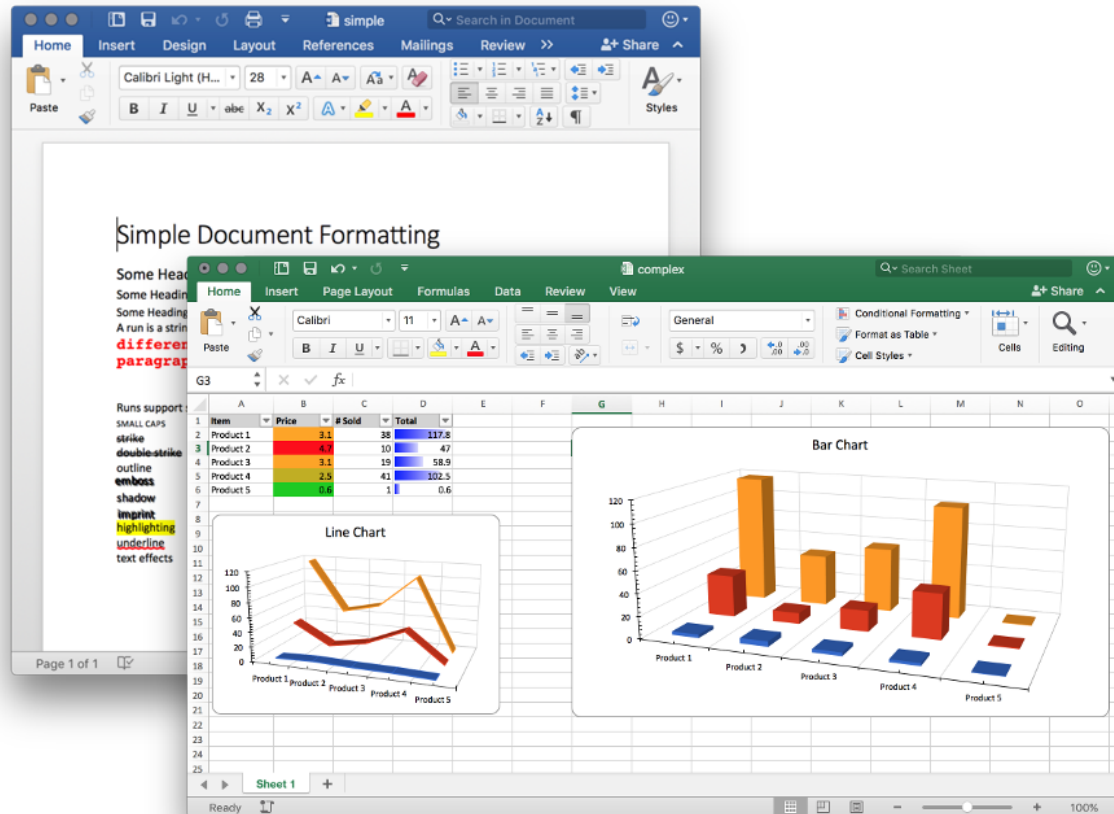


---

**unioffice** is a library for creation of Office Open XML documents (.docx, .xlsx and .pptx). Its goal is to be the most compatible and highest performance Go library for creation and editing of docx/xlsx/pptx files.

build passing release v1.34.0 license UniDoc EULA godoc reference



## Status

- Documents (docx) [Word]
  - Read/Write/Edit
  - Formatting
  - Images
  - Tables
  - Word to PDF (docx to pdf)
- Spreadsheets (xlsx) [Excel]
  - Read/Write/Edit

- 
- Cell formatting including conditional formatting
  - Cell validation (drop down combobox, rules, etc.)
  - Retrieve cell values as formatted by Excel (e.g. retrieve a date or number as displayed in Excel)
  - Formula Evaluation (100+ functions supported currently, more will be added as required)
  - Embedded Images
  - All chart types
- PowerPoint (pptx) [PowerPoint]
    - Creation from templates
    - Textboxes/shapes

## Performance

There has been a great deal of interest in performance numbers for spreadsheet creation/reading lately, so here are unioffice numbers for this benchmark which creates a sheet with 30k rows, each with 100 columns.

```
1 creating 30000 rows * 100 cells took 3.92506863s
2 saving took 89ns
3 reading took 9.522383048s
```

Creation is fairly fast, saving is very quick due to no reflection usage, and reading is a bit slower. The downside is that the binary is large (33MB) as it contains generated structs, serialization and deserialization code for all of DOCX/XLSX/PPTX.

