Tim Tognazzini

github.com/ttognazzini/#$xlsx

Abstract

The #$XLSX Service Program allows RPG developers to write Excel files out directly in their RPG programs. The service package allows for most options including styling and print options to be added into the Excel file.

#$XLSX Service Program

Create formatted Excel files in RPG

# Introduction

The #$XLSX service program allows RPG developers to create pretty Excel files directly in their RPG code.

The program handles most the formatting options at all levels. These include the file properties, cell formatting, print options and many more.

The program is written in RPG however it uses a ported version of the C package ZLib to handle the Zip file work. This process is encapsulated in the #$ZIP service program include with this package. All copyright information for the ZLib package is included in the source code for it. The author of the #$XLSX service program takes no responsibility or credit for any of the software provided in the ZLib package. The ZLib functions used in #$XLSX have been prototyped and added to the service program #$Zip which is included in this package and documented in this document.

#$XLSX is a standalone program, it does not rely on JAVA and never starts a virtual machine.

To get started with using the #XLSX service program please review the [Example](#_Examples_and_Learning) section of this document. It walks you through most features of the package one step at a time. This document also contains detailed documentation for each function. So, after you get the hang of using the service program you can get details on all options by reviewing the procedures individually.

Both this document and the service program are a work in process. You will see many things labeled as TODO. These are things that need to be completed. The program works as is, but more features still need to be documented and added to the service program. This document includes a section on what still needs to be done, it will be updated as the author gets ideas for finds issues.

Contents

[Chapter 1 Introduction 1-1](#_Toc70689505)

[Chapter 2 Work that still needs to be done 2-5](#_Toc70689506)

[Complete a unit testing setup 2-5](#_Toc70689507)

[Future Update Ideas 2-6](#_Toc70689508)

[Chapter 3 Examples and Learning (How to) 3-7](#_Toc70689509)

[Terminology 3-7](#_Toc70689510)

[Example Source 3-7](#_Toc70689511)

[Example 1 - Hello World 3-7](#_Toc70689512)

[Ordering 3-9](#_Toc70689513)

[Example 2 – Adding more data and multiple rows 3-9](#_Toc70689514)

[Example 3 – Formatting cells 3-13](#_Toc70689515)

[Chapter 4 Procedure Details 4-19](#_Toc70689516)

[#$XLSXOpen = Create a New Workbook 4-19](#_Toc70689517)

[OutputName – tells #$XLSX what to name the output file 4-19](#_Toc70689518)

[Debug – Generates the xml with tabs and end of line characters. 4-19](#_Toc70689519)

[Author (Creator) – Sets the author of the file 4-20](#_Toc70689520)

[Company – Set the company property for the spread sheet 4-20](#_Toc70689521)

[Title – sets the title property for the Excel file 4-20](#_Toc70689522)

[Subject – sets the subject property for the file 4-20](#_Toc70689523)

[Manager – sets the manager property for the file 4-20](#_Toc70689524)

[Category – sets the category property for the file 4-20](#_Toc70689525)

[Status – sets the status property for the file 4-20](#_Toc70689526)

[Comments – Adds comments to the file 4-21](#_Toc70689527)

[HyperLinkBase – Adds the Hyper Link Base property 4-21](#_Toc70689528)

[Tags (Keywords) – Adds tags or keywords to the file 4-21](#_Toc70689529)

[CustomProperty – Adds custom properties to the file 4-21](#_Toc70689530)

[Buffering – Sets the file buffering option 4-22](#_Toc70689531)

[#$XL2Style = Create a style 4-23](#_Toc70689532)

[Name – The style name 4-23](#_Toc70689533)

[BoldWeight – Sets font to bold or not 4-23](#_Toc70689534)

[PointSize – Font Point Size 4-24](#_Toc70689535)

[Font – Font Name 4-24](#_Toc70689536)

[Color – Font Color 4-24](#_Toc70689537)

[Italic – Set cell to italic 4-24](#_Toc70689538)

[StrikeOut – Set Cell to have a line through the text 4-25](#_Toc70689539)

[TypeOffset – Set text to superscript or subscript. 4-25](#_Toc70689540)

[Underline – Sets underline style 4-26](#_Toc70689541)

