

XLSX Plugin 1.1

Create and modify XLSX documents with Hollywood

Andreas Falkenhahn

Table of Contents

1	General information	1
1.1	Introduction	1
1.2	Terms and conditions	1
1.3	Requirements	2
1.4	Installation	2
2	About <code>xlsx.hwp</code>	5
2.1	Credits	5
2.2	Frequently asked questions	5
2.3	Known issues	5
2.4	Future	5
2.5	History	5
3	Usage	7
3.1	Interfaces	7
3.2	Library interface	7
3.3	Serialization interface	7
4	Function reference	9
4.1	<code>xlsx.AddSheet</code>	9
4.2	<code>xlsx.CellRange</code>	9
4.3	<code>xlsx.ClearCellFormula</code>	10
4.4	<code>xlsx.ClearCellValue</code>	11
4.5	<code>xlsx.Close</code>	12
4.6	<code>xlsx.Create</code>	12
4.7	<code>xlsx.DeleteProperty</code>	13
4.8	<code>xlsx.DeleteSheet</code>	13
4.9	<code>xlsx.GetCellFormula</code>	14
4.10	<code>xlsx.GetCellReference</code>	15
4.11	<code>xlsx.GetCellValue</code>	15
4.12	<code>xlsx.GetColumnCount</code>	17
4.13	<code>xlsx.GetColumnWidth</code>	17
4.14	<code>xlsx.GetObjectType</code>	18
4.15	<code>xlsx.GetProperty</code>	18
4.16	<code>xlsx.GetRowCount</code>	19
4.17	<code>xlsx.GetRowHeight</code>	20
4.18	<code>xlsx.GetSheetCount</code>	20
4.19	<code>xlsx.GetSheetIndex</code>	21
4.20	<code>xlsx.GetSheetName</code>	21
4.21	<code>xlsx.GetSheetType</code>	21
4.22	<code>xlsx.GetSheetVisibility</code>	22
4.23	<code>xlsx.HaveCellFormula</code>	23

4.24	xlsx.HideColumn	23
4.25	xlsx.HideRow	24
4.26	xlsx.IsColumnHidden	24
4.27	xlsx.IsRowHidden	25
4.28	xlsx.IsSheetActive	25
4.29	xlsx.IsSheetSelected	26
4.30	xlsx.MoveSheet	26
4.31	xlsx.Open	27
4.32	xlsx.Save	28
4.33	xlsx.SaveAs	28
4.34	xlsx.SetCellFormula	28
4.35	xlsx.SetCellValue	29
4.36	xlsx.SetColumnWidth	31
4.37	xlsx.SetDefaultSheet	31
4.38	xlsx.SetProperty	32
4.39	xlsx.SetRowHeight	32
4.40	xlsx.SetSheetActive	33
4.41	xlsx.SetSheetName	33
4.42	xlsx.SetSheetSelected	34
4.43	xlsx.SetSheetVisibility	34
4.44	xlsx.UseSharedStrings	35
Appendix A Licenses		37
A.1	OpenXLSX license	37
A.2	pugixml license	37
A.3	MiniZ license	38
Index		39

1 General information

1.1 Introduction

The XLSX plugin allows you to conveniently read and write XLSX documents from Hollywood scripts. It offers a wide variety of functions to set and get cell values, cell types, cell formulas, document/worksheet properties and several other attributes. It also offers an iterator function for a high performance iteration of a large number of cells.

On top of that, the XLSX plugin also supports Hollywood's serialization interface which means that you can conveniently serialize Hollywood tables to XLSX documents by just a single call to Hollywood's `SerializeTable()` function. In the same manner you can also deserialize whole XLSX documents into Hollywood tables by a single call to Hollywood's `DeserializeTable()` function. It just doesn't get any easier!

1.2 Terms and conditions

xlsx.hwp is © Copyright 2022-2026 by Andreas Falkenhahn (in the following referred to as "the author"). All rights reserved.

The program is provided "as-is" and the author cannot be made responsible of any possible harm done by it. You are using this program absolutely at your own risk. No warranties are implied or given by the author.

This plugin may be freely distributed as long as the following three conditions are met:

1. No modifications must be made to the plugin.
2. It is not allowed to sell this plugin.
3. If you want to put this plugin on a coverdisc, you need to ask for permission first.

This software uses OpenXLSX Copyright (C) 2020 Kenneth Trolldal Balslev. See [Section A.1 \[OpenXLSX license\]](#), page 37, for details.

This software uses pugixml Copyright (C) 2006-2022 Arseny Kapoulkine. See [Section A.2 \[pugixml license\]](#), page 37, for details.

This software uses MiniZ Copyright (C) 2010-2014 Rich Geldreich. See [Section A.3 \[MiniZ license\]](#), page 38, for details.

All trademarks are the property of their respective owners.

DISCLAIMER: THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDER AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU

FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.3 Requirements

- Hollywood 9.0 or better

1.4 Installation

Installing `xlsx.hwp` is straightforward and simple: Just copy the file `xlsx.hwp` for the platform of your choice to Hollywood's plugins directory. The location where Hollywood plugins must be installed differs from platform to platform. Here is a breakdown of the different search paths that Hollywood will scan for plugins:

- AmigaOS and compatibles: Plugins can either be stored in a directory named `Plugins` that is in the same directory as the Hollywood interpreter or they can be installed globally in `LIBS:Hollywood`. Hollywood will first look for plugins in the `Plugins` directory in its program directory, then it will scan `LIBS:Hollywood`.
- Android: The Android version of Hollywood also supports Hollywood plugins. You have to copy them to the directory `Hollywood/Plugins` on your SD card. Hollywood will scan this location on every startup and load all plugins from there.
- Linux: Plugins must be stored in a directory named `Plugins` (case sensitive!) that is in the same directory as the Hollywood interpreter.
- macOS: `~/Library/Application Support/AirsoftSoftwair/Hollywood/Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins inside Hollywood's app bundle in `Hollywood.app/Contents/Resources/Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges.
- Windows: `C:\Users\...\AppData\Local\AirsoftSoftwair\Hollywood\Plugins` is the directory that should be used for storing plugins. If it doesn't exist, just create it. Alternatively, you can also store plugins in the `Plugins` directory in Hollywood's installation directory, e.g. in `C:\Program Files\Hollywood\Plugins` but this isn't recommended because it requires you to modify the Hollywood distribution and it might also require administrator privileges to write to Hollywood's installation directory.

On Windows you should also copy the file `xlsx.chm` to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above). After you have copied `xlsx.chm` to the `Docs` directory, you will be able to get online help by pressing F1 when the cursor is over a `xlsx.hwp` function in the Hollywood IDE.

