

    _keepTogether says that it cannot be split and must stay whole. If the entire comment does not fit in the page then the comment is moved to the next page.

    Note: The comment table rows have a white border instead of having no border so the table rows size will be the same as the rest of the table rows with borders.

11. **CommentLine**

    Contains a single comment line.

12. **Spacer2**

    Adds a spacer item between comment items.

13. **Dummy2**

    The dummy part only purpose is to help differentiate between the comment line item table and the table cell it is in.

14. **Summary**

    The report summary item.

    The summary is in a separate item and not a fixed part of the footer item so if the comments overflow onto the next pages then the summary will not be repeated with the footer item in the next pages.

    Note: The summary table rows have a white border instead of having no border so the table rows size will be the same as the rest of the table rows with borders.

15. **Dummy3**

    The dummy part only purpose is to help differentiate between the summary item table and the table cell it is in.

The exercise introduces the following DocxFactory paging features.

_pageSize<n>

_pageSize sets the page size in number of rows and divides the report into pages.

_pageSize is set at the top level item for the page the top level item starts

More specifically, after an item is pasted if the total size of the items in the page is greater than the page size then a new page is started, all the item parents upto the top level item are copied with their field values and the item is moved to the new page.

Note: DocxFactory does not have a repeat header paging feature but when an item is moved to a new page all its parents are copied to the new page that can contain column labels, header information or a title and are repeated for every page their children continue onto.

Note: In case an item is pasted without pasting its parent first and the missing parent is filled in, if the item does not fit in the page and is moved to a new page then the parent is moved with him. You can paste an item without first pasting its parent for parents that are only needed with their children and there is no need for them alone on the page.

For example: a parent that contains the column labels for its line items. The parent and its column labels is only needed with its line items. There is no need for the parent and its column labels alone on the page.

Note: Because _pageSize activates the DocxFactory paging features and is set at the top level item that starts a new page, it can be set for some top level items and not set for others allowing you to use DocxFactory paging features for some top level items and Word paging features for others.

Note: Because the DocxFactory paging features use page breaks to start a new page all the pages in the new document will never go over their page size and the word paging features will never be activated.

Note: You can also enter page sizes with a decimal point by using a "d" instead of "." because "." are not allowed in bookmark names. For example: _pageSize10d5 (for page size 10.5).

_size<n>

_size lets you set the item size manually for items with table rows that have a different height than the rest of the table rows or with anything other than tables rows like paragraphs.

If you are setting the item size for items with child items do not include the child items size and all table rows with child items in them.

Note: You can also enter item sizes with a decimal point by using a "d" instead of "." because "." are not allowed in bookmark names. For example: _size1d5 (for size 1.5).

_keepTogether

_keepTogether says that the item cannot be split and must stay whole so either the item fits in the page or the entire item is moved to the next page.

_keepTogether is not needed for items with no children because DocxFactory only moves items when the page size is exceeded so if the item does not have any children then it is the smallest thing that can be moved and it cannot be split. The problem is with the parent items that can be split across pages.

_keepWithPrev

_keepWithPrev says that the item cannot be separated from the previous item pasted in the group so if the item does not fit in the page and is moved to the next page then the previous item is moved with him. For example: use keep with previous for total lines so they will never be alone on the page.

filler<n>

Items named “filler<n>” are page filler items. The filler item fills the page to its full size. There should only be one filler item for a page.

If the filler item is made up of 3 or more table rows then the filler item is expanded to fill the page by repeating the middle table row, the filler item fills the page in increments of one table row and if _size<n> is added then the page needs to be filled by <n> table rows or more otherwise the page is not filled.

If the filler item is made up of less than 3 table rows then the filler item is repeated to fill the page, the filler item fills the page in increments of the filler item size and _size<n> works in the regular way and sets the filler item size.

The filler item in the template divides the item group into top and bottom items. When the page is filled the filler item is inserted between the top and bottom items keeping the top items at the top and pushing the bottom items to the bottom. If a top item is pasted after a bottom item was pasted then the top item is moved to the next page keeping the top items at the top and the bottom item at the bottom.

spacer<n>

Items named “spacer<n>” are group spacer items. The spacer item is added between the items in the group. There should only be one spacer item for a group.