[Alignment/VerticalAlignment – Sets Horizontal/Vertical Alignment 4-26](#_Toc70689542)

[DataFormat – Sets the Data formatting option 4-27](#_Toc70689543)

[Borders 4-29](#_Toc70689544)

[WrapText – Flags the cell to wrap text 4-30](#_Toc70689545)

[FillForegroundColor – Set the fill foreground color 4-31](#_Toc70689546)

[FillBackgroundColor – Sets the fill back ground color 4-31](#_Toc70689547)

[FillPattern – Sets the fill pattern 4-32](#_Toc70689548)

[#$XLSXWkSh = Create a New Worksheet 4-32](#_Toc70689549)

[Header/Footer – Set page headers or footers 4-33](#_Toc70689550)

[DifferentOddAndEvenPages – Set headings and footing to be different for odd and even pages 4-36](#_Toc70689551)

[DifferentFirstpage – Set headings and footing to be different for the first page 4-36](#_Toc70689552)

[Margins – setting page margins 4-37](#_Toc70689553)

[SheetName 4-39](#_Toc70689554)

[AutoBreaks 4-39](#_Toc70689555)

[ColumnWidth 4-39](#_Toc70689556)

[Zoom 4-39](#_Toc70689557)

[Repeating Columns/Row 4-39](#_Toc70689558)

[Freeze Columns 4-39](#_Toc70689559)

[Freeze Rows 4-40](#_Toc70689560)

[Print Orientation 4-40](#_Toc70689561)

[Print Scale 4-40](#_Toc70689562)

[Filter 4-40](#_Toc70689563)

[Paper Size 4-41](#_Toc70689564)

[Print area 4-41](#_Toc70689565)

[Print Quality 4-41](#_Toc70689566)

[Print Center Vertically 4-41](#_Toc70689567)

[Print Center Horizontally 4-41](#_Toc70689568)

[Print Page Order 4-41](#_Toc70689569)

[#$XLSXChar = add a character field to an Excel File 4-41](#_Toc70689570)

[#$XLSXNumr = add a value field to Excel file 4-41](#_Toc70689571)

[#$XLSXDate = add a date field to Excel file 4-42](#_Toc70689572)

[#$XLSXYYMD = add a date field to Excel file 4-42](#_Toc70689573)

[#$XLSXMDY = add a date field to Excel file 4-43](#_Toc70689574)

[#$XLSXMDYY = add a date field to Excel file 4-43](#_Toc70689575)

[#$XL2Null = Skips a column, but does not create the cell 4-44](#_Toc70689576)

[#$XLSXForm = add a formula field to a Excel File 4-44](#_Toc70689577)

[#$XLSXNext = Start a new row 4-44](#_Toc70689578)

[#$XLSXCell – Returns a Cell or Range Relative to the Current Cell 4-44](#_Toc70689579)

[Optional Parameters 4-45](#_Toc70689580)

[Examples: 4-49](#_Toc70689581)

[#$XLSXC = Shortcut for #$XSLSCell 4-51](#_Toc70689582)

[#$XLSXClose = Close the open Excel File 4-52](#_Toc70689583)

[Chapter 5 #$ZIP Service Program 5-53](#_Toc70689584)

[Example Programs 5-53](#_Toc70689585)

[Ordering 5-53](#_Toc70689586)

[#$ZipOpen – Open the Zip file 5-54](#_Toc70689587)

[#$ZipFile - Add file inside a zip file 5-55](#_Toc70689588)

[#$ZipWrite - Write data into a file inside a zip file 5-55](#_Toc70689589)

[#ZipWriteB - Same as write but for a buffer 5-55](#_Toc70689590)

[#$ZipClose - Close an open zip file archive 5-56](#_Toc70689591)

[Chapter 6 Appendix 1 – Colors 6-57](#_Toc70689592)

[Chapter 7 Appendix 2 – Theme Color constants 7-59](#_Toc70689593)

[Chapter 8 Appendix 3 - Data Format rules 8-61](#_Toc70689594)

[Text and spacing 8-63](#_Toc70689595)

[Display both text and numbers 8-63](#_Toc70689596)

[Include a section for text entry 8-64](#_Toc70689597)

[Add spaces 8-64](#_Toc70689598)

[Repeat characters 8-64](#_Toc70689599)