On macOS copy the `xlsx` directory that is inside the `Docs` directory of `xlsx.hwp`'s distribution archive to the `Docs` directory which you can find in the same directory as the `Plugins` directory (see above).

On Linux copy the **xlsx** directory that is inside the **Docs** directory of **xlsx.hwp**'s distribution archive to the **Docs** directory of your Hollywood installation.

2 About `xlsx.hwp`

2.1 Credits

`xlsx.hwp` was written by Andreas Falkenhahn. This plugin was first designed as a proof-of-concept for Hollywood 9's new serialization interface and was later expanded into a full library for dealing with XLSX documents. Thanks have to go to Kenneth Troidal Balslev for his wonderful OpenXLSX on which this plugin is based.

If you want to contact me, you can either send an e-mail to andreas@airsoftsoftwair.de or use the contact form on <http://www.hollywood-mal.com>.

2.2 Frequently asked questions

This section covers some frequently asked questions. Please read them first before asking on the forum because your problem might have been covered here.

Q: Is there a Hollywood forum where I can get in touch with other users?

A: Yes, please check out the "Community" section of the official Hollywood Portal online at <http://www.hollywood-mal.com>.

Q: Where can I ask for help?

A: There's an active forum at <http://forums.hollywood-mal.com>. You're welcome to join it and ask your question there.

Q: I have found a bug.

A: Please post about it in the "Bugs" section of the forum.

2.3 Known issues

Here is a list of things that `xlsx.hwp` doesn't support yet or that may be confusing in some way:

- tbd

2.4 Future

Here are some things that are on my to do list:

- add support for cell formatting
- add support for embedding images

Don't hesitate to contact me if `xlsx.hwp` lacks a certain feature that is important for your project.

2.5 History

Please see the file `history.txt` for a complete change log of `xlsx.hwp`.

3 Usage

3.1 Interfaces

There are two ways of using this plugin: Either through the library interface or through the serialization interface. Using the plugin through the serialization interface is easier and very convenient but it comes at the expense of flexibility. Using the plugin through the library interface is a bit more difficult but offers full flexibility. Please see the next two chapters for a brief overview of the two different interfaces.

3.2 Library interface

The typical way of using this plugin is to deal with XLSX documents through the plugin's library interface. The library interface consists of a variety of functions that allow you to open and save XLSX documents, set and get cell values and other document and worksheet properties. For example, here is a script which creates an XLSX document that has 100 rows and 30 columns. The cell values will be set to a text string that contains each cell's column and row and the XLSX document will be saved as `test.xlsx`.

```
@REQUIRE "xlsx"
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
  For Local x = 1 to 30
    xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
  Next
Next
xlsx.Save(1)
xlsx.Close(1)
```

Alternatively, you can also use the plugin's serialization interface. This is easier because it only requires a single function call to convert Hollywood tables to XLSX documents and vice versa but you won't have fine-tuned control over everything as you have when using the library interface.

See the next chapter for more details on the plugin's serialization interface.

3.3 Serialization interface

If you don't want to use `xlsx.hwp`'s library interface (see above) for some reason, you can also use the plugin's serialization interface. This is easier to use because it only requires a single function call to convert Hollywood tables to XLSX documents and vice versa but you won't have fine-tuned control over everything as you have when using the library interface.

Access to the `xlsx.hwp`'s serialization interface is through Hollywood's `SerializeTable()` and `DeserializeTable()` functions, or, alternatively, through the `ReadTable()` and `WriteTable()` functions. By using the serialization interface, you can convert an XLSX document into a Hollywood table through just a single function call:

```
t = DeserializeTable(FileToString("test.xlsx"), "xlsx")
```

The code above will read all rows and columns from `test.xlsx` and store them in the Hollywood table `t`. You could then print all rows and columns in that table like this:

```
For Local y = 0 To ListItems(t) - 1
  For Local x = 0 To ListItems(t[y]) - 1
    DebugPrint(t[y][x])
  Next
Next
```

You could then simply change cell values by writing new values to the `t` table. For example, the following code changes the value of the cell in the 5th column and the 10th row to "Hello":

```
t[9][4] = "Hello"
```

When you're done with all modifications, you can simply convert your Hollywood table back into an XLSX document in just a single line like this:

```
StringToFile(SerializeTable(t, "xlsx"), "test2.xlsx")
```

The code above will convert the table `t` to an XLSX document using the `xlsx.hwp` plugin and save the XLSX document as `test2.xlsx`.

As you can see, the serialization interface is very easy to use but doesn't offer as much flexibility as the library interface which gives you fine-tuned control over many XLSX documents features.

4 Function reference

4.1 `xlsx.AddSheet`

NAME

`xlsx.AddSheet` – add a new worksheet

SYNOPSIS

```
xlsx.AddSheet(id, name$[, pos])
```

FUNCTION

This function will add a new worksheet to the XLSX document specified by `id`. The worksheet will be given the name specified by `name$`. The optional argument `pos` allows you to specify where the worksheet should be inserted in the XLSX document (worksheet positions start at 1). If `pos` is omitted or set to an out of range position, the new worksheet will be added as the last worksheet.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>name\$</code>	name for the new worksheet
<code>pos</code>	optional: desired insert position, starting from 1 for the first worksheet (defaults to 0, which means insert as the last worksheet)

4.2 `xlsx.CellRange`

NAME

`xlsx.CellRange` – iterate over cell range

SYNOPSIS

```
ref = xlsx.CellRange(id, startx, starty, endx, endy[, sheet])
ref = xlsx.CellRange(id, startcell$, endcell$[, sheet])
```

FUNCTION

This function can be used to iterate over a range of cells. You need to pass the cell where the iteration should start and the cell where it should end. `xlsx.CellRange()` will then return an iterator function which can be used together with Hollywood's generic `for` loop. The iterator function will return a reference to a cell that can be passed to all functions that deal with cells like `xlsx.SetCellValue()` or `xlsx.GetCellValue()`.

Passing a cell reference returned by `xlsx.CellRange()` to functions like `xlsx.SetCellValue()` or `xlsx.GetCellValue()` is much faster than addressing the cell using its column and row position or its alphanumerical identifier (e.g. "A1"). That's why it's recommended to use `xlsx.CellRange()` whenever you need to iterate over lots of cells, especially in huge XLSX documents with thousands of columns and rows.

