

The openxlsx2 book

Jan Marvin Garbuszus (JanMarvin) and openxlsx2 authors

Table of contents

Preface	5
1 Introduction	6
1.1 Installation	7
1.2 Working with the package	7
1.3 Example	8
1.4 Authors and contributions	9
1.5 License	9
1.6 A note on speed and memory usage	10
1.7 Invitation to contribute	10
2 basics	11
2.1 Importing data	11
2.1.1 Basic import	11
2.1.2 <code>col_names</code> - first row as column name	12
2.1.3 <code>detect_dates</code> - convert cells to R dates	13
2.1.4 <code>show_formula</code> - show formulas instead of results	13
2.1.5 <code>dims</code> - read specific dimension	14
2.1.6 <code>cols</code> - read selected columns	14
2.1.7 <code>rows</code> - read selected rows	15
2.1.8 <code>convert</code> - convert input to guessed type	15
2.1.9 <code>skip_empty_rows</code> - remove empty rows	16
2.1.10 <code>skip_empty_cols</code> - remove empty columns	16
2.1.11 <code>row_names</code> - keep rownames from input	17
2.1.12 <code>types</code> - convert column to specific type	17
2.1.13 <code>start_row</code> - where to begin	17
2.1.14 <code>na.strings</code> - define missing values	18
2.1.15 Importing as workbook	18
2.2 Exporting data	19
2.2.1 Exporting data frames or vectors	19
2.2.2 Exporting a <code>wbWorkbook</code>	19
2.3 <code>dims</code> / <code>wb_dims()</code>	20

3	styling	22
3.1	Colors, text rotation and number formats	22
3.1.1	the quick way: using high level functions	22
3.1.2	the long way: using bare metal functions	23
4	Working with number formats	26
4.1	numfmts	26
4.2	numfmts2	26
5	Modifying the column widths	28
5.1	wb_set_col_widths	28
6	Adding borders	29
6.1	add borders	29
6.2	styled table	29
6.2.1	the quick way: using high level functions	30
6.2.2	the long way: creating everything from the bone	30
7	Use workbook colors and modify them	32
8	Copy cell styles	34
9	Style strings	36
10	Create custom table styles	38
11	Named styles	42
12	Conditional Formatting	46
12.1	Rule applies to all each cell in range	46
12.2	Highlight row dependent on first cell in row	47
12.3	Highlight column dependent on first cell in column	48
12.4	Highlight entire range cols X rows dependent only on cell A1	49
12.5	Highlight cells in column 1 based on value in column 2	50
12.6	Highlight duplicates using default style	50
12.7	Cells containing text	51
12.8	Cells not containing text	52
12.9	Cells begins with text	52
12.10	Cells ends with text	53
12.11	Colorscale colors cells based on cell value	53
12.12	Between	55
12.13	Top N	56
12.14	Bottom N	57
12.15	Logical Operators	58

12.16(Not) Contains Blanks	58
12.17(Not) Contains Errors	59
12.18Iconset	59
12.19Unique Values	60
13 Databars	61
14 Sparklines	64
15 charts	65
15.1 Add plot to workbook	65
15.2 Add {ggplot2} plot to workbook	65
15.3 Add plot via {rvg}	66
15.4 Add {mschart} plots	67
16 Pivot tables	70
17 Form control	74
18 Upgrade from openxlsx	76
18.1 Basic read and write functions	76
18.1.1 Read xlsx or xlsx files	76
18.2 Write xlsx files	77
18.3 Basic workbook functions	77
18.3.1 Loading a workbook	78
18.3.2 Styles	78
18.3.3 Conditional formatting	80
18.3.4 Data validation	80
18.3.5 Saving	81
References	82

Preface

This is a work in progress book describing the features of [openxlsx2](#) (Barbone and Garbuszus 2023). Having written a book before, I never imagined to do this again and therefore I shall not do it. But still I consider it a nice addition to have something more flexible as our *vignettes*.

This manual was compiled using:

```
R.version
```

```
platform      _  
arch          x86_64-pc-linux-gnu  
arch          x86_64  
os            linux-gnu  
system        x86_64, linux-gnu  
status  
major         4  
minor         3.1  
year          2023  
month         06  
day           16  
svn rev       84548  
language      R  
version.string R version 4.3.1 (2023-06-16)  
nickname      Beagle Scouts
```

and

```
packageVersion("openxlsx2")
```

```
[1] '0.8'
```

Graphics might reflect earlier states and are not constantly updated. If you find any irregularities where our code produces different output than expected, please let us know in the issue tracker at <https://github.com/JanMarvin/openxlsx2/>.

1 Introduction

Unfortunately the entire business world is still built almost entirely on Microsofts Office tools and whenever data is involved, this means that is is largely built on the spreadsheet software Excel. R users that want to interact with this previously closed source file format had to rely on various packages (the following is not necessarily a complete list of all packages). Packages that create workbook objects like `xlsx` (Dragulescu and Arendt 2023) and `openxlsx` (Schauberger and Walker 2023) and packages for special tasks namely `readxl` (Wickham and Bryan 2023), `readxlsb`(Allen 2023), `tidyxl` (Garmonsway 2022), `writexl` (Ooms 2023) and `WriteXLS` (Schwartz 2022), some are Windows exclusive interacting with Excel via a DCOM server `RDCOMClient` and `RExcel` ¹, some are not, `XLconnect`.

In Excel 2007 a new open standard called OOXML(short for office open xml)² which we will refer to as *openxml* was introduced. In December 2006 this standard was accepted by the ECMA and it subsequently replaced the previously used `xls` files wherever people are working with spreadsheet software (after all we are all aware that accounting does not really care whatever file format they are using as long as it opens up in their favorite spreadsheet software). The openxml standard introduced the so called Excel 2007 workbook format `xlsx`. These files are a collection of zipped XML-files. This makes is easy to import the files to R, because all you need is a tool to unzip the files and an XML-parser to import the files as data frames. Still, since there are various tasks available to interact with spreadsheet file, there are also various tools required. If all you want to do is read from files `readxl` is probably enough, if all you want to do is write `xlsx` files `writexl` is probably the fastest choice available. Yet there are a plethora of other tasks available and this book is about them.

The predecessor to `openxlsx2` (Barbone and Garbuszus 2023) called `openxlsx` (originally founded by Andrew Walker) was inspired by the `rJava` based `xlsx` package, but dropped the `rJava` dependency, and the support for the old `xls` files and wrote a custom XML parser in `Rcpp` (Eddelbuettel and François 2011). Later Phillip Schauberger picked up the abandoned `openxlsx` package and continues to maintain it. Finally `openxlsx2` was forked from `openxlsx` to include (1) the `pugixml` (Kapoulkine 2006-2022) library to address shortcomings of the `openxlsx` XML parser and (2) to switch to the `R6` (Chang 2021) package to introduce modern programming flows. Since then `openxlsx2` has evolved a lot, includes many new features and is approaching a stable API release 1.0. This manual is supposed to bundle and extend the existing vignettes and to document the changes.

¹See <https://github.com/omegahat/RDCOMClient>.

²See https://wikipedia.org/wiki/Office_Open_XML.

1.1 Installation

You can install the stable version of `openxlsx2` with:

```
install.packages('openxlsx2')
```

You can install the development version of `openxlsx2` from [GitHub](#) with:

```
# install.packages("remotes")
remotes::install_github("JanMarvin/openxlsx2")
```

Or from [r-universe](#) with:

```
# Enable repository from janmarvin
options(repos = c(
  janmarvin = 'https://janmarvin.r-universe.dev',
  CRAN = 'https://cloud.r-project.org'))
# Download and install openxlsx2 in R
install.packages('openxlsx2')
```

1.2 Working with the package

We offer two different variants how to work with `openxlsx2`.

- The first one is to simply work with R objects. It is possible to read (`read_xlsx()`) and write (`write_xlsx()`) data from and to files. We offer a number of options in the commands to support various features of the openxml format, including reading and writing named ranges and tables. Furthermore, there are several ways to read certain information of an openxml spreadsheet without having opened it in a spreadsheet software before, e.g. to get the contained sheet names or tables.
- As a second variant `openxlsx2` offers the work with so called `wbWorkbook` objects. Here an openxml file is read into a corresponding `wbWorkbook` object (`wb_load()`) or a new one is created (`wb_workbook()`). Afterwards the object can be further modified using various functions. For example, worksheets can be added or removed, the layout of cells or entire worksheets can be changed, and cells can be modified (overwritten or rewritten). Afterwards the `wbWorkbook` objects can be written as openxml files and processed by suitable spreadsheet software.

1.3 Example

This is a basic example which shows you how to solve a common problem:

```
library(openxlsx2)
# read xlsx or xlsxm files
path <- system.file("extdata/openxlsx2_example.xlsx", package = "openxlsx2")
read_xlsx(path)
```

	Var1	Var2	NA	Var3	Var4	Var5	Var6	Var7	Var8
3	TRUE	1	NA	1	a	2023-05-29 3209324	This	#DIV/0!	01:27:15
4	TRUE	NA	NA	#NUM!	b	2023-05-23	<NA>	0	14:02:57
5	TRUE	2	NA	1.34	c	2023-02-01	<NA>	#VALUE!	23:01:02
6	FALSE	2	NA	<NA>	#NUM!	<NA>	<NA>	2	17:24:53
7	FALSE	3	NA	1.56	e	<NA>	<NA>	<NA>	<NA>
8	FALSE	1	NA	1.7	f	2023-03-02	<NA>	2.7	08:45:58
9	NA	NA	NA	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
10	FALSE	2	NA	23	h	2023-12-24	<NA>	25	<NA>
11	FALSE	3	NA	67.3	i	2023-12-25	<NA>	3	<NA>
12	NA	1	NA	123	<NA>	2023-07-31	<NA>	122	<NA>

```
# or import workbooks
wb <- wb_load(path)
wb
```

A Workbook object.

Worksheets:

Sheets: Sheet1 Sheet2

Write order: 1, 2

```
# read a data frame
wb_to_df(wb)
```

	Var1	Var2	NA	Var3	Var4	Var5	Var6	Var7	Var8
3	TRUE	1	NA	1	a	2023-05-29 3209324	This	#DIV/0!	01:27:15
4	TRUE	NA	NA	#NUM!	b	2023-05-23	<NA>	0	14:02:57
5	TRUE	2	NA	1.34	c	2023-02-01	<NA>	#VALUE!	23:01:02
6	FALSE	2	NA	<NA>	#NUM!	<NA>	<NA>	2	17:24:53

7	FALSE	3	NA	1.56	e	<NA>	<NA>	<NA>	<NA>
8	FALSE	1	NA	1.7	f	2023-03-02	<NA>	2.7	08:45:58
9	NA	NA	NA	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
10	FALSE	2	NA	23	h	2023-12-24	<NA>	25	<NA>
11	FALSE	3	NA	67.3	i	2023-12-25	<NA>	3	<NA>
12	NA	1	NA	123	<NA>	2023-07-31	<NA>	122	<NA>

```
# and save
temp <- temp_xlsx()
if (interactive()) wb_save(wb, temp)

## or create one yourself
wb <- wb_workbook()
# add a worksheet
wb$add_worksheet("sheet")
# add some data
wb$add_data("sheet", cars)
# open it in your default spreadsheet software
if (interactive()) wb$open()
```

1.4 Authors and contributions

For a full list of all authors that have made this package possible and for whom we are grateful, please see:

```
system.file("AUTHORS", package = "openxlsx2")
```

If you feel like you should be included on this list, please let us know. If you have something to contribute, you are welcome. If something is not working as expected, open issues or if you have solved an issue, open a pull request. Please be respectful and be aware that we are volunteers doing this for fun in our unpaid free time. We will work on problems when we have time or need.

1.5 License

The `openxlsx2` package is licensed under the MIT license and is based on `openxlsx` (by Alexander Walker and Philipp Schauburger; COPYRIGHT 2014-2022) and `pugixml` (by Arseny Kapoulkine; COPYRIGHT 2006-2022). Both released under the MIT license.

1.6 A note on speed and memory usage

The current state of `openxlsx2` is that it is reasonably fast. That is, it works well with reasonably large input data when reading or writing. It may not work well with data that tests the limits of the `openxml` specification. Things may slow down on the R side of things, and performance and usability will depend on the speed and size of the local operating system's CPU and memory.

Note that there are at least two cases where `openxlsx2` constructs potentially large data frames (i) when loading, `openxlsx2` usually needs to read the entire input file into pugixml and convert it into long data frame(s), and `wb_to_df()` converts one long data frame into two data frames that construct the output object and (ii) when adding data to the workbook, `openxlsx2` reshapes the input data frame into a long data frame and stores it in the workbook, and writes the entire worksheet into a pugixml file that is written when it is complete. Applying cell styles, date conversions etc. will further slow down the process and finally the sheets will be zipped to provide the xlsx output.

Therefore, if you are faced with an unreasonably large dataset, either give yourself enough time, use another package to write the xlsx output (`openxlsx2` was not written with the intention of working with maximum memory efficiency), and by all means use other ways to store data (binary file formats or a database). However, we are always happy to improve, so if you have found a way to improve what we are currently doing, please let us know and open an issue or a pull request.

1.7 Invitation to contribute

We have put a lot of work into `openxlsx2` to make it useful for our needs, improving what we found useful about `openxlsx` and removing what we didn't need. We do not claim to be omniscient about all the things you can do with spreadsheet software, nor do we claim to be omniscient about all the things you can do in `openxlsx2`. Nevertheless, we are quite fond of our little package and invite others to try it out and comment on what they like and of course what they think we are missing or if something doesn't work. `openxlsx2` is a complex piece of software that certainly does not work bug-free, even if we did our best. If you want to contribute to the development of `openxlsx2`, please be our guest on our Github. Join or open a discussion, post or fix issues or write us a mail.

2 basics

Welcome to the basic manual to `openxlsx2`. In this manual you will learn how to use `openxlsx2` to import data from `xlsx`-files to R as well as how to export data from R to `xlsx`, and how to import and modify these `openxml` workbooks in R. This package is based on the work of many contributors to `openxlsx`. It was mostly rewritten using `pugixml` and `R6` making use of modern technology, providing a fresh and easy to use R package.