2. Compile the template.

3. Create the .DOCX file.

Copy and run the code below.

```
#include "WordProcessingMerger.h"

#include <exception>
#include <iostream>
#include <ctime>

using namespace DocxFactory;
using namespace std;



int main()
{
    try
    {
        WordProcessingMerger& l_merger =
            WordProcessingMerger::getInstance();

        time_t l_start = clock();

        l_merger.load(
            "/opt/DocxFactory/exercises/templates/docxfactory_paging.dfw");

        l_merger.paste("Account");

        for (int i = 0; i < 10; i++)
        {
            l_merger.paste("Withdraw");
        }

        for (int i = 0; i < 9; i++)
        {
            l_merger.paste("Deposit");
        }

        l_merger.paste("Total");

        l_merger.paste("Summary");

        for (int i = 0; i < 3; i++)
        {
            l_merger.paste("Comment");

            for (int j = 0; j < 5; j++)
            {
                l_merger.paste("CommentLine");
            }
        }

        l_merger.save("/tmp/docxfactory_paging.docx");
```

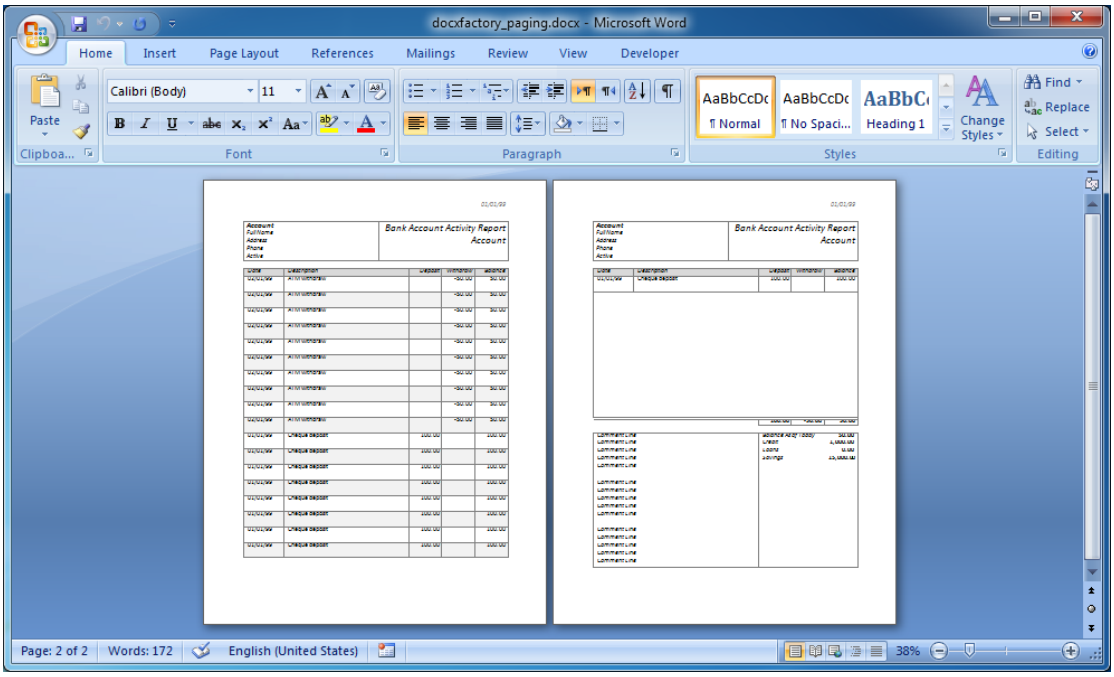
```
        cout<< "Completed (in "
            << (double) (clock() - l_start) / CLOCKS_PER_SEC
            << " seconds)."
            << endl;
    }

    catch (const exception& p_exception)
    {
        cout << p_exception.what() << endl;
    }
}
```

Note: *The detail item and its column labels is only needed with the line items and the footer item containing the comments and the summary item is only needed with its child items, there is no need for them alone on the page.*

For that reason the detail and footer items are not pasted and are filled in when their child items are pasted so if the first child item pasted does not fit in the page and is moved to the new page then they are moved with him so they will not remain alone on the page.

4. Open the created .DOCX file (see picture below).

   You can change the number of line items, number of comments etc. in the code and recreate the new document to see how the report is divided into pages.