`xlsx.CellRange()` supports two ways of specifying the start and cells: You can either specify the cells to use by passing their column (x) and row (y) positions in the `startx` / `starty` and `endx` / `endy` arguments. Those positions start from 1 for the first column

and row. Alternatively, you can also specify the cells by passing their alphanumerical references in the `startcell$` and `endcell$` parameters, e.g. "A10" for the first cell in the 10th row. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>startx</code>	column index of the start cell
<code>starty</code>	row index of the start cell
<code>endx</code>	column index of the end cell
<code>endy</code>	row index of the end cell
<code>startcell\$</code>	alphanumerical start cell reference (e.g. "A1"), only used when <code>startx</code> and <code>starty</code> are omitted
<code>endcell\$</code>	alphanumerical end cell reference (e.g. "Z100"), only used when <code>endx</code> and <code>endy</code> are omitted
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

<code>ref</code>	a cell reference
------------------	------------------

EXAMPLE

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For ref In xlsx.CellRange(1, 1, 1, cols, rows)
    DebugPrint((xlsx.GetCellValue(1, ref)))
Next
xlsx.Close(1)
```

The code above opens `test.xlsx` and prints the values of all cells.

4.3 xlsx.ClearCellFormula

NAME

`xlsx.ClearCellFormula` – clear cell formula

SYNOPSIS

```
xlsx.ClearCellFormula(id, x, y, f$[, sheet])
xlsx.ClearCellFormula(id, ref, f$[, sheet])
```

FUNCTION

This function clears the formula of the specified cell. After calling this function, `xlsx.HaveCellFormula()` will return `False`. There are two ways of specifying the cell

whose formula should be cleared: You can either specify the cell to use by passing the cell's column (x) and row (y) position in the **x** and **y** arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the **ref** parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional **sheet** parameter (starting from 1 for the first worksheet). If the **sheet** parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

id	identifier of the XLSX document to use
x	column index of the cell to use (starting from 1)
y	row index of the cell to use (starting from 1)
ref	cell reference (e.g. "A1" or an iterator state), only used when x and y are omitted
sheet	optional: index of the worksheet to use (defaults to the index of the default worksheet)

4.4 xlsx.ClearCellValue

NAME

`xlsx.ClearCellValue` – clear cell value

SYNOPSIS

```
xlsx.ClearCellValue(id, x, y[, sheet])
xlsx.ClearCellValue(id, ref[, sheet])
```

FUNCTION

This function clears the value of the specified cell. There are two ways of specifying the cell whose value should be cleared: You can either specify the cell to use by passing the cell's column (x) and row (y) position in the **x** and **y** arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the **ref** parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional **sheet** parameter (starting from 1 for the first worksheet). If the **sheet** parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

id	identifier of the XLSX document to use
x	column index of the cell to use (starting from 1)
y	row index of the cell to use (starting from 1)
ref	cell reference (e.g. "A1" or an iterator state), only used when x and y are omitted

sheet optional: index of the worksheet to use (defaults to the index of the default worksheet)

4.5 `xlsx.Close`

NAME

`xlsx.Close` – close XLSX document

SYNOPSIS

```
xlsx.Close(id)
```

FUNCTION

This function closes the specified XLSX document created by either `xlsx.Open()` or `xlsx.Create()`. Note that this function won't save any changes you have made to the XLSX document. If you want changes to be saved to the XLSX document, you must call `xlsx.Save()` or `xlsx.SaveAs()` first.

INPUTS

id identifier of the XLSX document to be closed

4.6 `xlsx.Create`

NAME

`xlsx.Create` – create an empty XLSX document

SYNOPSIS

```
[id] = xlsx.Create(id, filename$)
```

FUNCTION

This function will create an empty XLSX document containing a single worksheet named "Sheet1". Note that the XLSX document won't be saved to `filename$` until you call either `xlsx.Save()` or `xlsx.SaveAs()` on it.

INPUTS

id identifier for the XLSX document or `Nil` for auto id selection

filename\$
desired path and filename for the new document

RESULTS

id optional: identifier of the document; will only be returned when you pass `Nil` as argument 1 (see above)

EXAMPLE

```
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
  For Local x = 1 to 30
    xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
  Next
```

Next

```
xlsx.Save(1)
```

```
xlsx.Close(1)
```

The code above will create a new XLSX document and add 30 columns and 100 rows to it. The document will be saved as `test.xlsx`.

4.7 xlsx.DeleteProperty

NAME

`xlsx.DeleteProperty` – delete document property

SYNOPSIS

```
xlsx.DeleteProperty(id, prop)
```

FUNCTION

This function allows you to clear the document property specified by `prop`. The `prop` parameter must be one of the following special constants:

```
#XLSX_PROPERTY_TITLE  
#XLSX_PROPERTY_SUBJECT  
#XLSX_PROPERTY_CREATOR  
#XLSX_PROPERTY_KEYWORDS  
#XLSX_PROPERTY_DESCRIPTION  
#XLSX_PROPERTY_LASTMODIFIEDBY  
#XLSX_PROPERTY_LASTPRINTED  
#XLSX_PROPERTY_CREATIONDATE  
#XLSX_PROPERTY_MODIFICATIONDATE  
#XLSX_PROPERTY_CATEGORY  
#XLSX_PROPERTY_APPLICATION  
#XLSX_PROPERTY_DOCSECURITY  
#XLSX_PROPERTY_SCALECROP  
#XLSX_PROPERTY_MANAGER  
#XLSX_PROPERTY_COMPANY  
#XLSX_PROPERTY_LINKSUPTODATE  
#XLSX_PROPERTY_SHAREDDOC  
#XLSX_PROPERTY_HYPERLINKBASE  
#XLSX_PROPERTY_HYPERLINKSCHANGED  
#XLSX_PROPERTY_APPVERSION
```

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>prop</code>	property to clear (see above for possible values)

4.8 xlsx.DeleteSheet

NAME

`xlsx.DeleteSheet` – delete worksheet

SYNOPSIS

```
xlsx.DeleteSheet(id, idx)
```

FUNCTION

This function can be used to delete the worksheet at the position specified by `idx` from the XLSX document specified by `id`. Worksheet positions are counted from 1. Note that you cannot delete all worksheets from an XLSX document; there needs to be at least one worksheet in the XLSX document.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>idx</code>	index of the worksheet to delete (first worksheet is at index 1)

4.9 xlsx.GetCellFormula**NAME**

`xlsx.GetCellFormula` – get cell formula

SYNOPSIS