Over the years many people have worked on the tricky task to handle `xls` and `xlsx` files. Notably `openxlsx`, but there are countless other R-packages as well as third party libraries or calculation software capable of handling such files. Please feel free to use and test your files with other software and or let us know about your experience. Open an issue on github or write us a mail.

2.1 Importing data

Coming from `openxlsx` you might know about `read.xlsx()` (two functions, one for files and one for workbooks) and `readWorkbook()`. Functions that do different things, but mostly the same. In `openxlsx2` we tried our best to reduce the complexity under the hood and for the user as well. In `openxlsx2` they are replaced with `read_xlsx()`, `wb_read()` and they share the same underlying function `wb_to_df()`.

For this example we will use example data provided by the package. You can locate it in our “inst/extdata” folder. The files are included with the package source and you can open them in any calculation software as well.

2.1.1 Basic import

We begin with the `openxlsx2_example.xlsx` file by telling R where to find this file on our system

```
xlsxFile <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
```

The object contains a path to the `xlsx` file and we pass this file to our function to read the workbook into R

```
# import workbook
wb_to_df(xlsxFile)
#>      Var1 Var2 NA  Var3  Var4      Var5      Var6      Var7      Var8
#> 3   TRUE   1 NA    1     a 2023-05-29 3209324 This #DIV/0! 01:27:15
#> 4   TRUE  NA NA  #NUM!    b 2023-05-23      <NA>      0 14:02:57
#> 5   TRUE   2 NA   1.34    c 2023-02-01      <NA> #VALUE! 23:01:02
#> 6  FALSE   2 NA  <NA> #NUM!      <NA>      <NA>      2 17:24:53
#> 7  FALSE   3 NA   1.56    e      <NA>      <NA>      <NA>
#> 8  FALSE   1 NA   1.7    f 2023-03-02      <NA>      2.7 08:45:58
#> 9    NA  NA NA  <NA>  <NA>      <NA>      <NA>      <NA>
#> 10 FALSE   2 NA    23    h 2023-12-24      <NA>      25      <NA>
#> 11 FALSE   3 NA   67.3    i 2023-12-25      <NA>      3      <NA>
#> 12    NA   1 NA   123  <NA> 2023-07-31      <NA>     122      <NA>
```

The output is created as a data frame and contains data types date, logical, numeric and character. The function to import the file to R, `wb_to_df()` provides similar options as the `openxlsx` functions `read.xlsx()` and `readWorkbook()` and a few new functions we will go through the options. As you might have noticed, we return the column of the xlsx file as the row name of the data frame returned. Per default the first sheet in the workbook is imported. If you want to switch this, either provide the `sheet` parameter with the correct index or provide the sheet name.

2.1.2 col_names - first row as column name

In the previous example the first imported row was used as column name for the data frame. This is the default behavior, but not always wanted or expected. Therefore this behavior can be disabled by the user.

```
# do not convert first row to column names
wb_to_df(xlsxFile, col_names = FALSE)
#>      B    C D    E    F      G      H      I      J
#> 2    NA Var2 NA  Var3  Var4      Var5      Var6      Var7      Var8
#> 3   TRUE   1 NA    1     a 2023-05-29 3209324 This #DIV/0! 01:27:15
#> 4   TRUE <NA> NA  #NUM!    b 2023-05-23      <NA>      0 14:02:57
#> 5   TRUE   2 NA   1.34    c 2023-02-01      <NA> #VALUE! 23:01:02
#> 6  FALSE   2 NA  <NA> #NUM!      <NA>      <NA>      2 17:24:53
#> 7  FALSE   3 NA   1.56    e      <NA>      <NA>      <NA>
#> 8  FALSE   1 NA   1.7    f 2023-03-02      <NA>      2.7 08:45:58
#> 9    NA <NA> NA  <NA>  <NA>      <NA>      <NA>      <NA>
#> 10 FALSE   2 NA    23    h 2023-12-24      <NA>      25      <NA>
#> 11 FALSE   3 NA   67.3    i 2023-12-25      <NA>      3      <NA>
```

```
#> 12      NA      1 NA      123 <NA> 2023-07-31      <NA>      122      <NA>
```

2.1.3 detect_dates - convert cells to R dates

The creators of the openxml standard are well known for mistakenly treating something as a date and `openxlsx2` has built in ways to identify a cell as a date and will try to convert the value for you, but unfortunately this is not always a trivial task and might fail. In such a case we provide an option to disable the date conversion entirely. In this case the underlying numerical value will be returned.

```
# do not try to identify dates in the data
wb_to_df(xlsxFile, detect_dates = FALSE)
#>      Var1 Var2 NA  Var3  Var4  Var5      Var6  Var7      Var8
#> 3  TRUE      1 NA      1      a 45075 3209324 This #DIV/0! 0.06059028
#> 4  TRUE      NA NA #NUM!      b 45069      <NA>      0 0.58538194
#> 5  TRUE      2 NA  1.34      c 44958      <NA> #VALUE! 0.95905093
#> 6 FALSE      2 NA  <NA> #NUM!      NA      <NA>      2 0.72561343
#> 7 FALSE      3 NA  1.56      e      NA      <NA>      <NA>      NA
#> 8 FALSE      1 NA  1.7      f 44987      <NA>      2.7 0.36525463
#> 9      NA      NA NA  <NA>  <NA>      NA      <NA>      <NA>      NA
#> 10 FALSE      2 NA      23      h 45284      <NA>      25      NA
#> 11 FALSE      3 NA  67.3      i 45285      <NA>      3      NA
#> 12      NA      1 NA      123 <NA> 45138      <NA>      122      NA
```

2.1.4 show_formula - show formulas instead of results

Sometimes things might feel off. This can be because the openxml files are not updating formula results in the sheets unless they are opened in software that provides such functionality as certain tabular calculation software. Therefore the user might be interested in the underlying functions to see what is going on in the sheet. Using `show_formula` this is possible

```
# return the underlying Excel formula instead of their values
wb_to_df(xlsxFile, show_formula = TRUE)
#>      Var1 Var2 NA  Var3  Var4      Var5      Var6      Var7      Var8
#> 3  TRUE      1 NA      1      a 2023-05-29 3209324 This      E3/0 01:27:15
#> 4  TRUE      NA NA #NUM!      b 2023-05-23      <NA>      C4 14:02:57
#> 5  TRUE      2 NA  1.34      c 2023-02-01      <NA>      #VALUE! 23:01:02
#> 6 FALSE      2 NA  <NA> #NUM!      <NA>      <NA>      C6+E6 17:24:53
#> 7 FALSE      3 NA  1.56      e      <NA>      <NA>      <NA>      <NA>
#> 8 FALSE      1 NA  1.7      f 2023-03-02      <NA>      C8+E8 08:45:58
```

```
#> 9      NA      NA NA <NA> <NA>      <NA>      <NA>      <NA>
#> 10 FALSE      2 NA      23      h 2023-12-24      <NA>      SUM(C10,E10)      <NA>
#> 11 FALSE      3 NA      67.3      i 2023-12-25      <NA>      PRODUCT(C11,E3)      <NA>
#> 12      NA      1 NA      123 <NA> 2023-07-31      <NA>      E12-C12      <NA>
```

2.1.5 dims - read specific dimension

Sometimes the entire worksheet contains too much data, in such case we provide functions to read only a selected dimension range. Such a range consists of either a specific cell like “A1” or a cell range in the notation used in the `openxml` standard

```
# read dimension without column names
wb_to_df(xlsxFile, dims = "A2:C5", col_names = FALSE)
#>   A      B      C
#> 2 NA      NA Var2
#> 3 NA TRUE      1
#> 4 NA TRUE <NA>
#> 5 NA TRUE      2
```

Alternatively, if you don't know the Excel sheet's address, you can use `wb_dims()` to specify the dimension. See below or `in?wb_dims` for more details.

```
# read dimension without column names with `wb_dims()`
wb_to_df(xlsxFile, dims = wb_dims(rows = 2:5, cols = 1:3), col_names = FALSE)
#>   A      B      C
#> 2 NA      NA Var2
#> 3 NA TRUE      1
#> 4 NA TRUE <NA>
#> 5 NA TRUE      2
```

2.1.6 cols - read selected columns

If you do not want to read a specific cell, but a cell range you can use the column attribute. This attribute takes a numeric vector as argument

```
# read selected cols
wb_to_df(xlsxFile, cols = c("A:B", "G"))
#>   NA      Var1      Var5
#> 3 NA      TRUE 2023-05-29
#> 4 NA      TRUE 2023-05-23
```

```
#> 5  NA  TRUE 2023-02-01
#> 6  NA FALSE      <NA>
#> 7  NA FALSE      <NA>
#> 8  NA FALSE 2023-03-02
#> 9  NA  NA      <NA>
#> 10 NA FALSE 2023-12-24
#> 11 NA FALSE 2023-12-25
#> 12 NA  NA 2023-07-31
```

2.1.7 rows - read selected rows

The same goes with rows. You can select them using numeric vectors

```
# read selected rows
wb_to_df(xlsxFile, rows = c(2, 4, 6))
#>   Var1 Var2 NA  Var3  Var4      Var5 Var6 Var7      Var8
#> 4  TRUE  NA NA  #NUM!      b 2023-05-23  NA   0 14:02:57
#> 6 FALSE   2 NA  <NA> #NUM!      <NA>  NA   2 17:24:53
```

2.1.8 convert - convert input to guessed type

In xml exists no difference between value types. All values are per default characters. To provide these as numerics, logicals or dates, `openxlsx2` and every other software dealing with xlsx files has to make assumptions about the cell type. This is especially tricky due to the notion of worksheets. Unlike in a data frame, a worksheet can have a wild mix of all types of data. Even though the conversion process from character to date or numeric is rather solid, sometimes the user might want to see the data without any conversion applied. This might be useful in cases where something unexpected happened or the import created warnings. In such a case you can look at the raw input data. If you want to disable date detection as well, please see the entry above.

```
# convert characters to numerics and date (logical too?)
wb_to_df(xlsxFile, convert = FALSE)
#>   Var1 Var2  NA  Var3  Var4      Var5      Var6      Var7      Var8
#> 3  TRUE   1 <NA>    1    a 2023-05-29 3209324 This #DIV/0! 01:27:15
#> 4  TRUE <NA> <NA> #NUM!    b 2023-05-23      <NA>      0 14:02:57
#> 5  TRUE   2 <NA>  1.34    c 2023-02-01      <NA> #VALUE! 23:01:02
#> 6 FALSE   2 <NA> <NA> #NUM!      <NA>      <NA>      2 17:24:53
#> 7 FALSE   3 <NA>  1.56    e      <NA>      <NA> <NA>      <NA>
#> 8 FALSE   1 <NA>   1.7    f 2023-03-02      <NA>      2.7 08:45:58
```

```
#> 9  <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
#> 10 FALSE 2 <NA> 23 h 2023-12-24 <NA> 25 <NA>
#> 11 FALSE 3 <NA> 67.3 i 2023-12-25 <NA> 3 <NA>
#> 12 <NA> 1 <NA> 123 <NA> 2023-07-31 <NA> 122 <NA>
```

2.1.9 skip_empty_rows - remove empty rows

Even though `openxlsx2` imports everything as requested, sometimes it might be helpful to remove empty lines from the data. These might be either left empty intentional or empty because they were formatted, but the cell value was removed afterwards. This was added mostly for backward comparability, but the default has been changed to `FALSE`. The behavior has changed a bit as well. Previously empty cells were removed prior to the conversion to R data frames, now they are removed after the conversion and are removed only if they are completely empty

```
# erase empty rows from dataset
wb_to_df(xlsxFile, sheet = 1, skip_empty_rows = TRUE) |> tail()
#>   Var1 Var2 NA Var3 Var4 Var5 Var6 Var7 Var8
#> 6 FALSE 2 NA <NA> #NUM! <NA> <NA> 2 17:24:53
#> 7 FALSE 3 NA 1.56 e <NA> <NA> <NA> <NA>
#> 8 FALSE 1 NA 1.7 f 2023-03-02 <NA> 2.7 08:45:58
#> 10 FALSE 2 NA 23 h 2023-12-24 <NA> 25 <NA>
#> 11 FALSE 3 NA 67.3 i 2023-12-25 <NA> 3 <NA>
#> 12 NA 1 NA 123 <NA> 2023-07-31 <NA> 122 <NA>
```

2.1.10 skip_empty_cols - remove empty columns

The same for columns

```
# erase empty columns from dataset
wb_to_df(xlsxFile, skip_empty_cols = TRUE)
#>   Var1 Var2 Var3 Var4 Var5 Var6 Var7 Var8
#> 3 TRUE 1 1 a 2023-05-29 3209324 This #DIV/0! 01:27:15
#> 4 TRUE NA #NUM! b 2023-05-23 <NA> 0 14:02:57
#> 5 TRUE 2 1.34 c 2023-02-01 <NA> #VALUE! 23:01:02
#> 6 FALSE 2 <NA> #NUM! <NA> <NA> 2 17:24:53
#> 7 FALSE 3 1.56 e <NA> <NA> <NA> <NA>
#> 8 FALSE 1 1.7 f 2023-03-02 <NA> 2.7 08:45:58
#> 9 NA NA <NA> <NA> <NA> <NA> <NA> <NA>
#> 10 FALSE 2 23 h 2023-12-24 <NA> 25 <NA>
```



```
#> 11 FALSE      3  67.3      i 2023-12-25      <NA>      3      <NA>
#> 12      NA      1   123 <NA> 2023-07-31      <NA>     122     <NA>
```

2.1.11 row_names - keep rownames from input

Sometimes the data source might provide rownames as well. In such a case you can `openxlsx2` to treat the first column as rowname

```
# convert first row to rownames
wb_to_df(xlsxFile, sheet = 2, dims = "C6:G9", row_names = TRUE)
#>           mpg cyl disp  hp
#> Mazda RX4    21.0   6  160 110
#> Mazda RX4 Wag 21.0   6  160 110
#> Datsun 710    22.8   4  108  93
```