[Decimal places, spaces, colors, and conditions 8-64](#_Toc70689600)

[Include decimal places and significant digits 8-64](#_Toc70689601)

[Display a thousands separator 8-65](#_Toc70689602)

[Specify colors 8-65](#_Toc70689603)

[Specify conditions 8-65](#_Toc70689604)

[Currency, percentages, and scientific notation 8-66](#_Toc70689605)

[Include currency symbols 8-66](#_Toc70689606)

[Display percentages 8-66](#_Toc70689607)

[Display scientific notations 8-66](#_Toc70689608)

[Dates and times 8-66](#_Toc70689609)

[Display days, months, and years 8-66](#_Toc70689610)

[Month versus minutes 8-66](#_Toc70689611)

[Display hours, minutes, and seconds 8-66](#_Toc70689612)

[Minutes versus month 8-67](#_Toc70689613)

[AM and PM 8-67](#_Toc70689614)

[Illegal date and time values 8-67](#_Toc70689615)

# Work that still needs to be done

Add validation messages to all arguments. The #$AddStyle is done, do it to the rest of them.

Finish the example stuff. It needs to walk through how to do everything each with example programs.

Finish this document, search for anything with a TODO and get it done. Then review the document for completeness and grammar/spelling.

There is a strike through variable in the header DS it does not seem to be getting used, figure out if it is needed.

Figure out if there is a way to automatically highlight a group of cells, also to position to certain cell or worksheet. Looks like this is done in the sheet in the SheetViews section:

<sheetView tabSelected="1" workbookViewId="0">

<selection activeCell="E4" sqref="E4"/>

</sheetView>

Figure out how to add hyperlinks.

Figure out how to do a forced page break. We need to be able to add page breaks to NVT103PI on item changes.

Figure out if styles can be added any time before they are used on a cell. I don’t remember why I started saying that they have to be added between #$XLSXOpen and #$XLSXWkSh.

Figure out how to show negatives in red. Hopefully it has something to do with the style and not conditional formatting.

## Complete a unit testing setup

Currently the unit test is program #$XLSXTS. It still needs a lot of work. It needs to test every option and probably include information in the spread sheet on what to look for on each sheet based on what is being tested.

## Future Update Ideas

* Make the themes work.
* Add options for tables.
* Add options for pivot tables.
* Add conditional formatting.
* See if there is a way to auto fit the columns. I think I saw something in one of the XML files that stated to auto size the rows and columns. If this is an option add something like “ColumnWidth:\*AUTO” to the #$XLSXWKSH procedure.
* Create a read excel file function. This sounds hard, but using the EXPAT XML parser ported by Scott Klement, it would just be open the worksheet document only and parse the data. I guess it would have to handle referenced strings as well which would kind of suck but it would be doable. It would probably be a separate service program. I would have to research the unzip features of Zlib as well.
* Allow multiple sheets to be opened at once and written to at the same time till each one is closed. This idea came about because I want to be able to read from an input source and write data to different sheets based on some calculate criteria. To accomplish this each add data procedure will have to accept a sheet argument. If it is not passed and more than one sheet is opened an error will be sent. Also, the open sheet data structure will have to be changed to an array so all open sheet properties can be saved, I will probably have to limit the number of open sheets to like 20 or something. When adding data, the procedure will need to write the physical data to the correct sheet file. I probably need to implement some kind of versioning before this change is made since the protocalls will be changed.

# Examples and Learning (How to)

This section discusses how to use the #$XLSX service program. It shows examples of using the service program and where to look for more detailed information about the procedures or properties. First it helps if you understand some of the terminology used in the document. Read the Terminology section and then move on to the first example.

## Terminology

* Workbook – the entire collection, basically the Excel file itself. It can contain multiple worksheets.
* Worksheet – One sheet in a workbook.
* Style – set of properties that contains all the formatting options to use for a cell. Styles are setup in advance and then used when adding data to the worksheet.