```
f$ = xlsx.GetCellFormula(id, x, y[, sheet])
f$ = xlsx.GetCellFormula(id, ref[, sheet])
```

FUNCTION

This function returns the formula of a certain cell. There are two ways of specifying the cell whose formula should be returned: You can either specify the cell to use by passing the cell's column (`x`) and row (`y`) position in the `x` and `y` arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the `ref` parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

Note that this function will fail in case the cell doesn't have a formula. You can use `xlsx.HaveCellFormula()` to check if the cell has got a formula.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>x</code>	column index of the cell to use (starting from 1)
<code>y</code>	row index of the cell to use (starting from 1)
<code>ref</code>	cell reference (e.g. "A1" or an iterator state), only used when <code>x</code> and <code>y</code> are omitted
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

<code>f\$</code>	the cell's formula
------------------	--------------------

4.10 `xlsx.GetCellReference`

NAME

`xlsx.GetCellReference` – get cell reference

SYNOPSIS

```
ref = xlsx.GetCellReference(id, x, y[, xyref, sheet])
ref = xlsx.GetCellReference(id, ref[, xyref, sheet])
```

FUNCTION

This function returns a reference to the specified cell, either as a column/row reference or an alphanumerical cell id. If the `xyref` parameter is set to `True`, the cell reference will be returned as a pair of column (x) and row (y) coordinates to the cell. If `xyref` is set to `False` (also the default), the cell reference will be returned as an alphanumerical string containing column and row identifier of the cell, e.g. "A1".

There are two ways of specifying the cell whose reference should be returned: You can either specify the cell to use by passing the cell's column (x) and row (y) position in the `x` and `y` arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the `ref` parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>x</code>	column index of the cell to use (starting from 1)
<code>y</code>	row index of the cell to use (starting from 1)
<code>ref</code>	cell reference (e.g. "A1" or an iterator state), only used when <code>x</code> and <code>y</code> are omitted
<code>xyref</code>	<code>True</code> if you want the reference as a pair of column/row coordinates or <code>False</code> if you want the reference as an alphanumerical string (defaults to <code>False</code>)
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

<code>ref</code>	the cell reference
------------------	--------------------

4.11 `xlsx.GetCellValue`

NAME

`xlsx.GetCellValue` – get cell value

SYNOPSIS

```
v, t = xlsx.GetCellValue(id, x, y[, sheet])
v, t = xlsx.GetCellValue(id, ref[, sheet])
```

FUNCTION

This function returns the value of a certain cell. There are two ways of specifying the cell whose value should be returned: You can either specify the cell to use by passing the cell's column (x) and row (y) position in the **x** and **y** arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the **ref** parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional **sheet** parameter (starting from 1 for the first worksheet). If the **sheet** parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

`xlsx.GetCellValue()` returns two values: The actual cell value in the first return value and the cell value type in the second return value. The return value type will be one of the following special constants:

- #INTEGER** An integer number.
- #DOUBLE** A floating point value.
- #STRING** A string value.
- #BOOLEAN** A boolean value (either **True** or **False**).
- #NIL** The cell is empty.
- #VOID** Indicates an invalid value, e.g. NaN or a logical error like division by zero.

Note that when trying to get the values of many cells it's usually much faster to use the `xlsx.CellRange()` function together with a generic for loop to iterate over the desired cells. This is especially recommended when dealing with large XLSX documents that have thousands of cells.

INPUTS

- id** identifier of the XLSX document to use
- x** column index of the cell to use (starting from 1)
- y** row index of the cell to use (starting from 1)
- ref** cell reference (e.g. "A1" or an iterator state), only used when **x** and **y** are omitted
- sheet** optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

- v** cell value
- t** type of the cell value (see above for possible types)

EXAMPLE

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For Local y = 1 To rows
  For Local x = 1 to cols
```

```
        DebugPrint((xlsx.GetCellValue(1, x, y)))
    Next
    DebugPrint("*****")
Next
xlsx.Close(1)
```

The code above opens `test.xlsx` and prints the values of all cells.

4.12 xlsx.GetColumnCount

NAME

`xlsx.GetColumnCount` – get number of worksheet columns

SYNOPSIS

```
cols = xlsx.GetColumnCount(id[, idx])
```

FUNCTION

This function returns the number of columns in the worksheet that is at the index specified by `idx` in the XLSX document. If the `idx` argument is omitted, the default worksheet set using `xlsx.SetDefaultSheet()` will be used. Worksheet indices start at 1 for the first worksheet.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>idx</code>	optional: index of worksheet to query (defaults to the index of the default worksheet)

RESULTS

<code>cols</code>	number of columns in the specified worksheet
-------------------	--

4.13 xlsx.GetColumnWidth

NAME

`xlsx.GetColumnWidth` – get column width

SYNOPSIS

```
width = xlsx.GetColumnWidth(id, col[, sheet])
```

FUNCTION

This function returns the width of the column specified in `col`. Column indices start at 1. The width is returned in font units of the normal display font and can be a fractional value. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>col</code>	column index to use (starting from 1)

sheet optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

width column width in font units

4.14 `xlsx.GetObjectType`

NAME

`xlsx.GetObjectType` – get XLSX document object type

SYNOPSIS

```
type = xlsx.GetObjectType()
```

FUNCTION

This function returns the object type used by XLSX documents opened using the `xlsx.Open()` or `xlsx.Create()` functions. You can then use this object type with functions from Hollywood's object library such as `GetAttribute()`, `SetObjectData()`, `GetObjectData()`, etc.

In particular, Hollywood's `GetAttribute()` function may be used to query certain properties of XLSX documents. The following attributes are currently supported by `GetAttribute()` for XLSX documents:

#XLSXATTRSHEETS:

Returns the number of sheets in the XLSX document.

INPUTS

none

RESULTS

type internal XLSX document type for use with Hollywood's object library

EXAMPLE

```
xlsx.Open(1, "test.xlsx")
XLSX_DOCUMENT = xlsx.GetObjectType()
numsheets = GetAttribute(XLSX_DOCUMENT, 1, #XLSXATTRSHEETS)
```

The code above opens `test.xlsx` and queries the number of sheets in the document via `GetAttribute()`.