2.1.12 types - convert column to specific type

If the user know better than the software what type to expect in a worksheet, this can be provided via `types`. This parameter takes a named numeric. 0 is character, 1 is numeric and 2 is date

```
# define type of the data.frame
wb_to_df(xlsxFile, cols = c(2, 5), types = c("Var1" = 0, "Var3" = 1))
#>      Var1  Var3
#> 3  TRUE  1.00
#> 4  TRUE   NaN
#> 5  TRUE  1.34
#> 6 FALSE   NA
#> 7 FALSE  1.56
#> 8 FALSE  1.70
#> 9  <NA>   NA
#> 10 FALSE 23.00
#> 11 FALSE 67.30
#> 12 <NA> 123.00
```

2.1.13 start_row - where to begin

Often the creator of the worksheet has used a lot of creativity and the data does not begin in the first row, instead it begins somewhere else. To define the row where to begin reading,

define it via the `start_row` parameter

```
# start in row 5
wb_to_df(xlsxFile, start_row = 5, col_names = FALSE)
#>      B C D      E      F      G H      I      J
#> 5  TRUE 2 NA  1.34      c 2023-02-01 NA #VALUE! 23:01:02
#> 6 FALSE 2 NA      NA #NUM!      <NA> NA      2 17:24:53
#> 7 FALSE 3 NA  1.56      e      <NA> NA      <NA>      <NA>
#> 8 FALSE 1 NA  1.70      f 2023-03-02 NA      2.7 08:45:58
#> 9      NA NA NA      NA <NA>      <NA> NA      <NA>      <NA>
#> 10 FALSE 2 NA  23.00      h 2023-12-24 NA      25      <NA>
#> 11 FALSE 3 NA  67.30      i 2023-12-25 NA      3      <NA>
#> 12      NA 1 NA 123.00 <NA> 2023-07-31 NA      122      <NA>
```

2.1.14 `na.strings` - define missing values

There is the “#N/A” string, but often the user will be faced with custom missing values and other values we are not interested. Such strings can be passed as character vector via `na.strings`

```
# na strings
wb_to_df(xlsxFile, na.strings = "")
#>      Var1 Var2 NA  Var3  Var4      Var5      Var6      Var7      Var8
#> 3  TRUE      1 NA      1      a 2023-05-29 3209324 This #DIV/0! 01:27:15
#> 4  TRUE      NA NA #NUM!      b 2023-05-23      <NA>      0 14:02:57
#> 5  TRUE      2 NA  1.34      c 2023-02-01      <NA> #VALUE! 23:01:02
#> 6 FALSE      2 NA  <NA> #NUM!      <NA>      <NA>      2 17:24:53
#> 7 FALSE      3 NA  1.56      e      <NA>      <NA>      <NA>      <NA>
#> 8 FALSE      1 NA  1.7      f 2023-03-02      <NA>      2.7 08:45:58
#> 9      NA      NA NA  <NA> <NA>      <NA>      <NA>      <NA>      <NA>
#> 10 FALSE      2 NA      23      h 2023-12-24      <NA>      25      <NA>
#> 11 FALSE      3 NA  67.3      i 2023-12-25      <NA>      3      <NA>
#> 12      NA      1 NA  123 <NA> 2023-07-31      <NA>      122      <NA>
```

2.1.15 Importing as workbook

In addition to importing directly from `xlsx` or `xlsm` files, `openxlsx2` provides the `wbWorkbook` class used for importing and modifying entire the `openxml` files in R. This `workbook` class is the heart of `openxlsx2` and probably the reason why you are reading this manual in the first place.

Importing a file into a workbook looks like this:

```
# the file we are going to load
xlsxFile <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
# loading the file into the workbook
wb <- wb_load(file = xlsxFile)
```

The additional options `wb_load()` provides are for internal use: `sheet` loads only a selected sheet from the workbook and `data_only` reads only the data parts from a workbook and ignores any additional graphics or pivot tables. Both functions create workbook objects that can only be used to read data, and we do not recommend end users to use them. Especially not if they intend to re-export the workbook afterwards.

Once a workbook is imported, we provide several functions to interact with and modify it (the `wb_to_df()` function mentioned above works the same way for an imported workbook). It is possible to add new sheets and remove sheets, as well as to add or remove data. R-plots can be inserted and also the style of the workbook can be changed, new fonts, background colors and number formats. There is a wealth of options explained in the man pages and the additional style vignette (more vignettes to follow).

2.2 Exporting data

2.2.1 Exporting data frames or vectors

If you want to export a data frame from R, you can use `write_xlsx()` which will create an xlsx file. This file can be tweaked further. See `?openxlsx2::write_xlsx` to see all the options. (Further explanation and examples will follow).

```
write_xlsx(x = mtcars, file = "mtcars.xlsx")
```

2.2.2 Exporting a wbWorkbook

Imported workbooks can be saved as xlsx or xlsxm files with the wrapper `wb_save()` or with `wb$save()`. Both functions take the filename and an optional `overwrite` option. If the latter is set, an optional guard is provided to check if the file you want to write already exists. But be careful, this is optional. The default is to save the file and replace an existing file. Of course, on Windows, files that are locked (for example, if they were opened by another process) will not be replaced.

```
# replace the existing file
wb$save("mtcars.xlsx")

# do not overwrite the existing file
try(wb$save("mtcars.xlsx", overwrite = FALSE))
```

2.3 dims/ wb_dims()

In `openxlsx2` functions that interact with worksheet cells are using `dims` as argument and require the users to provide these. `dims` are cells or cell ranges in A1 notation. The single argument `dims` hereby replaces `col/row`, `cols/rows` and `xy`. Since A1 notation is rather simple in the first few columns it might get confusing after the 26. Therefore we provide a wrapper to construct it:

```
# various options
wb_dims(from_row = 4)
#> [1] "A4"

wb_dims(rows = 4, cols = 4)
#> [1] "D4"
wb_dims(rows = 4, cols = "D")
#> [1] "D4"

wb_dims(rows = 4:10, cols = 5:9)
#> [1] "E4:I10"

wb_dims(rows = 4:10, cols = "A:D") # same as below
#> [1] "A4:D10"
wb_dims(rows = seq_len(7), cols = seq_len(4), from_row = 4)
#> [1] "A4:D10"
# 10 rows and 15 columns from indice B2.
wb_dims(rows = 1:10, cols = 1:15, from_col = "B", from_row = 2)
#> [1] "B2:P11"

# data + col names
wb_dims(x = mtcars)
#> [1] "A1:K33"
# only data
wb_dims(x = mtcars, select = "data")
#> [1] "A2:K33"
```

```

# The dims of the values of a column in `x`
wb_dims(x = mtcars, cols = "cyl")
#> [1] "B2:B33"
# a column in `x` with the column name
wb_dims(x = mtcars, cols = "cyl", select = "x")
#> [1] "B1:B33"
# rows in `x`
wb_dims(x = mtcars)
#> [1] "A1:K33"

# in a wb chain
wb <- wb_workbook()$
  add_worksheet()$
  add_data(x = mtcars)$
  add_fill(
    dims = wb_dims(x = mtcars, rows = 1:5), # only 1st 5 rows of x data
    color = wb_color("yellow")
  )$
  add_fill(
    dims = wb_dims(x = mtcars, select = "col_names"), # only column names
    color = wb_color("cyan2")
  )

# or if the data's first coord needs to be located in B2.

wb_dims_custom <- function(...) {
  wb_dims(x = mtcars, from_col = "B", from_row = 2, ...)
}

wb <- wb_workbook()$
  add_worksheet()$
  add_data(x = mtcars, dims = wb_dims_custom())$
  add_fill(
    dims = wb_dims_custom(rows = 1:5),
    color = wb_color("yellow")
  )$
  add_fill(
    dims = wb_dims_custom(select = "col_names"),
    color = wb_color("cyan2")
  )

```

3 styling

Welcome to the styling manual for **openxlsx2**. In this manual you will learn how to use **openxlsx2** to style your worksheets. data from xlsx-files to R as well as how to export data from R to xlsx, and how to import and modify these openxml workbooks in R.

3.1 Colors, text rotation and number formats

Below we show you two ways how to create styled tables with **openxlsx2** one using the high level functions to style worksheet areas and one using the bare metal approach of creating the identical table. We show both ways to create styles in **openxlsx2** to show how you could build on our functions or create your very own functions.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC		
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X25	X26	X27	X28			
1	4413231602	43812	4398444	44322	4354608	4483349	447385348	447385348	448404	4424216	44253	1/8	22.02.19	23. Jul 21	09. Aug	Dec 22	6:13 PM	2:58:44 PM	22:52	07:54:08	21:05:19	09:54	43:056	43:981	44:78756	44:19248	58:17	1071955:1530	07:172	44:004	45:006:1206
2	4431625149	43140	4470846	44628	44413154	4429848	4464877339	437404	4447716	44804	7/11	18:10:22	09. Oct 24	16. May	May 21	3:12 PM	11:02:04 PM	19:25	19:45:48	14:07:21	04:38	44:515	44:785	43:713134	44:00511	23:46	1059827:0111	18:254	45:004	44:132:10073	
3	4530928811	44881	4412174	45:053	4473227	4397146	443244638	441604	44053	45331	5/29	07:04:22	28. Aug 23	29. Dec	Apr 22	4:10 PM	2:31:38 AM	20:57	00:25:19	25:04:20	17:58	45:524	43:840	42:88458	43:76528	35:26	1067871:0454	58:473	45:404	44:131:08622	
4	44443131216	44280	4456385	44686	4461154	4360336	444802376	438404	44046	44394	11/13	10:04:21	30. Oct 22	10. Jul	Mar 20	3:57 PM	2:35:37 AM	21:05	15:54:54	13:07:18	17:35	44:889	45:288	44:70206	45:30237	08:41	1040526:0317	04:280	44:904	44:335:28147	
5	451574649	44941	4464570	44263	4450265	4355515	4442509276	437604	44935	45043	27/02	09:11:21	20. Jan 19	17. Aug	Dec 22	8:23 AM	6:01:48 AM	14:02	08:24:12	27:11:21	07:36	45:091	43:786	44:91050	43:65832	38:58	1051738:5690	06:121	43:464	44:743:58379	
6	4535585107	44931	4416521	45:082	4408481	4414935	4565259:08	432604	438801/3	44794	5/73	18:11:21	06. Oct 20	19. Feb	Dec 20	9:08 AM	1:29:22 AM	22:20	07:14:53	23:07:21	00:54	44:643	44:500	44:28536	44:62071	25:19	1048970:1749	34:094	45:264	44:786:13292	
7	4469980849	44900	4425907	44:595	4397241	4363273	4403255:85	443604	4552915	44380	53/57	05:01:22	21. Feb 21	20. Mar	May 25	7:42 AM	2:04:01 PM	00:46	03:49:11	12:07:21	23:02	44:106	44:740	44:72379	44:91377	07:03	1055285:4356	03:449	44:464	44:786:19514	
8	4274189101	44826	4387849	44748	4387961	4462579	4385772116	447604	4499978	43993	17/71	03:05:24	30. Sep 22	06. Oct	Mar 19	5:19 AM	1:14:27 AM	04:30	07:57:39	02:01:22	23:09	44:332	44:770	43:80247	44:54280	46:34	1057421:1101	51:211	44:764	45:504:88066	
9	4400275867	44751	4384916	44:576	4456929	4500950	444449726	442604	44563	44154	30/50	06:05:21	08. Jul 21	07. Aug	Dec 20	4:14 AM	1:07:53 PM	04:53	15:03:35	07:01:23	18:56	44:292	44:277	44:37343	43:95823	56:11	1071748:3616	38:534	44:864	44:648:45184	
10	4416656761	44410	4461246	44:096	4391815	4372970	4461631:68	442604	440341/9	44172	5/61	07:12:21	07. Oct 19	05. Jan	Dec 22	9:13 AM	1:05:29 PM	19:57	06:24:56	28:12:21	02:33	44:184	44:488	43:36646	44:96635	01:13	1079737:4441	26:279	44:464	44:081:16186	
11	451233654	44274	4469276	45:159	4417174	4488120	4468627:05	440604	441251/5	44470	7/41	15:06:23	27. Mar 24	30. Oct	Jun 22	5:09 PM	6:18:58 AM	23:48	16:46:48	27:09:20	21:04	44:835	44:978	45:09956	44:53868	35:59	1056967:1505	47:156	44:464	45:152:10117	
12	446436867	44233	4447342	44:111	4436187	4487082	443196073	441604	44713	45165	36/59	13:04:23	23. Jan 23	22. Oct	Oct 20	11:49 PM	12:36:23 AM	04:29	05:03:52	24:07:20	04:08	43:779	43:636	45:47625	43:85376	43:07	1053780:6617	51:393	45:004	44:265:34961	
13	4466442816	44058	4495586	45:658	4446734	4462873	4385385:40	454604	443421/2	45716	56/85	02:06:23	14. Jan 22	13. Mar	Feb 20	5:40 AM	12:30:48 PM	22:34	22:02:28	23:06:21	20:23	44:925	44:010	45:040:08	43:67363	46:12	1052780:4216	45:283	44:864	44:265:34961	
14	450542891	44329	4558180	45:295	4408217	4388435	4514506:78	444604	446761/2	45302	49/54	19:10:20	13. Jul 23	28. May	Nov 21	10:43 PM	8:55:30 PM	14:20	13:39:22	24:02:22	01:25	44:924	44:622	44:42882	45:60213	34:50	1061133:4041	58:038	45:004	45:26975	
15	4413550817	43742	4437148	44:113	4457464	4437770	4424994:40	44604	444624	44170	8/79	20:09:24	23. Aug 19	22. Mar	Aug 21	9:16 PM	4:32:01 PM	18:28	06:57:18	05:02:20	23:27	43:779	44:197	44:42350	44:06743	46:36	1064472:0758	16:141	45:864	43:118:29993	
16	4545357879	45448	4316241	43:874	4448127	4428848	4396864:04	446604	440101/4	43469	7/55	12:10:21	06. Sep 22	26. Aug	Dec 18	7:10 PM	12:45:25 AM	04:04	05:56:58	25:05:21	17:00	44:759	45:204	43:60258	44:41416	58:53	1073097:4708	57:274	44:864	44:201:55936	
17	4472030702	45114	4500218	44:050	4391017	4475646	4431286:48	451604	440061/5	44228	5/22	07:07:24	18. Nov 20	10. Jan	Sep 21	11:03 PM	1:12:30 PM	12:04	05:11:51	23:02:23	20:00	45:793	44:818	44:88266	44:73936	47:50	1062375:5711	11:440	44:664	44:697:89145	
18	4335232748	43821	4405039	44:587	4446461	4423730	4434545:74	442604	4416527	44493	25/49	17:08:19	19. Sep 22	08. Nov	Dec 21	3:19 AM	1:12:15 AM	13:20	01:18:30	06:07:21	21:42	44:135	44:484	44:32704	44:61618	27:13	1089701:0121	37:142	45:264	43:546:13556	
19	4481312323	44220	4406216	44:807	45:24579	4498623	4500600:38	439804	442741/6	44812	55/56	16:09:21	02. Aug 21	27. Apr	Dec 21	4:29 AM	1:09:50 AM	04:47	10:41:39	13:08:20	09:02	44:913	43:607	44:07951	43:89242	25:25	1065480:0144	34:331	45:964	44:099:13446	
20	448136077	44185	4501319	44:251	4469458	4423613	4449102:92	454604	438121/5	44978	13/64	30:07:23	22. Aug 22	09. Mar	Jan 20	3:19 PM	7:26:13 PM	23:10	06:23:13	04:04:21	20:24	44:010	44:458	44:66033	43:36566	46:40	1082469:1942	49:380	44:264	45:486:14554	
21	4385135784	44877	4428595	43:016	4446688	4502825	4486249:99	442604	443445/8	44823	19/24	04:08:20	16. Sep 21	21. Apr	May 22	9:49 AM	5:44:11 AM	08:34	02:00:27	23:07:23	08:31	45:361	43:788	44:22365	44:79705	04:31	1046039:3659	19:333	45:364	44:707:756	
22	4432330392	44398	4376650	44:419	4420951	4386113	4416489:29	440604	454994/5	43669	33/53	14:07:20	09. Sep 21	17. Aug	Oct 23	9:11 PM	9:31:36 AM	09:02	11:40:49	07:07:22	06:35	44:548	44:387	45:97377	45:07636	55:35	1056147:0036	50:019	44:764	44:089:45802	
23	4384587573	44585	4434462	44:008	4330445	4374461	4455301:71	438604	44388	44915	16/29	02:40:20	23. Mar 23	06. Dec	May 22	9:49 AM	5:44:11 AM	08:34	02:00:27	23:07:23	08:31	45:361	43:788	44:22365	44:79705	04:31	1046039:3659	19:333	45:364	44:707:756	
24	4403946537	44428	4436692	43:518	45:07189	4624278	4426379:93	437604	436892/7	44196	9/56	29:01:20	15. Nov 19	02. Aug	Dec 19	1:15 AM	1:28:03 AM	18:39	10:44:27	21:11:21	04:47	43:634	44:721	43:76823	44:17042	48:00	1099397:2157	35:014	44:464	44:289:11495	
25	4409773021	44420	4444720	44:233	4353534	4421264	4449469:39	441604	44075	44541	1/65	05:01:21	31. Jul 20	14. May	Jan 21	10:01 AM	11:04:56 PM	18:17	22:47:14	20:44:22	16:51	44:416	43:867	45:38061	43:38103	36:16	1064492:3320	25:362	44:764	44:895:43721	
26	4450788521	45104	4465783	44:554	4485467	44609526	4394707:33	448604	447133/7	44485	3/8	07:03:22	27. Dec 23	03. Feb	Mar 22	7:23 PM	6:04:34 PM	03:50	11:56:05	30:10:21	01:21	44:152	44:318	44:43532	44:59464	13:50	1053587:5658	18:035	44:664	44:945:96168	
27	4490897181	44319	4412828	44:125	45:50355	4379730	4371550:51	451604	442351/2	44681	31/49	14:08:18	02. Apr 22	03. Feb	Mar 22	3:10 AM	1:03:54 PM	04:58	20:46:37	06:08:22	23:32	44:335	44:656	45:04059	44:53133	28:27	1072580:2521	42:493	44:364	44:445:89387	
28	4450121208	45286	4480163	44:781	4364264	4477550	4552545:34	448604	441321/6	44457	23/56	31:12:21	24. Jul 18	27. Mar	May 18	7:41 PM	3:55:37 AM	04:23	20:08:12	27:11:20	23:38	44:095	44:009	43:00006	44:79349	39:01	1045156:5940	09:269	44:464	43:790:35885	