Third Party Acknowledgement

DocxFactory only includes third party components that have a permissive free license. According to Wikipedia, "A permissive free software license is a class of free software license with minimal requirements about how the software can be redistributed. Such licenses therefore make no guarantee that future generations of the software will remain free. This is in contrast to licenses which have reciprocity / share-alike requirements. Both sets of free software licenses offer the same freedoms in terms of how the software can be used, studied, and privately modified. A major difference is that when the software is being redistributed (either modified or unmodified), permissive licenses permit the redistributor to restrict access to the modified source code, while copyleft licenses do not allow this restriction.". The third party components included in DocxFactory are standard components widely used in commercial products.

The following documentation notices are required by third party components included in DocxFactory:

DocxFactory includes the ICU version 1.2.0. Such technology is subject to the following terms and conditions:

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that

(a) this copyright and permission notice appear with all copies of the Data Files or Software,

(b) this copyright and permission notice appear in associated documentation, and

(c) there is clear notice in each modified Data File or in the Software as well as in the documentation associated with the Data File(s) or Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

DocxFactory includes the Xerces-C++ XML Parser version 3.1.1. Such technology is subject to the following terms and conditions:

The Apache Software License, Version 1.1. Copyright (c) 1999. The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:
   "This product includes software developed by the
    Apache Software Foundation (http://www.apache.org/)."
   Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

DocxFactory includes ImageMagick version 6.9.1. Such technology is subject to the following terms and conditions:

1)

Copyright (C) 2003 ImageMagick Studio, a non-profit organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall ImageMagick Studio be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the ImageMagick Studio shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the ImageMagick Studio.

2)

In November 2002, the GraphicsMagick Group created GraphicsMagick from ImageMagick Studio's ImageMagick source. ImageMagick adopted some of their improvements to existing programs and scripts, including:

Unix configure
Installation instructions Test programs
PerlMagick test scripts
JP2 image coder
Windows configure
Windows IMDisplay
Windows ATL COM

In addition the Windows distrbution of ImageMagick includes this new program:

Windows ATL7 COM

The improvements and the ATL7 COM is copyright:

3)

From 1991 to August 1999, ImageMagick was developed and distributed by E. I. du Pont de Nemours and Company:

Copyright 1999 E. I. du Pont de Nemours and Company

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall E. I. du Pont de Nemours and Company be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the E. I. du Pont de Nemours and Company shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the E. I. du Pont de Nemours and Company.

4)

This copyright is limited to some code (for locating an installed Ghostscript under Windows) in the file magick/nt_base.c which was incorporated from the gsview package:

Copyright (C) 2000-2002, Ghostgum Software Pty Ltd. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this file ("Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of this Software, and to permit persons to whom this file is furnished to do so, subject to the following conditions:

This Software is distributed with NO WARRANTY OF ANY KIND. No author or distributor accepts any responsibility for the consequences of using it, or for whether it serves any particular purpose or works at all, unless he or she says so in writing.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

5)

The Base64Decode() and Base64Encode() methods are based on source code obtained from OpenSSH. This source code is distributed under the following license.

Copyright (c) 2000 Markus Friedl. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6)

The C++ API known as "Magick++", and which resides in the Magick++ directory, is distributed under the following license:

Copyright 1999 - 2003 Bob Friesenhahn <bfriesen@simple.dallas.tx.us>

Permission is hereby granted, free of charge, to any person obtaining a copy of the source files and associated documentation files ("Magick++"), to deal in Magick++ without restriction, including without limitation of the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of Magick++, and to permit persons to whom the Magick++ is furnished to do so, subject to the following conditions:

This copyright notice shall be included in all copies or substantial portions of Magick++. The copyright to Magick++ is retained by its author and shall not be subsumed or replaced by any other copyright.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall Bob Friesenhahn be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with Magick++ or the use or other dealings in Magick++.

7)

ImageMagick makes use of third-party "delegate" libraries to support certain optional features. These libraries bear their own copyrights and licenses, which may be more or less restrictive than the ImageMagick license. For convenience, when ImageMagick is bundled with (or compiled with) "delegate" libraries, a copy of the licenses for these libraries is provided in a "licenses" directory.

DocxFactory includes the ZINT backend (aka ZINT shared library) version 2.4.3. Such technology is subject to the following terms and conditions:

Copyright (C) 2008 Robin Stuart <robin@zint.org.uk>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.