4.15 `xlsx.GetProperty`

NAME

`xlsx.GetProperty` – get document property

SYNOPSIS

```
val$ = xlsx.GetProperty(id, prop)
```

FUNCTION

This function allows you to get the value of the document property specified by **prop**. The **prop** parameter must be one of the following special constants:

```
#XLSX_PROPERTY_TITLE
#XLSX_PROPERTY_SUBJECT
#XLSX_PROPERTY_CREATOR
#XLSX_PROPERTY_KEYWORDS
#XLSX_PROPERTY_DESCRIPTION
#XLSX_PROPERTY_LASTMODIFIEDBY
#XLSX_PROPERTY_LASTPRINTED
#XLSX_PROPERTY_CREATIONDATE
#XLSX_PROPERTY_MODIFICATIONDATE
#XLSX_PROPERTY_CATEGORY
#XLSX_PROPERTY_APPLICATION
#XLSX_PROPERTY_DOCSECURITY
#XLSX_PROPERTY_SCALECROP
#XLSX_PROPERTY_MANAGER
#XLSX_PROPERTY_COMPANY
#XLSX_PROPERTY_LINKSUPTODATE
#XLSX_PROPERTY_SHAREDDOC
#XLSX_PROPERTY_HYPERLINKBASE
#XLSX_PROPERTY_HYPERLINKSCHANGED
#XLSX_PROPERTY_APPVERSION
```

INPUTS

id identifier of the XLSX document to use

prop property to get (see above for possible values)

RESULTS

val\$ value of property

4.16 xlsx.GetRowCount**NAME**

`xlsx.GetRowCount` – get number of worksheet rows

SYNOPSIS

```
rows = xlsx.GetRowCount(id[, idx])
```

FUNCTION

This function returns the number of rows in the worksheet that is at the index specified by **idx** in the XLSX document. If the **idx** argument is omitted, the default worksheet set using `xlsx.SetDefaultSheet()` will be used. Worksheet indices start at 1 for the first worksheet.

INPUTS

id identifier of the XLSX document to use

`idx` optional: index of worksheet to query (defaults to the index of the default worksheet)

RESULTS

`rows` number of rows in the specified worksheet

4.17 `xlsx.GetRowHeight`

NAME

`xlsx.GetRowHeight` – get row height

SYNOPSIS

```
height = xlsx.GetRowHeight(id, row[, sheet])
```

FUNCTION

This function returns the height of the row specified in `row`. Row indices start at 1. The height is returned in font units of the normal display font and can be a fractional value. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

`id` identifier of the XLSX document to use

`row` row index to use (starting from 1)

`sheet` optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

`height` row height in font units

4.18 `xlsx.GetSheetCount`

NAME

`xlsx.GetSheetCount` – get number of worksheets in document

SYNOPSIS

```
n = xlsx.GetSheetCount(id)
```

FUNCTION

This function returns the number of worksheets in the XLSX document specified by `id`. Since there cannot be XLSX documents without any worksheets, the return value will always be at least 1.

INPUTS

`id` identifier of the XLSX document to use

RESULTS

`n` number of workshhets in the XLSX document

4.19 `xlsx.GetSheetIndex`

NAME

`xlsx.GetSheetIndex` – get worksheet index

SYNOPSIS

```
idx = xlsx.GetSheetIndex(id, name$)
```

FUNCTION

This function returns the position of the worksheet specified by `name$` in the XLSX document specified by `id`. Worksheet indices start at 1. If the worksheet can't be found in the XLSX document, 0 will be returned.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>name\$</code>	name of the worksheet whose position should be retrieved

RESULTS

<code>idx</code>	position of the worksheet or 0 if not found
------------------	---

4.20 `xlsx.GetSheetName`

NAME

`xlsx.GetSheetName` – get worksheet name

SYNOPSIS

```
name$ = xlsx.GetSheetName(id, idx)
```

FUNCTION

This function returns the name of the worksheet at the position specified by `idx`. Worksheet indices start at 1.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>idx</code>	position of the worksheet whose name should be retrieved

RESULTS

<code>name\$</code>	name of the worksheet at the specified position
---------------------	---

4.21 `xlsx.GetSheetType`

NAME

`xlsx.GetSheetType` – get worksheet type

SYNOPSIS

```
type = xlsx.GetSheetType(id, idx)
```

FUNCTION

This function returns the type of the worksheet at the position specified by `idx`. Worksheet indices start at 1. The return value will be one of the following constants:

`#XLSX_SHEETTYPE_WORKSHEET`
A normal worksheet.

`#XLSX_SHEETTYPE_CHARTSHEET`
A chart worksheet.

`#XLSX_SHEETTYPE_DIALOGSHEET`
A dialog worksheet.

`#XLSX_SHEETTYPE_MACROSHEET`
A macro worksheet.

INPUTS

`id` identifier of the XLSX document to use

`idx` position of the worksheet whose type should be retrieved (starting from 1)

RESULTS

`type` type of the worksheet at the specified position

4.22 `xlsx.GetSheetVisibility`

NAME

`xlsx.GetSheetVisibility` – get sheet visibility

SYNOPSIS

```
vis = xlsx.GetSheetVisibility(id[, sheet])
```

FUNCTION

This function can be used to get the visibility state of the worksheet specified by the `sheet` parameter. The return value will be one of the following special constants:

`#XLSX_VISIBILITY_VISIBLE`
The sheet is visible.

`#XLSX_VISIBILITY_HIDDEN`
The sheet is hidden but can be unhidden by users opening the XLSX file in a spreadsheet app.

`#XLSX_VISIBILITY_VERYHIDDEN`
The sheet is hidden and can't be unhidden by users opening the XLSX file in a spreadsheet app.

The `sheet` parameter is optional. If it is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

`id` identifier of the XLSX document to use

sheet optional: index of the worksheet to (un)select (defaults to the index of the default worksheet)

RESULTS

vis sheet visibility state (see above for possible values)

4.23 `xlsx.HaveCellFormula`

NAME

`xlsx.HaveCellFormula` – check if cell has a formula

SYNOPSIS

```
bool = xlsx.HaveCellFormula(id, x, y[, sheet])
bool = xlsx.HaveCellFormula(id, ref[, sheet])
```

FUNCTION

This function returns **True** if the specified cell has a formula, otherwise **False** is returned. There are two ways of specifying the cell you want to check: You can either specify the cell to use by passing the cell's column (**x**) and row (**y**) position in the **x** and **y** arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the **ref** parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional **sheet** parameter (starting from 1 for the first worksheet). If the **sheet** parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

id identifier of the XLSX document to use

x column index of the cell to use (starting from 1)

y row index of the cell to use (starting from 1)

ref cell reference (e.g. "A1" or an iterator state), only used when **x** and **y** are omitted

sheet optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

bool **True** if the cell has a formula, **False** otherwise

4.24 `xlsx.HideColumn`

NAME

`xlsx.HideColumn` – show or hide a column

SYNOPSIS

```
xlsx.HideColumn(id, col, hidden[, sheet])
```

FUNCTION

This function can be used to show or hide the column specified by `col`. Column indices start at 1. The `hidden` argument must be set to `True` to hide the column or `False` to unhide it. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>col</code>	column index to use (starting from 1)
<code>hidden</code>	<code>True</code> to hide the column, <code>False</code> to show it
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