Figure 3.1: The example below, with increased column width.

3.1.1 the quick way: using high level functions

```
# add some dummy data
set.seed(123)
mat <- matrix(rnorm(28 * 28, mean = 44444, sd = 555), ncol = 28)
colnames(mat) <- make.names(seq_len(ncol(mat)))
border_col <- wb_color(theme = 1)
border_sty <- "thin"
```

```

# prepare workbook with data and formatted first row
wb <- wb_workbook() %>%
  wb_add_worksheet("test") %>%
  wb_add_data(x = mat) %>%
  wb_add_border(dims = "A1:AB1",
    top_color = border_col, top_border = border_sty,
    bottom_color = border_col, bottom_border = border_sty,
    left_color = border_col, left_border = border_sty,
    right_color = border_col, right_border = border_sty,
    inner_hcolor = border_col, inner_hgrid = border_sty
  ) %>%
  wb_add_fill(dims = "A1:AB1", color = wb_color(hex = "FF334E6F")) %>%
  wb_add_font(dims = "A1:AB1", name = "Arial", bold = TRUE, color = wb_color(hex = "FFFFFF")) %>%
  wb_add_cell_style(dims = "A1:AB1", horizontal = "center", text_rotation = 45)

# create various number formats
x <- c(
  0, 1, 2, 3, 4, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
  37, 38, 39, 40, 45, 46, 47, 48, 49
)

# apply the styles
for (i in seq_along(x)) {
  cell <- sprintf("%s2:%s29", int2col(i), int2col(i))
  wb <- wb %>% wb_add_numfmt(dims = cell, numfmt = x[i])
}

# wb$open()

```

3.1.2 the long way: using bare metal functions

```

# create workbook
wb <- wb_workbook() %>% wb_add_worksheet("test")

# add some dummy data to the worksheet
set.seed(123)
mat <- matrix(rnorm(28 * 28, mean = 44444, sd = 555), ncol = 28)
colnames(mat) <- make.names(seq_len(ncol(mat)))
wb$add_data(x = mat, col_names = TRUE)

```

```

# create a border style and assign it to the workbook
black <- wb_color(hex = "FF000000")
new_border <- create_border(
  bottom = "thin", bottom_color = black,
  top = "thin", top_color = black,
  left = "thin", left_color = black,
  right = "thin", right_color = black
)
wb$styles_mgr$add(new_border, "new_border")

# create a fill style and assign it to the workbook
new_fill <- create_fill(patternType = "solid", fgColor = wb_color(hex = "FF334E6F"))
wb$styles_mgr$add(new_fill, "new_fill")

# create a font style and assign it to the workbook
new_font <- create_font(sz = 20, name = "Arial", b = TRUE, color = wb_color(hex = "FFFFFFF"))
wb$styles_mgr$add(new_font, "new_font")

# create a new cell style, that uses the fill, the font and the border style
new_cellxfs <- create_cell_style(
  num_fmt_id = 0,
  horizontal = "center",
  text_rotation = 45,
  fill_id = wb$styles_mgr$get_fill_id("new_fill"),
  font_id = wb$styles_mgr$get_font_id("new_font"),
  border_id = wb$styles_mgr$get_border_id("new_border")
)
# assign this style to the workbook
wb$styles_mgr$add(new_cellxfs, "new_styles")

# assign the new cell style to the header row of our data set
cell <- sprintf("A1:%s1", int2col(nrow(mat)))
wb <- wb %>% wb_set_cell_style(
  dims = cell,
  style = wb$styles_mgr$get_xf_id("new_styles")
)

## style the cells with some builtin format codes (no new numFmt entry is needed)
# add builtin style ids
x <- c(

```



```

1, 2, 3, 4, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
37, 38, 39, 40, 45, 46, 47, 48, 49
)

# create styles
new_cellxfs <- create_cell_style(num_fmt_id = x, horizontal = "center")

# assign the styles to the workbook
for (i in seq_along(x)) {
  wb$styles_mgr$add(new_cellxfs[i], paste0("new_style", i))
}

# new styles are 1:28
new_styles <- wb$styles_mgr$get_xf()
for (i in as.integer(new_styles$id[new_styles$name %in% paste0("new_style", seq_along(x))])
  cell <- sprintf("%s2:%s29", int2col(i), int2col(i))
  wb <- wb %>% wb_set_cell_style(dims = cell, style = i)
}

# assign a custom tabColor
wb$worksheets[[1]]$sheetPr <- xml_node_create(
  "sheetPr",
  xml_children = xml_node_create(
    "tabColor",
    xml_attributes = wb_color(hex = "FF00FF00")
  )
)

# # look at the beauty you've created
# wb_open(wb)

```

4 Working with number formats

Per default `openxlsx2` will pick up number formats for selected R classes.

4.1 numfmts

```
## Create Workbook object and add worksheets
wb <- wb_workbook()
wb$add_worksheet("S1")
wb$add_worksheet("S2")

df <- data.frame(
  "Date" = Sys.Date() - 0:19,
  "T" = TRUE, "F" = FALSE,
  "Time" = Sys.time() - 0:19 * 60 * 60,
  "Cash" = paste("$", 1:20), "Cash2" = 31:50,
  "hLink" = "https://CRAN.R-project.org/",
  "Percentage" = seq(0, 1, length.out = 20),
  "TinyNumbers" = runif(20) / 1E9, stringsAsFactors = FALSE
)

## openxlsx will apply default Excel styling for these classes
class(df$Cash) <- c(class(df$Cash), "currency")
class(df$Cash2) <- c(class(df$Cash2), "accounting")
class(df$hLink) <- "hyperlink"
class(df$Percentage) <- c(class(df$Percentage), "percentage")
class(df$TinyNumbers) <- c(class(df$TinyNumbers), "scientific")

wb$add_data("S1", x = df, start_row = 4, row_names = FALSE)
wb$add_data_table("S2", x = df, start_row = 4, row_names = FALSE)
```

4.2 numfmts2

In addition, you can set the style to be picked up using `openxlsx2` options.

```

wb <- wb_workbook()
wb <- wb_add_worksheet(wb, "test")

options("openxlsx2.dateFormat" = "yyyy")
options("openxlsx2.datetimeFormat" = "yyyy-mm-dd")
options("openxlsx2.numFmt" = "€ #.0")

df <- data.frame(
  "Date" = Sys.Date() - 0:19,
  "T" = TRUE, "F" = FALSE,
  "Time" = Sys.time() - 0:19 * 60 * 60,
  "Cash" = paste("$", 1:20), "Cash2" = 31:50,
  "hLink" = "https://CRAN.R-project.org/",
  "Percentage" = seq(0, 1, length.out = 20),
  "TinyNumbers" = runif(20) / 1E9, stringsAsFactors = FALSE,
  "numeric" = 1
)

## openxlsx will apply default Excel styling for these classes
class(df$Cash) <- c(class(df$Cash), "currency")
class(df$Cash2) <- c(class(df$Cash2), "accounting")
class(df$hLink) <- "hyperlink"
class(df$Percentage) <- c(class(df$Percentage), "percentage")
class(df$TinyNumbers) <- c(class(df$TinyNumbers), "scientific")

wb$add_data("test", df)

```

5 Modifying the column widths

5.1 `wb_set_col_widths`

```
wb <- wb_workbook() %>%  
  wb_add_worksheet() %>%  
  wb_add_data(x = mtcars, row_names = TRUE)  
  
cols <- 1:12  
wb <- wb %>% wb_set_col_widths(cols = cols, widths = "auto")
```

6 Adding borders

6.1 add borders

```
wb <- wb_workbook()
# full inner grid
wb$add_worksheet("S1", grid_lines = FALSE)$add_data(x = mtcars)
wb$add_border(
  dims = "A2:K33",
  inner_hgrid = "thin", inner_hcolor = wb_color(hex = "FF808080"),
  inner_vgrid = "thin", inner_vcolor = wb_color(hex = "FF808080")
)
# only horizontal grid
wb$add_worksheet("S2", grid_lines = FALSE)$add_data(x = mtcars)
wb$add_border(dims = "A2:K33", inner_hgrid = "thin", inner_hcolor = wb_color(hex = "FF808080"))
# only vertical grid
wb$add_worksheet("S3", grid_lines = FALSE)$add_data(x = mtcars)
wb$add_border(dims = "A2:K33", inner_vgrid = "thin", inner_vcolor = wb_color(hex = "FF808080"))
# no inner grid
wb$add_worksheet("S4", grid_lines = FALSE)$add_data(x = mtcars)
wb$add_border("S4", dims = "A2:K33")
```

6.2 styled table

Below we show you two ways how to create styled tables with `openxlsx2` one using the high level functions to style worksheet areas and one using the bare metal approach of creating the identical table.