## Example Source

Each example here has source and a working program on the AS400. They are all stored in the #$XLSX library. This table provides the names of the programs and source members for each example. You can copy these examples into a development library if you want to play with them, but do not change them in the #$XLSX library so they stay consistent with this document.

|  |  |  |
| --- | --- | --- |
| Example | Program/Source | Comment |
| Example 1 | #$XLSXE1, #$XLSXE1F | Basic Hello World example, the one followed by an F is in free format. |
| Example 2 | #$XLSXE2 | Example using different add data functions, Uses file #$XLSXINV |
| Example 3 | #$XLSXE3 | Example using Styles to format cells |
| Example 9 | #$XLSXE9 | Used in the #$XLSXCELL Example. RPG program #$XLSX9PF uses SQL to setup the temp file it uses. |

## Example 1 - Hello World

To keep with programming history we will start off with a simple hello world spread sheet. This program will be the simplest program that can possibly be used to create a spread sheet. The spread sheet will only populate one cell with the text “Hello World”. The following is the source code for the program. It is written in fixed format for people still using that, and then in free format.

Fixed format hello world program:

0001.00 H DATEDIT(\*YMD) OPTION(\*SRCSTMT:\*NODEBUGIO:\*NOSHOWCPY) DEBUG INDENT('| ')

0002.00 H DFTACTGRP(\*NO) BNDDIR('#$XLSX1.0/#$XLSX')

0003.00 F\* #$XLSX Hello World Program Fixed Format

0004.00 D/INCLUDE #$INCXLSX

0005.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0006.00 C\*

0007.00 C\* Start the Excel Work book - MUST BE DONE FIRST

0008.00 C CALLP #$XLSXOpen('OutputName:+

0009.00 C /EMAIL/TMP/#$XLSXE1.xlsx')

0010.00 C\*

0011.00 C\* CREATE A NEW SHEET(TAB)

0012.00 C CALLP #$XLSXWkSh()

0013.00 C\*

0014.01 C\* POPULATE THE FIRST CELL IN THE FIRST ROW

0015.00 C CALLP #$XLSXCHAR('Hello World’)

0016.00 C\*

0017.00 C\* Close the open XLS File

0018.00 C CALLP #$XLSXClose

0019.00 C\*

0020.00 C SETON LR

0021.00 C RETURN

0021.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Free format hello world program:

0000.50 \*\*free

0001.00 Ctl-Opt DatEdit(\*YMD) Option(\*NoDebugIO:\*SrcStmt:\*NoShowCpy) Indent('| ')

0002.00 DftActGrp(\*NO) BndDir('#$XLSX1.0/#$XLSX');

0003.00 // #$XLSX Hello World Program Free Format

0004.00 /include #$INCXLSX //copy book for prototype definitions

0006.00

0007.00 // Start the Excel Work book - MUST BE DONE FIRST

0008.00 #$XLSXOpen('OutputName:/EMAIL/TMP/#$XLSXE1.xlsx');

0009.00

0010.00 // Create a New Sheet(Tab)

0012.00 #$XLSXWkSh();

0013.00

0014.00 // Populate the First Cell in the First Row

0015.00 #$XLSXChar('Hello World');

0016.00

0017.00 // Close the open XLS File

0018.00 #$XLSXClose();

0019.00

0020.00 \*inlr=\*on;

0021.00 return;

Lines 1 and 2 – Default H specs. Line one is not really needed but should be included for standardization. Line 2 is required. The DFTACTGRP(\*NO) is required by any ILE program using procedures. The BNDDIR(‘#$XLSX1.0/#$XLSX’) binds the service program #$XLSX to this program so it’s functions can be used.

Line 3 – Just a comment with the program description.

Line 4 – Includes all the prototypes for the #$XLSX functions.

Line 8 – Opens the work book, this must be done first. It is passed one parameter which tells it where to save the workbook and what to call it. There are a lot of other options that can be added. For a full list see the [#$XLSOpen](#_#$XLSXOpen__=) section of this document. The format for the parameters is discussed in more detail there, but for a brief overview they are passed as character data containing a key and a value separated by a colon.

Line 12 – This creates the work sheet, it has a bunch of options available, but none are required. You can see the options in the [#$XLSWkSh](#_#$XLSXWkSh__=) section. They are passed as keyed pairs just the #$XLSXOpen. You must create a work sheet before you can add any data.

Line 15 – This adds a character value to a cell. When the worksheet is created a pointer is setup to keep track of the row and column, when you do #$XLSXChar it updates the current cell and moves the pointer to the next cell. Examples of adding other data types and multiple columns will be show in Example 2.