4.25 xlsx.HideRow**NAME**

`xlsx.HideRow` – show or hide a row

SYNOPSIS

```
xlsx.HideRow(id, row, hidden[, sheet])
```

FUNCTION

This function can be used to show or hide the row specified by `row`. Row indices start at 1. The `hidden` argument must be set to `True` to hide the row or `False` to unhide it. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>row</code>	row index to use (starting from 1 for the first row)
<code>hidden</code>	<code>True</code> to hide the row, <code>False</code> to show it
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

4.26 xlsx.IsColumnHidden**NAME**

`xlsx.IsColumnHidden` – get column visibility state

SYNOPSIS

```
hidden = xlsx.IsColumnHidden(id, col[, sheet])
```

FUNCTION

This function returns **True** if the column at index `col` is currently hidden or **False** if it is visible. Column indices start at 1. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>col</code>	column index to use (starting from 1)
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

<code>hidden</code>	True if the column is hidden, False otherwise
---------------------	---

4.27 `xlsx.IsRowHidden`

NAME

`xlsx.IsRowHidden` – get row visibility state

SYNOPSIS

```
hidden = xlsx.IsRowHidden(id, row[, sheet])
```

FUNCTION

This function returns **True** if the row at index `row` is currently hidden or **False** if it is visible. Row indices start at 1. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>row</code>	row index to use (starting from 1)
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

<code>hidden</code>	True if the row is hidden, False otherwise
---------------------	--

4.28 `xlsx.IsSheetActive`

NAME

`xlsx.IsSheetActive` – check if sheet is active

SYNOPSIS

```
active = xlsx.IsSheetActive(id[, sheet])
```

FUNCTION

This function returns **True** if the worksheet specified by **sheet** is active, **False** otherwise. The **sheet** parameter is optional. If it is omitted, the worksheet set by **xlsx.SetDefaultSheet()** will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

id identifier of the XLSX document to use

sheet optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

active **True** if the sheet is active, **False** otherwise

4.29 xlsx.IsSheetSelected**NAME**

xlsx.IsSheetSelected – check if sheet is selected

SYNOPSIS

sel = **xlsx.IsSheetSelected**(**id**[, **sheet**])

FUNCTION

This function returns **True** if the worksheet specified by **sheet** is selected, **False** otherwise. The **sheet** parameter is optional. If it is omitted, the worksheet set by **xlsx.SetDefaultSheet()** will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

id identifier of the XLSX document to use

sheet optional: index of the worksheet to use (defaults to the index of the default worksheet)

RESULTS

sel **True** if the sheet is selected, **False** otherwise

4.30 xlsx.MoveSheet**NAME**

xlsx.MoveSheet – change worksheet position

SYNOPSIS

xlsx.MoveSheet(**id**, **idx**, **newpos**)

FUNCTION

This function can be used to change the position of the worksheet at **idx** to the position specified by **newpos**. Worksheet indices are counted from 1.

INPUTS

id identifier of the XLSX document to use

`idx` worksheet whose position should be changed (starting from 1)
`newpos` desired new position for the worksheet (starting from 1)

4.31 `xlsx.Open`

NAME

`xlsx.Open` – open an XLSX document for reading and/or writing

SYNOPSIS

```
[id] = xlsx.Open(id, filename$)
```

FUNCTION

This function attempts to open the XLSX document specified by `filename$` and assigns `id` to it. If you pass `Nil` in `id`, `xlsx.Open()` will automatically choose a vacant identifier and return it. The file specified in `filename$` must exist or this function will fail. If you want to create a new xlsx document, use the `xlsx.Create()` function.

Although `xlsx.hwp` will automatically close all open XLSX documents when it quits, it is strongly advised that you close an open XLSX document when you are done with it using the `xlsx.Close()` function because otherwise you are wasting resources.

Note that `xlsx.Open()` will create a standard Hollywood object which can also be used with functions from Hollywood's object library such as `GetAttribute()`, `SetObjectData()`, `GetObjectData()`, etc. See [Section 4.14 \[xlsx.GetObjectType\]](#), [page 18](#), for details.

INPUTS

`id` identifier for the XLSX document or `Nil` for auto id selection
`filename$` name of the file to open

RESULTS

`id` optional: identifier of the document; will only be returned when you pass `Nil` as argument 1 (see above)

EXAMPLE

```
xlsx.Open(1, "test.xlsx")
cols = xlsx.GetColumnCount(1)
rows = xlsx.GetRowCount(1)
For Local y = 1 To rows
  For Local x = 1 to cols
    DebugPrint((xlsx.GetCellValue(1, x, y)))
  Next
  DebugPrint("*****")
Next
xlsx.Close(1)
```

The code above opens `test.xlsx` and prints the values of all cells.

4.32 `xlsx.Save`

NAME

`xlsx.Save` – save XLSX document

SYNOPSIS

```
xlsx.Save(id)
```

FUNCTION

This function saves the XLSX document specified by `id` to the file that was specified when opening the XLSX document using `xlsx.Open()` or creating it using `xlsx.Create()`. If you want to save the XLSX document to a different location, use `xlsx.SaveAs()`.

Note that this function won't close the XLSX document. You still need to call `xlsx.Close()` to free all resources associated with the XLSX document.

INPUTS

`id` identifier for the XLSX document

4.33 `xlsx.SaveAs`

NAME

`xlsx.SaveAs` – save XLSX document to new location

SYNOPSIS

```
xlsx.SaveAs(id, filename$)
```

FUNCTION

This function saves the XLSX document specified by `id` to the location specified by `filename$`. If you don't want to save the XLSX document to a new location, use `xlsx.Save()` instead.

Note that this function won't close the XLSX document. You still need to call `xlsx.Close()` to free all resources associated with the XLSX document.

INPUTS

`id` identifier for the XLSX document

`filename$`
 desired save location for the XLSX document

4.34 `xlsx.SetCellFormula`

NAME

`xlsx.SetCellFormula` – set cell formula

SYNOPSIS

```
xlsx.SetCellFormula(id, x, y, f$[, sheet])  
xlsx.SetCellFormula(id, ref, f$[, sheet])
```

FUNCTION

This function sets the formula of the specified cell to the one specified in `f$`. After calling this function, `xlsx.HaveCellFormula()` will return `True`. There are two ways of specifying the cell whose value should be set: You can either specify the cell to use by passing the cell's column (`x`) and row (`y`) position in the `x` and `y` arguments. Those positions start from 1 for the first column and row. Alternatively, you can also specify the cell by passing its reference in the `ref` parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

Note that the formula must be specified without the equal sign, e.g. you have to use "A1+A2" instead of "=A1+A2". Also note that the XLSX plugin won't compute the result of the formula, i.e. you can't expect `xlsx.GetCellValue()` to get the computation result after setting a cell formula. To have formula values computed, you need to open the XLSX document in Excel or LibreOffice's Calc and save it.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>x</code>	column index of the cell to use (starting from 1)
<code>y</code>	row index of the cell to use (starting from 1)
<code>ref</code>	cell reference (e.g. "A1" or an iterator state), only used when <code>x</code> and <code>y</code> are omitted
<code>f\$</code>	desired cell formula (don't include the equal sign here)
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

EXAMPLE

```
xlsx.SetCellFormula(1, "A3", "A1+A2")
```

The code above sets cell A3 to the sum of cells A1+A2.