X1	X2
1	3
2	4

6.2.1 the quick way: using high level functions

```
# add some dummy data to the worksheet
mat <- matrix(1:4, ncol = 2, nrow = 2)
colnames(mat) <- make.names(seq_len(ncol(mat)))

wb <- wb_workbook() %>%
  wb_add_worksheet("test") %>%
  wb_add_data(x = mat, col_names = TRUE, start_col = 2, start_row = 2) %>%
  # center first row
  wb_add_cell_style(dims = "B2:C2", horizontal = "center") %>%
  # add border for first row
  wb_add_border(
    dims = "B2:C2",
    bottom_color = wb_color(theme = 1), bottom_border = "thin",
    top_color = wb_color(theme = 1), top_border = "double",
    left_border = NULL, right_border = NULL
  ) %>%
  # add border for last row
  wb_add_border(
    dims = "B4:C4",
    bottom_color = wb_color(theme = 1), bottom_border = "double",
    top_border = NULL, left_border = NULL, right_border = NULL
  )
```

6.2.2 the long way: creating everything from the bone

```
# add some dummy data to the worksheet
mat <- matrix(1:4, ncol = 2, nrow = 2)
colnames(mat) <- make.names(seq_len(ncol(mat)))

wb <- wb_workbook() %>%
  wb_add_worksheet("test") %>%
  wb_add_data(x = mat, start_col = 2, start_row = 2)

# create a border style and assign it to the workbook
black <- wb_color(hex = "FF000000")
top_border <- create_border(
  top = "double", top_color = black,
  bottom = "thin", bottom_color = black
```

```

)

bottom_border <- create_border(bottom = "double", bottom_color = black)

wb$styles_mgr$add(top_border, "top_border")
wb$styles_mgr$add(bottom_border, "bottom_border")

# create a new cell style, that uses the fill, the font and the border style
top_cellxfs <- create_cell_style(
  numFmtId = 0,
  horizontal = "center",
  borderId = wb$styles_mgr$get_border_id("top_border")
)
bottom_cellxfs <- create_cell_style(
  numFmtId = 0,
  borderId = wb$styles_mgr$get_border_id("bottom_border")
)

# assign this style to the workbook
wb$styles_mgr$add(top_cellxfs, "top_styles")
wb$styles_mgr$add(bottom_cellxfs, "bottom_styles")

# assign the new cell style to the header row of our data set
cell <- "B2:C2"
wb <- wb %>% wb_set_cell_style(dims = cell, style = wb$styles_mgr$get_xf_id("top_styles"))
cell <- "B4:C4"
wb <- wb %>% wb_set_cell_style(dims = cell, style = wb$styles_mgr$get_xf_id("bottom_styles"))

```

7 Use workbook colors and modify them

The loop below will apply the tint attribute to the fill color

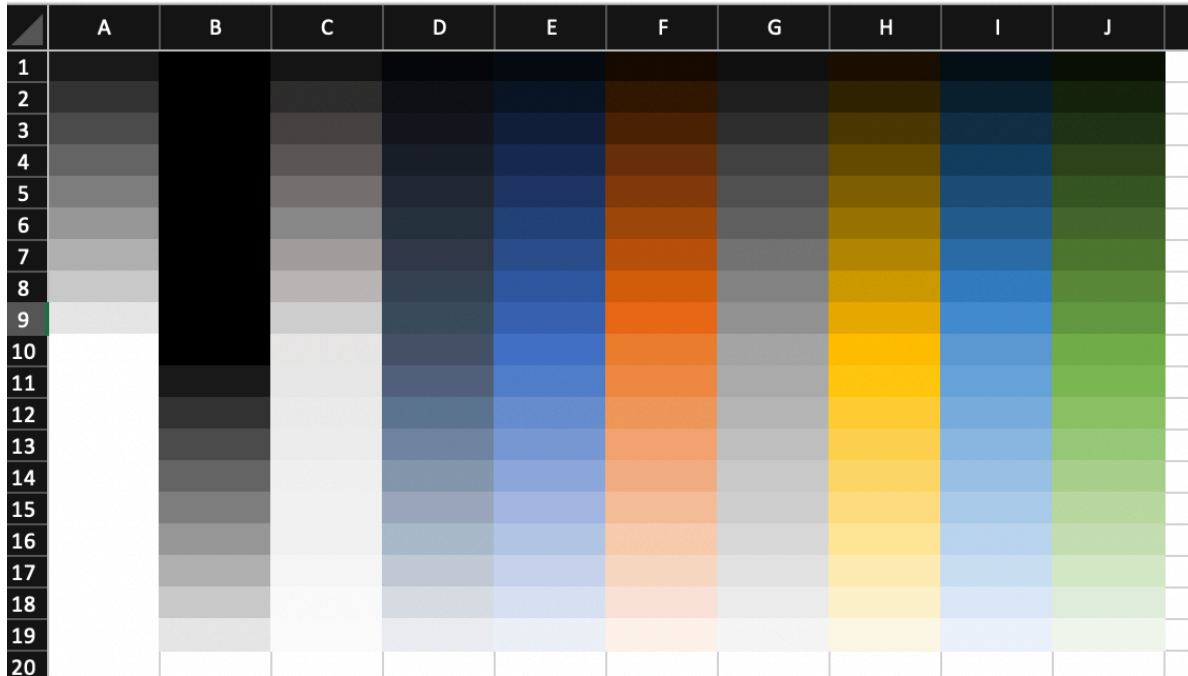


Figure 7.1: Tint variations of the theme colors.

```
wb <- wb_workbook() %>% wb_add_worksheet("S1")

tints <- seq(-0.9, 0.9, by = 0.1)

for (i in 0:9) {
  for (tnt in tints) {
    col <- paste0(int2col(i + 1), which(tints %in% tnt))

    if (tnt == 0) {
      wb <- wb %>% wb_add_fill(dims = col, color = wb_color(theme = i))
    } else {
```



```
    wb <- wb %>% wb_add_fill(dims = col, color = wb_color(theme = i, tint = tint))  
  }  
}  
}
```

8 Copy cell styles

It is possible to copy the styles of several cells at once. In the following example, the styles of some cells from a formatted workbook are applied to a previously empty cell range. Be careful though, `wb_get_cell_style()` returns only some styles, so you have to make sure that the copy-from and copy-to dimensions match in a meaningful way.

```
wb <- wb_load(system.file("extdata", "xlsx2_sheet.xlsx", package = "openxlsx2")) %>%  
  wb_set_cell_style(1, "A30:G35", wb_get_cell_style(., 1, "A10:G15"))  
# wb_open(wb)
```

	A	B	C	D	E	F	H	I
1								
2		Header						
3		Date	Value1		Value2		Value3	
4			€	%	€	%	€	%
6		Jan-21	1,000		431		29	
7		Feb-21	264	26 %	777	180.28 %	28	96.55 %
8		Mar-21	4	1 %	4567	587.77 %	27	96.43 %
9		Apr-21	4,393	120492 %	464	10.16 %	26	96.30 %
10		May-21	53	1 %	433	93.32 %	25	96.15 %
11		Jun-21	63	119 %	356	82.22 %	24	96.00 %
12		Jul-21	838	1324 %	354	99.44 %	23	95.83 %
13		Aug-21	23,131	2760 %	3355	947.74 %	22	95.65 %
14		Sep-21	2,323	10 %	334	9.96 %	21	95.45 %
15		Oct-21	3,323	143 %	541	161.98 %	20	95.24 %
16		Nov-21	35	1 %	555	102.59 %	20	100.00 %
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								
33								
34								
35								
36								
37								

9 Style strings

Using `fmt_txt()` is possible to style strings independently of the cell containing the string.

```
txt <-  
  fmt_txt("Embracing the full potential of ") +  
  fmt_txt("openxlsx2", bold = TRUE, size = 16) +  
  fmt_txt(" with ") +  
  fmt_txt("fmt_txt()", font = "Courier") +  
  fmt_txt(" !")  
  
wb <- wb_workbook()$add_worksheet()$add_data(x = txt, col_names = FALSE)
```

	A	B	C	D	E	F
1	Embracing the full potential of openxlsx2 with <code>fmt_txt()</code> !					
2						

As shown above it is possible to combine multiple styles together into a longer string. It is even possible to use `fmt_txt()` as `na.strings`:

```
df <- mtcars  
df[df < 4] <- NA  
  
na_red <- fmt_txt("N/A", color = wb_color("red"), italic = TRUE, bold = TRUE)  
  
wb <- wb_workbook()$add_worksheet()$add_data(x = df, na.strings = na_red)
```

D	E	F	G	H	I	J	K	
p	drat	wt	qsec	vs	am	gear	carb	
110	N/A	N/A	16.46	N/A	N/A	4	4	
110	N/A	N/A	17.02	N/A	N/A	4	4	
93	N/A	N/A	18.61	N/A	N/A	4	N/A	
110	N/A	N/A	19.44	N/A	N/A	N/A	N/A	
175	N/A	N/A	17.02	N/A	N/A	N/A	N/A	
105	N/A	N/A	20.22	N/A	N/A	N/A	N/A	
245	N/A	N/A	15.84	N/A	N/A	N/A		4
62	N/A	N/A	20	N/A	N/A	4	N/A	
95	N/A	N/A	22.9	N/A	N/A	4	N/A	
123	N/A	N/A	18.3	N/A	N/A	4		4
123	N/A	N/A	18.9	N/A	N/A	4		4
180	N/A	4.07	17.4	N/A	N/A	N/A	N/A	
180	N/A	N/A	17.6	N/A	N/A	N/A	N/A	

10 Create custom table styles

With `create_tablestyle()` it is possible to create your own table styles. This function uses `create_dxfs_style()` (just like your spreadsheet software does). Therefore, it is not quite as user-friendly. The following example shows how the function creates a red table style. The various dxfs styles must be created and assigned to the workbook (similar styles are used in conditional formatting). In `create_tablestyle()` these styles are assigned to the table style elements. Once the table style is created, it must also be assigned to the workbook. After that you can use it in the workbook like any other table style.

```
# a red table style
dx0 <- create_dxfs_style(
  border = TRUE,
  left_color = wb_color("red"),
  right_color = NULL, right_style = NULL,
  top_color = NULL, top_style = NULL,
  bottom_color = NULL, bottom_style = NULL
)

dx1 <- create_dxfs_style(
  border = TRUE,
  left_color = wb_color("red"),
  right_color = NULL, right_style = NULL,
  top_color = NULL, top_style = NULL,
  bottom_color = NULL, bottom_style = NULL
)

dx2 <- create_dxfs_style(
  border = TRUE,
  top_color = wb_color("red"),
  left_color = NULL, left_style = NULL,
  right_color = NULL, right_style = NULL,
  bottom_color = NULL, bottom_style = NULL
)

dx3 <- create_dxfs_style(
```

```

border = TRUE,
top_color = wb_color("red"),
left_color = NULL, left_style = NULL,
right_color = NULL, right_style = NULL,
bottom_color = NULL, bottom_style = NULL
)

dx4 <- create_dxfs_style(
  text_bold = TRUE
)

dx5 <- create_dxfs_style(
  text_bold = TRUE
)

dx6 <- create_dxfs_style(
  font_color = wb_color("red"),
  text_bold = TRUE,
  border = TRUE,
  top_style = "double",
  left_color = NULL, left_style = NULL,
  right_color = NULL, right_style = NULL,
  bottom_color = NULL, bottom_style = NULL
)

dx7 <- create_dxfs_style(
  font_color = wb_color("white"),
  text_bold = TRUE,
  bgFill = wb_color("red"),
  fgColor = wb_color("red")
)

dx8 <- create_dxfs_style(
  border = TRUE,
  left_color = wb_color("red"),
  top_color = wb_color("red"),
  right_color = wb_color("red"),
  bottom_color = wb_color("red")
)

```

```

wb <- wb_workbook() %>%
  wb_add_worksheet(grid_lines = FALSE)

wb$add_style(dx0)
wb$add_style(dx1)
wb$add_style(dx2)
wb$add_style(dx3)
wb$add_style(dx4)
wb$add_style(dx5)
wb$add_style(dx6)
wb$add_style(dx7)
wb$add_style(dx8)

# finally create the table
xml <- create_tablestyle(
  name = "red_table",
  whole_table = wb$styles_mgr$get_dxf_id("dx8"),
  header_row = wb$styles_mgr$get_dxf_id("dx7"),
  total_row = wb$styles_mgr$get_dxf_id("dx6"),
  first_column = wb$styles_mgr$get_dxf_id("dx5"),
  last_column = wb$styles_mgr$get_dxf_id("dx4"),
  first_row_stripe = wb$styles_mgr$get_dxf_id("dx3"),
  second_row_stripe = wb$styles_mgr$get_dxf_id("dx2"),
  first_column_stripe = wb$styles_mgr$get_dxf_id("dx1"),
  second_column_stripe = wb$styles_mgr$get_dxf_id("dx0")
)

wb$add_style(xml)

# create a table and apply the custom style
wb <- wb %>%
  wb_add_data_table(x = mtcars, table_style = "red_table")

```


	A	B	C	D	
1	mpg	cyl	disp	hp	drat
2	21	6	160	110	
3	21	6	160	110	
4	22.8	4	108	93	
5	21.4	6	258	110	
6	18.7	8	360	175	
7	18.1	6	225	105	
8	14.3	8	360	175	

11 Named styles

```
wb <- wb_workbook()$add_worksheet()

name <- "Normal"
dims <- "A1"
wb$add_data(dims = dims, x = name)

name <- "Bad"
dims <- "B1"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Good"
dims <- "C1"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Neutral"
dims <- "D1"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Calculation"
dims <- "A2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Check Cell"
dims <- "B2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Explanatory Text"
dims <- "C2"
```

```

wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Input"
dims <- "D2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Linked Cell"
dims <- "E2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Note"
dims <- "F2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Output"
dims <- "G2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Warning Text"
dims <- "H2"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Heading 1"
dims <- "A3"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Heading 2"
dims <- "B3"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Heading 3"
dims <- "C3"
wb$add_named_style(dims = dims, name = name)

```

```

wb$add_data(dims = dims, x = name)

name <- "Heading 4"
dims <- "D3"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Title"
dims <- "E3"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Total"
dims <- "F3"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

for (i in seq_len(6)) {

  name <- paste0("20% - Accent", i)
  dims <- paste0(int2col(i), "4")
  wb$add_named_style(dims = dims, name = name)
  wb$add_data(dims = dims, x = name)

  name <- paste0("40% - Accent", i)
  dims <- paste0(int2col(i), "5")
  wb$add_named_style(dims = dims, name = name)
  wb$add_data(dims = dims, x = name)

  name <- paste0("60% - Accent", i)
  dims <- paste0(int2col(i), "6")
  wb$add_named_style(dims = dims, name = name)
  wb$add_data(dims = dims, x = name)

  name <- paste0("Accent", i)
  dims <- paste0(int2col(i), "7")
  wb$add_named_style(dims = dims, name = name)
  wb$add_data(dims = dims, x = name)

}