## Installation

```
1 go get github.com/unidoc/unioffice/
```

## License key

This software package (unioffice) is a commercial product and requires a license code to operate.

To Get a Metered License API Key in the Free Tier, sign up on <https://cloud.unidoc.io>

## Document Examples

- Simple Text Formatting Text font colors, sizes, highlighting, etc.

- 
- Auto Generated Table of Contents Creating document headings with an auto generated TOC based off of the headings
  - Floating Image Placing an image somewhere on a page, absolutely positioned with different text wrapping.
  - Header & Footer Creating headers and footers including page numbering.
  - Multiple Headers & Footers Using different headers and footers depending on document section.
  - Inline Tables Adding an table with and without borders.
  - Using Existing Word Document as a Template Opening a document as a template to re-use the styles created in the document.
  - Filling out Form Fields Opening a document with embedded form fields, filling out the fields and saving the result as a new filled form.
  - Editing an existing document Open an existing document and replace/remove text without modifying formatting.

## **Spreadsheet Examples**

- Simple A simple sheet with a few cells
- Named Cells Different ways of referencing rows and cells
- Cell Number/Date/Time Formats Creating cells with various number/date/time formats
- Line Chart/Line Chart 3D Line Charts
- Bar Chart Bar Charts
- Mutiple Charts Multiple charts on a single sheet
- Named Cell Ranges Naming cell ranges
- Merged Cells Merge and unmerge cells
- Conditional Formatting Conditionally formatting cells, styling, gradients, icons, data bar
- Complex Multiple charts, auto filtering and conditional formatting
- Borders Individual cell borders and rectangular borders around a range of cells.
- Validation Data validation including combo box dropdowns.
- Frozen Rows/Cols A sheet with a frozen header column and row

## **Presentation Examples**

- Simple Text Boxes Simple text boxes and shapes
- Images Simple image insertion
- Template Creating a presentation from a template

---

## Raw Types

The OOXML specification is large and creating a friendly API to cover the entire specification is a very time consuming endeavor. This library attempts to provide an easy to use API for common use cases in creating OOXML documents while allowing users to fall back to raw document manipulation should the library's API not cover a specific use case.

The raw XML based types reside in the [schema/](#) directory. These types are accessible from the wrapper types via a `X()` method that returns the raw type.

For example, the library currently doesn't have an API for setting a document background color. However it's easy to do manually via editing the `CT_Background` element of the document.

```
1 doc := document.New()
2 doc.X().Background = wordprocessingml.NewCT_Background()
3 doc.X().Background.ColorAttr = &wordprocessingml.ST_HexColor{}
4 doc.X().Background.ColorAttr.ST_HexColorRGB = color.RGB(50, 50, 50).
   AsRGBString()
```

## Contribution guidelines

If you are interested in contributing, please contact us.

## Development Notes

The bash script file `run_test.sh` could be used to run test and update the test result (if required). This script could receive the following parameter: - `-s`: Save a baseline, updates all the test result. - `-v`: Run the test in verbose mode. - `-t` or `--testname` [test name]: Run a specific test name. For example `-t AddImage` would be running a `TestAddImage` test.

To run the script in dockerized environment, use the provided `Makefile` such as:

```
1 make docker-test
```

or

```
1 make docker-update-testdata
```

## Go Version Compatibility

Officially we support three latest Go versions, but internally we would test the build with up to five latest Go versions in our CI runner.

---

## **Support and consulting**

Please email us at [support@unidoc.io](mailto:support@unidoc.io) for any queries.

If you have any specific tasks that need to be done, we offer consulting in certain cases. Please contact us with a brief summary of what you need and we will get back to you with a quote, if appropriate.

## **License agreement**

The use of this software package is governed by the end-user license agreement (EULA) available at: <https://unidoc.io/eula/>