4.35 xlsx.SetCellValue**NAME**

`xlsx.SetCellValue` – set cell value

SYNOPSIS

```
xlsx.SetCellValue(id, x, y, val[, type, sheet])
xlsx.SetCellValue(id, ref, val[, type, sheet])
```

FUNCTION

This function sets the value of the specified cell to the value specified in `val`. There are two ways of specifying the cell whose value should be set: You can either specify the cell to use by passing the cell's column (`x`) and row (`y`) position in the `x` and `y` arguments. Those positions start from 1 for the first column and row. Alternatively, you

can also specify the cell by passing its reference in the `ref` parameter. This can either be a string, e.g. "A10" for the first cell in the 10th row, or an iterator state returned by the `xlsx.CellRange()` function. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

Optionally, you can also specify the value type in the `type` argument. Normally, this is not necessary since `xlsx.SetCellValue()` will determine the value type based on the type of the argument you pass in `val` but since Hollywood doesn't distinguish between boolean, integer, and floating point values it might be necessary to pass the `type` parameter in order to make sure the cell is set to the desired type. The `type` parameter can be one of the following special constants:

`#INTEGER` An integer value.
`#DOUBLE` A floating point value.
`#STRING` A string value.
`#BOOLEAN` A boolean value (either `True` or `False`).
`#NIL` This is a special type that will clear the cell.

Note that when trying to set the values of many cells it's usually much faster to use the `xlsx.CellRange()` function together with a generic for loop to iterate over the desired cells. This is especially recommended when dealing with large XLSX documents that have thousands of cells.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>x</code>	column index of the cell to use (starting from 1)
<code>y</code>	row index of the cell to use (starting from 1)
<code>ref</code>	cell reference (e.g. "A1" or an iterator state), only used when <code>x</code> and <code>y</code> are omitted
<code>val</code>	desired cell value
<code>type</code>	optional: type of the value (see above for possible constants)
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

EXAMPLE

```
xlsx.Create(1, "test.xlsx")
For Local y = 1 To 100
  For Local x = 1 to 30
    xlsx.SetCellValue(1, x, y, "Cell " .. x .. "/" .. y)
  Next
Next
xlsx.Save(1)
xlsx.Close(1)
```

The code above will create a new XLSX document and add 30 columns and 100 rows to it. The document will be saved as `test.xlsx`.

4.36 `xlsx.SetColumnWidth`

NAME

`xlsx.SetColumnWidth` – set column width

SYNOPSIS

```
xlsx.SetColumnWidth(id, col, width[, sheet])
```

FUNCTION

This function sets the width of the column specified in `col` to `width`. Column indices start at 1. The width is specified in font units of the normal display font and can be a fractional value. Optionally, you can also pass the index of the worksheet to use in the optional `sheet` parameter (starting from 1 for the first worksheet). If the `sheet` parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>col</code>	column index to use (starting from 1)
<code>width</code>	desired column width in font units
<code>sheet</code>	optional: index of the worksheet to use (defaults to the index of the default worksheet)

EXAMPLE

```
xlsx.SetColumnWidth(1, 1, 8.43)
```

The code above sets the width of the first column to 8.43.

4.37 `xlsx.SetDefaultSheet`

NAME

`xlsx.SetDefaultSheet` – set default worksheet

SYNOPSIS

```
xlsx.SetDefaultSheet(id, idx)
```

FUNCTION

This function can be used to set the default worksheet for the XLSX document specified by `id`. An XLSX document's default worksheet is the worksheet that is to be used in case no worksheet is explicitly specified when calling functions like `xlsx.SetCellValue()` or `xlsx.GetCellValue()`. You need to pass the position of the desired default worksheet in the `idx` argument. Worksheet indices start at 1 for the first worksheet.

By default, the first worksheet in the XLSX document is the default worksheet.

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>idx</code>	position of the worksheet that should be made the default (starting from 1)

4.38 `xlsx.SetProperty`

NAME

`xlsx.SetProperty` – set document property

SYNOPSIS

`xlsx.SetProperty(id, prop, val$)`

FUNCTION

This function allows you to set the document property specified by `prop` to the value specified by `val$`. The `prop` parameter must be one of the following special constants:

```
#XLSX_PROPERTY_TITLE
#XLSX_PROPERTY_SUBJECT
#XLSX_PROPERTY_CREATOR
#XLSX_PROPERTY_KEYWORDS
#XLSX_PROPERTY_DESCRIPTION
#XLSX_PROPERTY_LASTMODIFIEDBY
#XLSX_PROPERTY_LASTPRINTED
#XLSX_PROPERTY_CREATIONDATE
#XLSX_PROPERTY_MODIFICATIONDATE
#XLSX_PROPERTY_CATEGORY
#XLSX_PROPERTY_APPLICATION
#XLSX_PROPERTY_DOCSECURITY
#XLSX_PROPERTY_SCALECROP
#XLSX_PROPERTY_MANAGER
#XLSX_PROPERTY_COMPANY
#XLSX_PROPERTY_LINKSUPTODATE
#XLSX_PROPERTY_SHAREDDOC
#XLSX_PROPERTY_HYPERLINKBASE
#XLSX_PROPERTY_HYPERLINKSCHANGED
#XLSX_PROPERTY_APPVERSION
```

INPUTS

<code>id</code>	identifier of the XLSX document to use
<code>prop</code>	property to set (see above for possible values)
<code>val\$</code>	desired value for property

4.39 `xlsx.SetRowHeight`

NAME

`xlsx.SetRowHeight` – set row height

SYNOPSIS

`xlsx.SetRowHeight(id, row, height[, sheet])`

FUNCTION

This function sets the height of the row specified in `row` to `height`. Row indices start at 1. The height is specified in font units of the normal display font and can be a fractional

value. Optionally, you can also pass the index of the worksheet to use in the optional **sheet** parameter (starting from 1 for the first worksheet). If the **sheet** parameter is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used.