```

```

name <- "Comma"
dims <- "A8"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Comma [0]"
dims <- "B8"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Currency"
dims <- "C8"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Currency [0]"
dims <- "D8"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

name <- "Per cent"
dims <- "E8"
wb$add_named_style(dims = dims, name = name)
wb$add_data(dims = dims, x = name)

# wb$open()

```

	A	B	C	D	E	F	G	H
1	Normal	Bad	Good	Neutral				
2	Calculation	Check Cell	Explinator	Input	Linked Cell	Note	Output	Warning Text
3	Heading	Heading 2	Heading 3	Heading 4	Title	Total		
4	20% - Accent1	20% - Accent2	20% - Accent3	20% - Accent4	20% - Accent5	20% - Accent6		
5	40% - Accent1	40% - Accent2	40% - Accent3	40% - Accent4	40% - Accent5	40% - Accent6		
6	60% - Accent1	60% - Accent2	60% - Accent3	60% - Accent4	60% - Accent5	60% - Accent6		
7	Accent1	Accent2	Accent3	Accent4	Accent5	Accent6		
8	Comma	Comma [0]	Currency	Currency [0]	Per cent			
9								
10								

12 Conditional Formatting

```
library(openxlsx2)
```

```
wb <- wb_workbook()
negStyle <- create_dxfs_style(font_color = wb_color(hex = "FF9C0006"), bg_fill = wb_color(hex = "FF9C0006"))
posStyle <- create_dxfs_style(font_color = wb_color(hex = "FF006100"), bg_fill = wb_color(hex = "FF006100"))
wb$styles_mgr$add(negStyle, "negStyle")
wb$styles_mgr$add(posStyle, "posStyle")
```

12.1 Rule applies to all each cell in range

	A	B
	-5	A
	-4	B
	-3	C
	-2	D
	-1	E
	0	F
	1	G
	2	H
	3	I
0	4	J
1	5	K
2		

```
wb$add_worksheet("cellIs")
wb$add_data("cellIs", -5:5)
wb$add_data("cellIs", LETTERS[1:11], start_col = 2)
wb$add_conditional_formatting(
  "cellIs",
  dims = "A1:A11",
```

```

        rule = "!<0",
        style = "negStyle"
    )
    wb$add_conditional_formatting(
        "cellIs",
        dims = "A1:A11",
        rule = "=<0",
        style = "posStyle"
    )

```

12.2 Highlight row dependent on first cell in row

	A	B	
1	-5 A		
2	-4 B		
3	-3 C		
4	-2 D		
5	-1 E		
6	0 F		
7	1 G		
8	2 H		
9	3 I		
10	4 J		
11	5 K		
12			

```

wb$add_worksheet("Moving Row")
wb$add_data("Moving Row", -5:5)
wb$add_data("Moving Row", LETTERS[1:11], start_col = 2)
wb$add_conditional_formatting(
    "Moving Row",
    dims = "A1:B11",
    rule = "$A1<0",
    style = "negStyle"
)
wb$add_conditional_formatting(
    "Moving Row",
    dims = "A1:B11",

```

```

    rule = "$A1>0",
    style = "posStyle"
)

```

12.3 Highlight column dependent on first cell in column

	A	B
-5	A	
-4	B	
-3	C	
-2	D	
-1	E	
0	F	
1	G	
2	H	
3	I	
4	J	
5	K	

```

wb$add_worksheet("Moving Col")
wb$add_data("Moving Col", -5:5)
wb$add_data("Moving Col", LETTERS[1:11], start_col = 2)
wb$add_conditional_formatting(
  "Moving Col",
  dims = "A1:B11",
  rule = "A$1<0",
  style = "negStyle"
)
wb$add_conditional_formatting(
  "Moving Col",
  dims = "A1:B11",
  rule = "A$1>0",
  style = "posStyle"
)

```


12.4 Highlight entire range cols X rows dependent only on cell A1

1	-5	A	
2	-4	B	
3	-3	C	
4	-2	D	
5	-1	E	
6	0	F	
7	1	G	
8	2	H	
9	3	I	
10	4	J	
11	5	K	
12			
13			
14			
15	x	y	
16	1	0,287578	
17	2	0,788305	
18	3	0,408977	
19	4	0,883017	
20	5	0,940467	
21	6	0,045556	
22	7	0,528105	
23	8	0,892419	
24	9	0,551435	
25	10	0,456615	
26			

```
wb$add_worksheet("Dependent on")
wb$add_data("Dependent on", -5:5)
wb$add_data("Dependent on", LETTERS[1:11], start_col = 2)
wb$add_conditional_formatting(
  "Dependent on",
  dims = "A1:B11",
  rule = "$A$1 < 0",
  style = "negStyle"
)
```

```
wb$add_conditional_formatting(
  "Dependent on",
  dims = "A1:B11",
  rule = "$A$1>0",
  style = "posStyle"
)
```

12.5 Highlight cells in column 1 based on value in column 2

```
wb$add_data("Dependent on", data.frame(x = 1:10, y = runif(10)), startRow = 15)
wb$add_conditional_formatting(
  "Dependent on",
  dims = "A16:A25",
  rule = "B16<0.5",
  style = "negStyle"
)
wb$add_conditional_formatting(
  "Dependent on",
  dims = "A16:A25",
  rule = "B16>=0.5",
  style = "posStyle"
)
```

12.6 Highlight duplicates using default style

	A	
1	D	
2	N	
3	F	
4	I	
5	J	
6	K	
7	E	
8	C	
9	K	
10	I	
11		

```

wb$add_worksheet("Duplicates")
wb$add_data("Duplicates", sample(LETTERS[1:15], size = 10, replace = TRUE))
wb$add_conditional_formatting(
  "Duplicates",
  dims = "A1:A10",
  type = "duplicatedValues"
)

```

12.7 Cells containing text

	A	B
1	D-L-N-S-G-I-V-B-P-M	
2	S-X-T-O-G-D-A-H-P-K	
3	P-T-H-C-D-Y-L-Q-J-K	
4	Y-W-H-N-U-M-B-K-V-Z	
5	F-Y-H-L-D-M-N-P-A-X	
6	H-J-Z-R-U-I-G-T-Y-K	
7	A-Y-S-J-U-M-K-T-G-I	
8	I-E-W-N-X-F-A-J-Q-R	
9	Z-U-G-Y-I-T-F-R-Q-E	
10	Y-T-C-N-A-B-D-J-V-E	

```

fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
wb$add_worksheet("containsText")
wb$add_data("containsText", sapply(1:10, fn))
wb$add_conditional_formatting(
  "containsText",
  dim = "A1:A10",
  type = "containsText",
  rule = "A"
)
wb$add_worksheet("notcontainsText")

```

12.8 Cells not containing text

	A	B
1	D-L-N-S-G-I	V-B-P-M
2	S-X-T-O-G-D-A-H-P-K	
3	P-T-H-C-D-Y-L-Q-J-K	
4	Y-W-H-N-U-M-B-K-V-Z	
5	F-Y-H-L-D-M-N-P-A-X	
6	H-J-Z-R-U-I-G-T-Y-K	
7	A-Y-S-J-U-M-K-T-G-I	
8	I-E-W-N-X-F-A-J-Q-R	
9	Z-U-G-Y-I-T-F-R-Q-E	
10	Y-T-C-N-A-B-D-J-V-E	
11		

```
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
wb$add_data("notcontainsText", x = sapply(1:10, fn))
wb$add_conditional_formatting(
  "notcontainsText",
  dim = "A1:A10",
  type = "notContainsText",
  rule = "A"
)
```

12.9 Cells begins with text

76	O-L-N-S-W-Q-I-M-X-F	
77	A-P-H-E-J-I-W-N-Z-Y	
78	F-T-H-N-W-X-K-E-V-A	
79	A-E-C-D-X-N-R-J-L-P	
80	C-L-E-M-H-Q-X-S-F-B	
81	Q-W-Z-H-S-R-V-E-N-L	

```
fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
wb$add_worksheet("beginsWith")
wb$add_data("beginsWith", x = sapply(1:100, fn))
wb$add_conditional_formatting(
```

```

    "beginsWith",
    dim = "A1:A100",
    type = "beginsWith",
    rule = "A"
)

```

12.10 Cells ends with text

60	K-X-H-A-C-N-J-O-G-P	
61	L-T-I-C-S-M-H-Q-D-J	
62	Q-J-E-K-I-L-X-D-B-A	
63	S-P-K-G-E-B-I-O-F-R	
64	W-D-V-O-F-C-J-E-X-A	
65	C-H-B-N-S-A-Z-E-M-I	
66	O-O-N-L-Z-W-I-L-L-E-E-S	

```

fn <- function(x) paste(sample(LETTERS, 10), collapse = "-")
wb$add_worksheet("endsWith")
wb$add_data("endsWith", x = sapply(1:100, fn))
wb$add_conditional_formatting(
  "endsWith",
  dim = "A1:A100",
  type = "endsWith",
  rule = "A"
)

```

12.11 Colorscale colors cells based on cell value

```

df <- read_xlsx("https://github.com/JanMarvin/openxlsx-data/raw/main/readTest.xlsx", sheet = "readTest")
wb$add_worksheet("colorScale", zoom = 30)
wb$add_data("colorScale", x = df, col_names = FALSE) ## write data.frame

```

Rule is a vector or colors of length 2 or 3 (any hex color or any of `colors()`). If rule is NULL, min and max of cells is used. Rule must be the same length as style or L.

```

wb$add_conditional_formatting(
  "colorScale",
  dims = wb_dims(x = df, col_names = FALSE),

```



Figure 12.1: Yep, that is a color scale image.

```

    style = c("black", "white"),
    rule = c(0, 255),
    type = "colorScale"
)
wb$set_col_widths("colorScale", cols = seq_along(df), widths = 1.07)
wb$set_row_heights("colorScale", rows = seq_len(nrow(df)), heights = 7.5)

```

12.12 Between

1	-5
2	-4
3	-3
4	-2
5	-1
6	0
7	1
8	2
9	3
10	4
11	5
12	

Highlight cells in interval $[-2, 2]$

```

wb$add_worksheet("between")
wb$add_data("between", -5:5)
wb$add_conditional_formatting(
  "between",
  dims = "A1:A11",
  type = "between",
  rule = c(-2, 2)
)
wb$add_worksheet("topN")

```

12.13 Top N

	A	B
1	x	y
2	1	1,604212
3	2	-0,51541
4	3	1,012537
5	4	-0,03594
6	5	-0,66734
7	6	0,92338
8	7	1,3811
9	8	0,87825
0	9	-0,5094
1	10	-0,46979

```
wb$add_data("topN", data.frame(x = 1:10, y = rnorm(10)))
```

Highlight top 5 values in column x

```
wb$add_conditional_formatting(  
  "topN",  
  dims = "A2:A11",  
  style = "posStyle",  
  type = "topN",  
  params = list(rank = 5)  
)
```

Highlight top 20 percentage in column y

```
wb$add_conditional_formatting(  
  "topN",  
  dims = "B2:B11",  
  style = "posStyle",  
  type = "topN",  
  params = list(rank = 20, percent = TRUE)  
)  
wb$add_worksheet("bottomN")
```


12.14 Bottom N

	A	B	
1	x	y	
2	1	1,377676	
3	2	0,352826	
4	3	0,829574	
5	4	-0,3387	
6	5	1,261035	
7	6	-0,80876	
8	7	0,625352	
9	8	-0,81717	
10	9	-2,46258	
11	10	-1,34296	
12			

```
wb$add_data("bottomN", data.frame(x = 1:10, y = rnorm(10)))
```

Highlight bottom 5 values in column x

```
wb$add_conditional_formatting(  
  "bottomN",  
  dims = "A2:A11",  
  style = "negStyle",  
  type = "bottomN",  
  params = list(rank = 5)  
)
```

Highlight bottom 20 percentage in column y

```
wb$add_conditional_formatting(  
  "bottomN",  
  dims = "B2:B11",  
  style = "negStyle",  
  type = "bottomN",  
  params = list(rank = 20, percent = TRUE)  
)  
wb$add_worksheet("logical operators")
```

12.15 Logical Operators

	A	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
10	10	
11		

You can use Excel's logical Operators

```
wb$add_data("logical operators", 1:10)
wb$add_conditional_formatting(
  "logical operators",
  dims = "A1:A10",
  rule = "OR($A1=1,$A1=3,$A1=5,$A1=7)"
)
```

12.16 (Not) Contains Blanks

	A	B	
1			
2	1	1	
3	2	2	
4			
5			
6			

```
wb$add_worksheet("contains blanks")
wb$add_data(x = c(NA, 1, 2, ''), colNames = FALSE, na.strings = NULL)
wb$add_data(x = c(NA, 1, 2, ''), colNames = FALSE, na.strings = NULL, start_col = 2)
```

```
wb$add_conditional_formatting(dims = "A1:A4", type = "containsBlanks")
wb$add_conditional_formatting(dims = "B1:B4", type = "notContainsBlanks")
```

12.17 (Not) Contains Errors

	A	B
1	1	1
2	#VALUE!	#VALUE!
3		
4		

```
wb$add_worksheet("contains errors")
wb$add_data(x = c(1, NaN), colNames = FALSE)
wb$add_data(x = c(1, NaN), colNames = FALSE, start_col = 2)
wb$add_conditional_formatting(dims = "A1:A3", type = "containsErrors")
wb$add_conditional_formatting(dims = "A1:A3", type = "notContainsErrors")
```