Line 17 – This closes the work book and writes out the Excel file. Nothing can be changed after this.

Line 20, 21 – This just ends the RPG program.

## Ordering

The order in which functions are called is important. The system writes some data to disk during the process to prevent the program from using up too much RAM. To make things work properly the order of function usage must follow this pattern:

1. Open a workbook, use #$XLSOpen, if any other function is used before the workbook is opened the program will send an error message.
2. Add any styles, these must be added before a work sheet is opened. You add styles with the #$XLSXStyle function. Styles are explained in Example 3.
3. Then open a worksheet. You do this with function #$XLSXWkSh. If more properties need to be added, you can repeat the #$XLSWkSh function as many times as needed to set properties until the first piece of data is added to the work sheet.
4. Add data to the work sheet. There are 8 functions that add data to a worksheet. They are #$XLSXChar, #$XLSNum, #$XLSXDate, #$XLSYYMD, #$XLSXMDY, #$XLSXMDYY, #$XLSXForm and #$XLSXNull. As soon as one of these is used you can no longer change properties on the sheet. Example 2 will explain what each of these do.
5. Optionally add another worksheet, this is done with the #$XLSXWkSh function just like step 3. However, if a worksheet has already had data written to it, it will close that sheet and start a new one. If a new sheet is started, add data to it just like step 4. You can add up to 100 sheets to a workbook, but working on it in excel becomes troublesome if you add too many sheets. Example 4 walks you through creating a workbook with multiple sheets.
6. Close the worksheet with the #$XLSXClose function. This will write the Excel file out to disk. Nothing can be changed after this is run. If you need to start a new workbook you can now start over with step 1.

## Example 2 – Adding more data and multiple rows

Now that you have seen how easy it can be to create a simple spreadsheet. Let’s look at adding more data to work sheet. In the Hello World example, we just added data to one cell. It used the function #$XLSXChar to add character data to the first cell. There are 8 functions used for adding specific types of data to a workbook. They are listed below.

1. #$XLSChar – Add character data to a cell.
2. #$XLSNumr – Add numeric data to a cell.
3. #$XLSDate – Add a date to a cell. This one accepts an RPG date data parameter.
4. #$XLSYYMD – Add a date to a cell. This one accepts and numeric date in YYYYMMDD format.
5. #$XLSMDY – Add a date to a cell. This one accepts and numeric date in MMDDYY format.
6. #$XLSMDYY – Add a date to a cell. This one accepts and numeric date in MMDDYYYY format.
7. #$XLSForm – Add a formula to a cell.
8. #$XLSNull – Doesn’t really add data, it just moves past a cell without adding anything. It does let you add a style to the cell which is why it is included with the add data functions. Styles will be discussed in Example 3.

When the sheet is opened, a pointer is set to point at row 1 column A. When adding data, it is always added to the current cell then the pointer is moved over one. Each of these functions add data, except for #$XLSNull, and then moves the cell pointer to the next cell to the right.

When all data has been added to a row you can use the #$XLSXNext function to move down to the next row. This resets the current column to A and changes the row to the next one.

This system lets you write data to a spreadsheet one row at time, similar to data in a database file. This service program only creates spreadsheets. It does not allow you to change anything. Once data is written there is no way to get rid of it.

This example will create a spreadsheet that looks like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Report Title | | |  | Date |
|  |  |  |  |  |
| Item | Description | Setup Date | Qty on Hand | Price |
| 124-00 | Base Widget | 1/01/2020 | 10 | 25.00 |
| 124-03 | Green Widget | 1/02/2020 | 52 | 32.75 |
| 124-05 | Blue Widget | 1/03/2020 | 13 | 31.45 |
| 125-00 | Widget Connector | 4/01/2022 | 27 | .34 |
|  | Total Qty: |  | 92 |  |

This example does not cover formatting these cells to look nice, it will discuss the parameters to each of the functions covered so far. These parameters will include the style which is what is used to format data, however it will pass \*OMIT or be excluded completely for this example.

Each data writing function contains similar parameters, except #$XLSXNull which excludes the value. The parameters include the value to add, a style to handle formatting and a number of cells to merge into this one. The style and the number of cells are optional. In this example we will merge the report title with the 2 cells to the right of it. The next example will show formatting of the cells. Since RPG does not qualify parameters, we will use the \*omit option to skip the style parameter.