INPUTS

id	identifier of the XLSX document to use
row	row index to use (starting from 1)
height	desired row height in font units
sheet	optional: index of the worksheet to use (defaults to the index of the default worksheet)

EXAMPLE

```
xlsx.SetRowHeight(1, 1, 12.75)
```

The code above sets the height of the first row to 12.75.

4.40 xlsx.SetSheetActive

NAME

`xlsx.SetSheetActive` – make sheet active

SYNOPSIS

```
xlsx.SetSheetActive(id[, sheet])
```

FUNCTION

This function makes the worksheet specified by the **sheet** parameter the active one. The **sheet** parameter is optional. If it is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

id	identifier of the XLSX document to use
sheet	optional: index of the worksheet to make the active one (defaults to the index of the default worksheet)

4.41 xlsx.SetSheetName

NAME

`xlsx.SetSheetName` – set worksheet name

SYNOPSIS

```
xlsx.SetSheetName(id, idx, name$)
```

FUNCTION

This function sets the name of the worksheet at the position specified by **idx** to the string passed in **name\$**. Worksheet positions start at 1.

INPUTS

id	identifier of the XLSX document to use
-----------	--

`idx` position of the worksheet whose name should be set
`name$` desired worksheet name

4.42 `xlsx.SetSheetSelected`

NAME

`xlsx.SetSheetSelected` – select or unselect sheet

SYNOPSIS

`xlsx.SetSheetSelected(id, sel[, sheet])`

FUNCTION

This function can be used to select or unselect the worksheet specified by the `sheet` parameter. If the `sel` parameter is set to `True`, the sheet will be selected, otherwise it will be unselected. The `sheet` parameter is optional. If it is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

`id` identifier of the XLSX document to use
`sel` `True` to select the sheet, `False` to unselect it
`sheet` optional: index of the worksheet to (un)select (defaults to the index of the default worksheet)

4.43 `xlsx.SetSheetVisibility`

NAME

`xlsx.SetSheetVisibility` – set sheet visibility

SYNOPSIS

`xlsx.SetSheetVisibility(id, vis[, sheet])`

FUNCTION

This function can be used to set the visibility state of the worksheet specified by the `sheet` parameter. The `vis` parameter must be one of the following special constants:

`#XLSX_VISIBILITY_VISIBLE`

The sheet is visible.

`#XLSX_VISIBILITY_HIDDEN`

The sheet is hidden but can be unhidden by users opening the XLSX file in a spreadsheet app.

`#XLSX_VISIBILITY_VERYHIDDEN`

The sheet is hidden and can't be unhidden by users opening the XLSX file in a spreadsheet app.

The `sheet` parameter is optional. If it is omitted, the worksheet set by `xlsx.SetDefaultSheet()` will be used. Sheet indices start at 1 for the first worksheet.

INPUTS

id	identifier of the XLSX document to use
vis	desired sheet visibility state (see above for possible values)
sheet	optional: index of the worksheet to (un)select (defaults to the index of the default worksheet)

4.44 xlsx.UseSharedStrings**NAME**

`xlsx.UseSharedStrings` – toggle shared string mode

SYNOPSIS

`xlsx.UseSharedStrings(on)`

FUNCTION

This function allows you to control whether or not strings assigned to cells should be stored in a global shared string table in the XLSX or whether they should be embedded individually in the cell nodes. It's typically more efficient to use a global shared string table because identical strings only need to be stored once in that table which will decrease the file size in case there are many identical strings. If for some reason you don't want to use a global shared string table, you can use `xlsx.UseSharedStrings()` to disable this functionality by passing **False** in the `on` parameter.

Note that this function will only be effective when adding new strings to a document. If you're opening a document that uses shared strings and save it again, it will still keep its shared strings, even if you have disabled shared string mode using this function. `xlsx.UseSharedStrings()` will only affect new strings added to the document.

By default the global shared string table is enabled.

INPUTS

on **True** to enable the global shared string table, **False** to disable it

Appendix A Licenses

A.1 OpenXLSX license

Copyright (c) 2020, Kenneth Trolldal Balslev All rights reserved.

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

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

A.2 pugixml license

Copyright (c) 2006-2022 Arseny Kapoulkine

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

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A.3 MiniZ license

Copyright 2013-2014 RAD Game Tools and Valve Software

Copyright 2010-2014 Rich Geldreich and Tenacious Software LLC

All Rights Reserved.

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

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

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Index

<code>xlsx.AddSheet</code>	9	<code>xlsx.HaveCellFormula</code>	23
<code>xlsx.CellRange</code>	9	<code>xlsx.HideColumn</code>	23
<code>xlsx.ClearCellFormula</code>	10	<code>xlsx.HideRow</code>	24
<code>xlsx.ClearCellValue</code>	11	<code>xlsx.IsColumnHidden</code>	24
<code>xlsx.Close</code>	12	<code>xlsx.IsRowHidden</code>	25
<code>xlsx.Create</code>	12	<code>xlsx.IsSheetActive</code>	25
<code>xlsx.DeleteProperty</code>	13	<code>xlsx.IsSheetSelected</code>	26
<code>xlsx.DeleteSheet</code>	13	<code>xlsx.MoveSheet</code>	26
<code>xlsx.GetCellFormula</code>	14	<code>xlsx.Open</code>	27
<code>xlsx.GetCellReference</code>	14	<code>xlsx.Save</code>	27
<code>xlsx.GetCellValue</code>	15	<code>xlsx.SaveAs</code>	28
<code>xlsx.GetColumnCount</code>	17	<code>xlsx.SetCellFormula</code>	28
<code>xlsx.GetColumnWidth</code>	17	<code>xlsx.SetCellValue</code>	29
<code>xlsx.GetObjectType</code>	18	<code>xlsx.SetColumnWidth</code>	30
<code>xlsx.GetProperty</code>	18	<code>xlsx.SetDefaultSheet</code>	31
<code>xlsx.GetRowCount</code>	19	<code>xlsx.SetProperty</code>	31
<code>xlsx.GetRowHeight</code>	20	<code>xlsx.SetRowHeight</code>	32
<code>xlsx.GetSheetCount</code>	20	<code>xlsx.SetSheetActive</code>	33
<code>xlsx.GetSheetIndex</code>	20	<code>xlsx.SetSheetName</code>	33
<code>xlsx.GetSheetName</code>	21	<code>xlsx.SetSheetSelected</code>	34
<code>xlsx.GetSheetType</code>	21	<code>xlsx.SetSheetVisibility</code>	34
<code>xlsx.GetSheetVisibility</code>	22	<code>xlsx.UseSharedStrings</code>	35