12.18 Iconset

	A
1	↓
2	↘
3	→

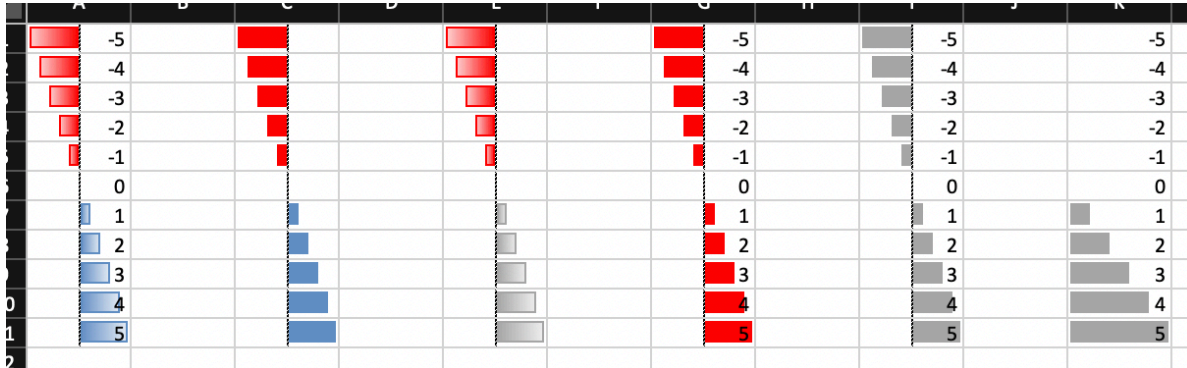
```
wb$add_worksheet("iconset")
wb$add_data(x = c(100, 50, 30), colNames = FALSE)
wb$add_conditional_formatting(
  dims = "A1:A6",
  rule = c(-67, -33, 0, 33, 67),
  type = "iconSet",
  params = list(
    percent = FALSE,
    iconSet = "5Arrows",
    reverse = TRUE)
)
```

12.19 Unique Values

	A
1	1
2	2
3	3
4	4
5	1
6	2
7	

```
wb$add_worksheet("unique values")
wb$add_data(x = c(1:4, 1:2), colNames = FALSE)
wb$add_conditional_formatting(dims = "A1:A6", type = "uniqueValues")
```

13 Databars



```
wb$add_worksheet("databar")
## Databars
wb$add_data("databar", -5:5, start_col = 1)
wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = "A1:A11",
  type = "dataBar"
) ## Default colors

wb$add_data("databar", -5:5, start_col = 3)
wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = "A1:A10",
  type = "dataBar",
  params = list(
    showValue = FALSE,
    gradient = FALSE
  )
) ## Default colors

wb$add_data("databar", -5:5, start_col = 5)
```

```

wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = "E1:E11",
  type = "dataBar",
  style = c("#a6a6a6"),
  params = list(showValue = FALSE)
)

wb$add_data("databar", -5:5, start_col = 7)
wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = "G1:G11",
  type = "dataBar",
  style = c("red"),
  params = list(
    showValue = TRUE,
    gradient = FALSE
  )
)


# custom color
wb$add_data("databar", -5:5, start_col = 9)
wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = wb_dims(cols = 9, rows = 1:11),
  type = "dataBar",
  style = c("#a6a6a6", "#a6a6a6"),
  params = list(showValue = TRUE, gradient = FALSE)
)

# with rule
wb$add_data(x = -5:5, start_col = 11)
wb <- wb_add_conditional_formatting(
  wb,
  "databar",
  dims = wb_dims(cols = 11, rows = 1:11),
  type = "dataBar",
  rule = c(0, 5),

```

```
style = c("#a6a6a6", "#a6a6a6"),  
params = list(showValue = TRUE, gradient = FALSE)  
)
```

14 Sparklines

	A	B	C	D	E	F	G	H	I	J	K	L	
1	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb		
2	21	6	160	110	3.9	2.62	16.46	0	1	4	4		
3	21	6	160	110	3.9	2.875	17.02	0	1	4	4		
4	22.8	4	108	93	3.85	2.32	18.61	1	1	4	1		

```
s1 <- create_sparklines("Sheet 1", "A3:K3", "L3")
wb <- wb_workbook() %>%
  wb_add_worksheet() %>%
  wb_add_data(x = mtcars) %>%
  wb_add_sparklines(sparklines = s1)
```


15 charts

The following manual will present various ways to add plots and charts to `openxlsx2` worksheets and even chartsheets. This assumes that you have basic knowledge how to handle `openxlsx2` and are familiar with either the default R graphics functions like `plot()` or `barplot()` and `grDevices`, or with the packages `{ggplot2}`, `{rvg}` or `{mschart}`. There are plenty of other manuals that cover using these better than we could ever tell you to.

```
library(openxlsx2) # openxlsx2 >= 0.4 for mschart and rvg support

## create a workbook
wb <- wb_workbook()
```

15.1 Add plot to workbook

You can include any image in PNG or JPEG format. Simply open a device and save the output and pass it to the worksheet with `wb_add_image()`.

```
myplot <- tempfile(fileext = ".jpg")
jpeg(myplot)
print(plot(AirPassengers))
#> NULL
dev.off()
#> pdf
#> 2

# Add basic plots to the workbook
wb$add_worksheet("add_image")$add_image(file = myplot)
```

15.2 Add {ggplot2} plot to workbook

You can include `{ggplot2}` plots similar to how you would include them with `openxlsx`. Call the plot first and afterwards use `wb_add_plot()`.

```

if (requireNamespace("ggplot2")) {

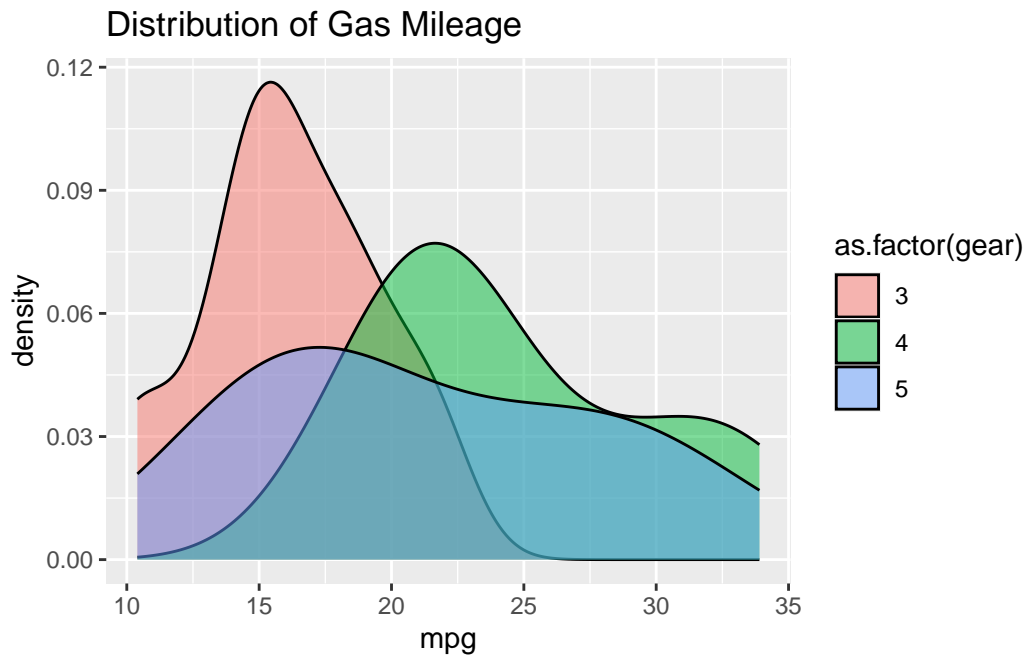
library(ggplot2)

print(ggplot(mtcars, aes(x = mpg, fill = as.factor(gear))) +
  ggtitle("Distribution of Gas Mileage") +
  geom_density(alpha = 0.5))

# Add ggplot to the workbook
wb$add_worksheet("add_plot")$
  add_plot(width = 5, height = 3.5, fileType = "png", units = "in")

}
#> Loading required namespace: ggplot2
#> Loading required namespace: testthat

```



15.3 Add plot via {rvg}

If you want vector graphics that can be modified in spreadsheet software the `dml_xlsx()` device comes in handy. You can pass the output via `wb_add_drawing()`.

```

if (requireNamespace("ggplot2") && requireNamespace("rvg")) {

  library(rvg)

  ## create rvg example
  tmp <- tempfile(fileext = ".xml")
  dml_xlsx(file = tmp, fonts = list(sans = "Bradley Hand"))
  print(ggplot(data = iris,
    mapping = aes(x = Sepal.Length, y = Petal.Width)) +
    geom_point() + labs(title = "With font Bradley Hand") +
    theme_minimal(base_family = "sans", base_size = 18))
  dev.off()

  # Add rvg to the workbook
  wb$add_worksheet("add_drawing")$
    add_drawing(xml = tmp)$
    add_drawing(xml = tmp, dims = NULL)

}
#> Loading required namespace: rvg

```

15.4 Add {mschart} plots

If you want native open xml charts, have a look at {mschart}. Create one of the chart files and pass it to the workbook with `wb_add_mschart()`. There are two options possible. 1. Either the default {mschart} output identical to the one in {officer}. Passing a data object and let {mschart} prepare the data. In this case `wb_add_mschart()` will add a new data region. 2. Passing a `wb_data()` object to {mschart}. This object contains references to the data on the worksheet and allows using data “as is”.

```

if (requireNamespace("mschart")) {

  library(mschart) # mschart >= 0.4 for openxlsx2 support

  ## create chart from mschart object (this creates new input data)
  mylc <- ms_linechart(
    data = browser_ts,
    x = "date",
    y = "freq",
    group = "browser"
  )
}

```

```

)

wb$add_worksheet("add_mschart")$add_mschart(dims = "A10:G25", graph = mylc)

## create chart referencing worksheet cells as input
# write data starting at B2
wb$add_worksheet("add_mschart - wb_data")$
  add_data(x = mtcars, dims = "B2")$
  add_data(x = data.frame(name = rownames(mtcars)), dims = "A2")

# create wb_data object this will tell this mschart
# from this PR to create a file corresponding to openxlsx2
dat <- wb_data(wb, dims = "A2:G10")

# create a few mscharts
scatter_plot <- ms_scatterchart(
  data = dat,
  x = "mpg",
  y = c("disp", "hp")
)

bar_plot <- ms_barchart(
  data = dat,
  x = "name",
  y = c("disp", "hp")
)

area_plot <- ms_areachart(
  data = dat,
  x = "name",
  y = c("disp", "hp")
)

line_plot <- ms_linechart(
  data = dat,
  x = "name",
  y = c("disp", "hp"),
  labels = c("disp", "hp")
)

```

```

# add the charts to the data
wb <- wb %>%
  wb_add_mschart(dims = "F4:L20", graph = scatter_plot) %>%
  wb_add_mschart(dims = "F21:L37", graph = bar_plot) %>%
  wb_add_mschart(dims = "M4:S20", graph = area_plot) %>%
  wb_add_mschart(dims = "M21:S37", graph = line_plot)

# add chartsheet
wb <- wb %>%
  wb_add_chartsheet() %>%
  wb_add_mschart(graph = scatter_plot)
}
#> Loading required namespace: mschart

```

16 Pivot tables

```
wb <- wb_workbook()$
  add_worksheet()$
  add_data(x = esoph)

df <- wb_data(wb)

wb$add_pivot_table(df, rows = "agegp", cols = "tobgp", data = c("ncontrols"))
wb$add_pivot_table(df, rows = "agegp", data = c("ncontrols", "ncases"))
wb$add_pivot_table(df, rows = "agegp", cols = "tobgp", data = c("ncontrols", "ncases"))

wb <- wb_workbook()$
  add_worksheet()$
  add_data(x = mtcars)

df <- wb_data(wb)

wb$add_pivot_table(df, dims = "A1", rows = "cyl", cols = "gear", data = c("disp", "hp"))
wb$add_pivot_table(df, dims = "A10", sheet = 2, rows = "cyl", cols = "gear", data = c("disp", "hp"))
wb$add_pivot_table(df, dims = "A20", sheet = 2, rows = "cyl", cols = "gear", data = c("disp", "hp"))
wb$add_pivot_table(df, dims = "A30", sheet = 2, rows = "cyl", cols = "gear", data = c("disp", "hp"))

## Pivot table example 1
wb <- wb_workbook() %>% wb_add_worksheet() %>% wb_add_data(x = mtcars, inline_strings = F)

df <- wb_data(wb)

# basic pivot table with filter, rows, cols and data
wb$add_pivot_table(df, dims = "A3", filter = "mpg", rows = "cyl", cols = "gear", data = "disp")

# same pivot table, but with "count" instead of "sum" and no style
wb$add_pivot_table(df, dims = "A10", sheet = 2, rows = "cyl", cols = "gear", data = c("disp", "hp"))

# nested pivot table with two variables for column, row and data and two different functions
```

```

# uses an autoformatid (not that I like it, just because I can do it)
wb$add_pivot_table(df, dims = "A20", sheet = 2, rows = c("cyl", "mpg"), cols = c("vs", "gear"),
  params = list(applyAlignmentFormats = "1",
    applyNumberFormats = "1",
    applyBorderFormats = "1",
    applyFontFormats = "1",
    applyPatternFormats = "1",
    applyWidthHeightFormats = "1",
    autoFormatId = "4099"))

# multiple filters on a pivot table
wb$add_pivot_table(df, dims = "A3", filter = c("am", "vs", "mpg", "hp", "wt"), rows = "cyl")

# using custom caption
wb$add_pivot_table(df, dims = "A20", sheet = 3, rows = "cyl", cols = "gear", data = c("display"))

# wb$open()

## Pivot table example 2
# pivot table with blanks and character variables on column and row
wb <- wb_workbook()$add_worksheet()$add_data(x = esoph)
df <- wb_data(wb, dims = "A1:E95")
wb$add_pivot_table(df, rows = "agegp", cols = "tobgp", data = c("ncontrols"))
# wb$open()

# original pivot table as reference
library(pivottabler)

pt <- PivotTable$new()
pt$addData(bhmtrains)
pt$addColumnDataGroups("TrainCategory")
pt$addRowDataGroups("TOC",
  outlineBefore=list(isEmpty=FALSE, groupStyleDeclarations=list(color="black"),
  outlineTotal=list(isEmpty=FALSE, groupStyleDeclarations=list(color="black"))
pt$addRowDataGroups("PowerType", addTotal=FALSE)
pt$defineCalculation(calculationName="TotalTrains", summariseExpression="n()")

```

		Express Passenger	Ordinary Passenger	Total
Arriva Trains Wales		3079	830	3909
	DMU	3079	830	3909
CrossCountry		22865	63	22928
	DMU	22133	63	22196
	HST	732		732
London Midland		14487	33792	48279
	DMU	5638	5591	11229
	EMU	8849	28201	37050
Virgin Trains		8594		8594
	DMU	2137		2137
	EMU	6457		6457
Total		49025	34685	83710

```

# use A:P
wb <- wb_workbook()$add_worksheet()$add_data(x = bhmtrains, na.strings = NULL)
df <- wb_data(wb, dims = "A:P")

# use TrainCategory on column and data
wb$add_pivot_table(
  df,
  rows = c("TOC", "PowerType"),
  cols = "TrainCategory",
  data = "TrainCategory",
  fun = "count"
)
# wb$open()

```


	A	B	C	D	E
1					
2					
3	count of TrainCategory	Column Labels			
4	Row Labels	Express Passenger	Ordinary Passenger	(blank)	Grand Total
5	▣ Arriva Trains Wales	3079	830		3909
6	DMU	3079	830		3909
7	▣ CrossCountry	22865	63		22928
8	DMU	22133	63		22196
9	HST	732			732
10	▣ London Midland	14487	33792		48279
11	DMU	5638	5591		11229
12	EMU	8849	28201		37050
13	▣ Virgin Trains	8594			8594
14	DMU	2137			2137
15	EMU	6457			6457
16	▣ (blank)				
17	(blank)				
18	Grand Total	49025	34685		83710
19					

17 Form control

```
wb <- wb_workbook()$
# Checkbox
add_worksheet()$
add_form_control(dims = "B2")$
add_form_control(dims = "B3", text = "A text")$
add_data(dims = "A4", x = 0, colNames = FALSE)$
add_form_control(dims = "B4", link = "A4")$
add_data(dims = "A5", x = TRUE, colNames = FALSE)$
add_form_control(dims = "B5", range = "'Sheet 1'!A5", link = "B5")$
# Radio
add_worksheet()$
add_form_control(dims = "B2", type = "Radio")$
add_form_control(dims = "B3", type = "Radio", text = "A text")$
add_data(dims = "A4", x = 0, colNames = FALSE)$
add_form_control(dims = "B4", type = "Radio", link = "A4")$
add_data(dims = "A5", x = 1, colNames = FALSE)$
add_form_control(dims = "B5", type = "Radio")$
# Drop
add_worksheet()$
add_form_control(dims = "B2", type = "Drop")$
add_form_control(dims = "B3", type = "Drop", text = "A text")$
add_data(dims = "A4", x = 0, colNames = FALSE)$
add_form_control(dims = "B4", type = "Drop", link = "A1", range = "D4:D15")$
add_data(dims = "A5", x = 1, colNames = FALSE)$
add_form_control(dims = "B5", type = "Drop", link = "'Sheet 3'!D1:D26", range = "A1")$
add_data(dims = "D1", x = letters)
```

		<input checked="" type="checkbox"/>	
		<input checked="" type="checkbox"/> A text	
	0	<input type="checkbox"/>	
	1	<input checked="" type="checkbox"/>	

	A	B	C	
1				
2		<input type="radio"/>		
3		<input type="radio"/> A text		
4	3	<input checked="" type="radio"/>		
5	1	<input type="radio"/>		
6				
7				
8				
9				

	A	B	C	D	E
1	2			a	
2		<input type="text"/>		b	
3		<input type="text"/>		c	
4	0	e <input type="text"/>		d	
5	1	<input type="text"/>		e	
6				f	
7				g	
8				h	
9				i	
10				j	

18 Upgrade from openxlsx

18.1 Basic read and write functions

Welcome to the `openxlsx2` update vignette. In this vignette we will take some common code examples from `openxlsx` and show you how similar results can be replicated in `openxlsx2`. Thank you for taking a look, and let's get started. While previous `openxlsx` functions used the `.` in function calls, as well as camelCase, we have tried to switch to snake_case (this is still a work in progress, there may still be function arguments that use camelCase).

18.1.1 Read xlsx or xlsxm files

The basic read function changed from `read.xlsx` to `read_xlsx`. Using a default xlsx file included in the package:

```
file <- system.file("extdata", "openxlsx2_example.xlsx", package = "openxlsx2")
```

The old syntax looked like this:

```
# read in openxlsx
openxlsx::read.xlsx(xlsxFile = file)
```

This has changed to this:

```
# read in openxlsx2
openxlsx2::read_xlsx(file = file)
```

#>	Var1	Var2	NA	Var3	Var4	Var5	Var6	Var7	Var8
#> 3	TRUE	1	NA	1	a	2023-05-29 3209324	This	#DIV/0!	01:27:15
#> 4	TRUE	NA	NA	#NUM!	b	2023-05-23	<NA>	0	14:02:57
#> 5	TRUE	2	NA	1.34	c	2023-02-01	<NA>	#VALUE!	23:01:02
#> 6	FALSE	2	NA	<NA>	#NUM!	<NA>	<NA>	2	17:24:53
#> 7	FALSE	3	NA	1.56	e	<NA>	<NA>	<NA>	<NA>
#> 8	FALSE	1	NA	1.7	f	2023-03-02	<NA>	2.7	08:45:58
#> 9	NA	NA	NA	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
#> 10	FALSE	2	NA	23	h	2023-12-24	<NA>	25	<NA>

```
#> 11 FALSE      3 NA  67.3      i 2023-12-25      <NA>      3      <NA>
#> 12      NA      1 NA  123    <NA> 2023-07-31      <NA>     122     <NA>
```

As you can see, we return the spreadsheet return codes (e.g., `#NUM`) in `openxlsx2`. Another thing to see above, we return the cell row as rowname for the data frame returned. `openxlsx2` should return a data frame of the selected size, even if it empty. If you preferred `openxlsx::readWorkbook()` this has become `wb_read()`. All of these are wrappers for the newly introduced function `wb_to_df()` which provides the most options. `read_xlsx()` and `wb_read()` were created for backward comparability.

18.2 Write xlsx files

Basic writing in `openxlsx2` behaves identical to `openxlsx`. Though be aware that `overwrite` is an optional parameter in `openxlsx2` and just like in other functions like `base::write.csv()` if you write onto an existing file name, this file will be replaced.

Setting the output to some temporary xlsx file

```
output <- temp_xlsx()
```

The previous write function looks like this:

```
# write in openxlsx
openxlsx::write_xlsx(iris, file = output, colNames = TRUE)
```

The new function looks quite similar:

```
# write in openxlsx2
openxlsx2::write_xlsx(iris, file = output, col_names = TRUE)
```

18.3 Basic workbook functions

Workbook functions have been renamed to begin with `wb_` there are plenty of these in the package, therefore looking at the man pages seems to be the fastest way. Yet, it all begins with loading the workbook.

18.3.1 Loading a workbook

A major feature in `openxlsx` are workbooks. Obviously they remain a central piece in `openxlsx2`. Previously you would load them with:

```
wb <- openxlsx::loadWorkbook(file = file)
```

In `openxlsx2` loading was changed to:

```
wb <- wb_load(file = file)
```

There are plenty of functions to interact with workbooks and we will not describe every single one here. A detailed list can be found over at [our references](#)

18.3.2 Styles

One of the biggest user facing change was the removal of the `stylesObject`. In the following section we use code from `openxlsx::addStyle()`

```
# openxlsx
## Create a new workbook
wb <- createWorkbook(creator = "My name here")
addWorksheet(wb, "Expenditure", gridLines = FALSE)
writeData(wb, sheet = 1, USPersonalExpenditure, rowNames = TRUE)

## style for body
bodyStyle <- createStyle(border = "TopBottom", borderColor = "#4F81BD")
addStyle(wb, sheet = 1, bodyStyle, rows = 2:6, cols = 1:6, gridExpand = TRUE)

## set column width for row names column
setColWidths(wb, 1, cols = 1, widths = 21)
```

In `openxlsx2` the same code looks something like this:

```
# openxlsx2 chained
border_color <- wb_color(hex = "#4F81BD")
wb <- wb_workbook(creator = "My name here")$
  add_worksheet("Expenditure", grid_lines = FALSE)$
  add_data(x = USPersonalExpenditure, row_names = TRUE)$
  add_border( # add the outer and inner border
    dims = "A1:F6",
```

```

    top_border = "thin", top_color = border_color,
    bottom_border = "thin", bottom_color = border_color,
    inner_hgrid = "thin", inner_hcolor = border_color,
    left_border = "", right_border = ""
  )$
  set_col_widths( # set column width
    cols = 1:6,
    widths = c(20, rep(10, 5))
  )$ # remove the value in A1
  add_data(dims = "A1", x = "")

```

The code above uses chaining. If you prefer piping, we provide the chained functions with the prefix `wb_` so `wb_add_worksheet()`, `wb_add_data()`, `wb_add_border()` and `wb_set_col_widths()` would be the functions to use with pipes `%>%` or `|>`.

With pipes the code from above becomes

```

# openxlsx2 with pipes
border_color <- wb_color(hex = "4F81BD")
wb <- wb_workbook(creator = "My name here") %>%
  wb_add_worksheet(sheet = "Expenditure", grid_lines = FALSE) %>%
  wb_add_data(x = USPersonalExpenditure, row_names = TRUE) %>%
  wb_add_border( # add the outer and inner border
    dims = "A1:F6",
    top_border = "thin", top_color = border_color,
    bottom_border = "thin", bottom_color = border_color,
    inner_hgrid = "thin", inner_hcolor = border_color,
    left_border = "", right_border = ""
  ) %>%
  wb_set_col_widths( # set column width
    cols = 1:6,
    widths = c(20, rep(10, 5))
  ) %>% # remove the value in A1
  wb_add_data(dims = "A1", x = "")

```

Be aware that chains modify an object in place and pipes do not.

```

# openxlsx2
wbp <- wb_workbook() %>% wb_add_worksheet()
wbc <- wb_workbook()$add_worksheet()

# need to assign wbp

```

```
wbp <- wbp %>% wb_add_data(x = iris)
wbc$add_data(x = iris)
```

You can re-use styles with `wb_get_cell_style()` and `wb_set_cell_style()`. Abandoning `stylesObject` in `openxlsx2` has the huge benefit that we can import and export a spreadsheet without changing any cell style. It is still possible to modify a cell style with `wb_add_border()`, `wb_add_fill()`, `wb_add_font()` and `wb_add_numfmt()`.

Additional examples regarding styles can be found in the styles vignette.

18.3.3 Conditional formatting

See `vignette("conditional-formatting")` for extended examples on formatting.

Here is a minimal example:

```
# openxlsx2 with chains
wb <- wb_workbook()$
  add_worksheet("a")$
  add_data(x = 1:4, col_names = FALSE)$
  add_conditional_formatting(dims = "A1:A4", rule = ">2")

# openxlsx2 with pipes
wb <- wb_workbook() %>%
  wb_add_worksheet("a") %>%
  wb_add_data(x = 1:4, col_names = FALSE) %>%
  wb_add_conditional_formatting(dims = "A1:A4", rule = ">2")
```

18.3.4 Data validation

Similarly, data validation has been updated and improved. This `openxlsx` code for data validation

```
# openxlsx
wb <- createWorkbook()
addWorksheet(wb, "Sheet 1")
writeDataTable(wb, 1, x = iris[1:30, ])
dataValidation(wb, 1,
  col = 1:3, rows = 2:31, type = "whole",
  operator = "between", value = c(1, 9)
)
```


looks in `openxlsx2` something like this:

```
# openxlsx2 with chains
wb <- wb_workbook()$
  add_worksheet("Sheet 1")$
  add_data_table(1, x = iris[1:30, ])$
  add_data_validation(1,
    dims = wb_dims(rows = 2:31, cols = 1:3),
    # alternatively, dims can also be "A2:C31" if you know the span in your Excel workbook
    type = "whole",
    operator = "between",
    value = c(1, 9)
  )

# openxlsx2 with pipes
wb <- wb_workbook() %>%
  wb_add_worksheet("Sheet 1") %>%
  wb_add_data_table(1, x = iris[1:30, ]) %>%
  wb_add_data_validation(
    sheet = 1,
    dims = "A2:C31", # alternatively, dims = wb_dims(rows = 2:31, cols = 1:3)
    type = "whole",
    operator = "between",
    value = c(1, 9)
  )
```

18.3.5 Saving

Saving has been switched from `saveWorkbook()` to `wb_save()` and opening a workbook has been switched from `openXL()` to `wb_open()`.

References

- Allen, Michael. 2023. *Readxlsb: Read 'Excel' Binary (.xlsb) Workbooks*. <https://CRAN.R-project.org/package=readxlsb>.
- Barbone, Jordan Mark, and Jan Marvin Garbuszus. 2023. *Openxlsx2: Read, Write and Edit 'Xlsx' Files*. <https://github.com/JanMarvin/openxlsx2>.
- Chang, Winston. 2021. *R6: Encapsulated Classes with Reference Semantics*. <https://CRAN.R-project.org/package=R6>.
- Dragulescu, Adrian, and Cole Arendt. 2023. *Xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. <https://CRAN.R-project.org/package=xlsx>.
- Eddelbuettel, Dirk, and Romain François. 2011. “Rcpp: Seamless R and C++ Integration.” *Journal of Statistical Software* 40 (8): 1–18. <https://doi.org/10.18637/jss.v040.i08>.
- Garmonsway, Duncan. 2022. *Tidylx: Read Untidy Excel Files*. <https://CRAN.R-project.org/package=tidylx>.
- Kapoulkine, Arseny. 2006-2022. *Pugixml*. <https://pugixml.org>.
- Ooms, Jeroen. 2023. *Writexl: Export Data Frames to Excel 'Xlsx' Format*. <https://CRAN.R-project.org/package=writexl>.
- Schauberg, Philipp, and Alexander Walker. 2023. *Openxlsx: Read, Write and Edit Xlsx Files*. <https://CRAN.R-project.org/package=openxlsx>.
- Schwartz, Marc. 2022. *WriteXLS: Cross-Platform Perl Based r Function to Create Excel 2003 (XLS) and Excel 2007 (XLSX) Files*. <https://CRAN.R-project.org/package=WriteXLS>.
- Wickham, Hadley, and Jennifer Bryan. 2023. *Readxl: Read Excel Files*. <https://CRAN.R-project.org/package=readxl>.