All data is passed as a constant so it can be passed either via a variable or by a constant value. This allows us to add constant values like titles and column header and still pass variable data from fields. This example will assume that you have a file called #$XLSInv with 5 fields in the example above.

The following is the code for this program. Each line will be explained below the code.

0001.00 H DATEDIT(\*YMD) OPTION(\*SRCSTMT:\*NODEBUGIO:\*NOSHOWCPY) DEBUG INDENT('| ')

0002.00 H DFTACTGRP(\*NO) BNDDIR('#$XLSX1.0/#$XLSX')

0003.00 F\* #$XLSX Example adding more data

0004.00 F#$XLSXINV IF E K DISK

0005.00 D/INCLUDE #$INCXLSX

0006.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0007.00 C\*

0008.00 C\* Start the Excel Work book - MUST BE DONE FIRST

0009.00 C CALLP #$XLSXOpen('OutputName:+

0010.00 C /EMAIL/TMP/#$XLSXE2.xlsx')

0011.00 C\*

0012.00 C\* CREATE A NEW SHEET(TAB)

0013.00 C CALLP #$XLSXWkSh()

0014.00 C\*

0015.00 C\* Add Title row

0016.00 C MOVE \*DATE #DATE 8 0

0017.00 C CALLP #$XLSXChar('Inventory Report':\*omit:2)

0018.00 C CALLP #$XLSXNull()

0019.00 C CALLP #$XLSXYYMD(#DATE)

0020.00 C\*

0021.00 C\* Skip a Line

0022.00 C CALLP #$XLSXNext()

0023.00 C\* 0024.00 C\* Add the header line

0025.00 C CALLP #$XLSXNext()

0026.00 C CALLP #$XLSXChar('Item')

0027.00 C CALLP #$XLSXChar('Description')

0028.00 C CALLP #$XLSXChar('Setup Date')

0029.00 C CALLP #$XLSXChar('Qty on Hand')

0030.00 C CALLP #$XLSXChar('Price')

0031.00 C\*

0032.00 C\* Loop through file and add each detail line, save count so a formula

0033.00 C\* can be added at the bottom

0034.00 C MOVE \*ZEROS COUNT 5 0

0035.00 C READ #$XLSXInv

0036.00 C DOW NOT %EOF

0037.00 C ADD 1 COUNT

0038.00 C CALLP #$XLSXNext()

0039.00 C CALLP #$XLSXChar(Item)

0040.00 C CALLP #$XLSXChar(Desc)

0041.00 C CALLP #$XLSXYYMD(Setup)

0042.00 C CALLP #$XLSXNumr(QtyOh)

0043.00 C CALLP #$XLSXNumr(Price)

0044.00 C READ #$XLSXInv

0045.00 C ENDDO

0046.00 C\*

0047.00 C\* ADD A TOTAL LINE USING A FORMULA TO CALCULATE IT

0048.00 C CALLP #$XLSXNext()

0049.00 C CALLP #$XLSXNull()

0050.00 C CALLP #$XLSXChar('Total:')

0051.00 C CALLP #$XLSXNull()

0052.00 C CALLP #$XLSXForm('=SUM(D4:D'+%char(count+3)+')')

0053.00 C\*

0054.00 C\* Close the open XLS File

0055.00 C CALLP #$XLSXClose

0056.00 C\*

0057.00 C SETON LR

0058.00 C RETURN 0021.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

At this point it is easier to look at the source code provided with this package for the test program while viewing the line explanations below. If you have access to a machine the software is loaded on.

**Lines 1 and 2** – Default H specs. Line 1 is not really needed but should be included for standardization. Line 2 is required. The DFTACTGRP(\*NO) is required by any ILE program using procedures. The BNDDIR(‘#$XLSX1.0/#$XLSX’) binds the service program #$XLSX to this program so it’s functions can be used.

**Line 3** – Just a comment with the program description.

**Line 4** – Defines our input file, this program reads through the file for simplicity, the data can come from anywhere. Generally SQL would be used to pull the data.

**Line 5** – Includes all the prototypes for the #$XLSX functions.

**Line 9** – Opens the work book, this must be done first. See Example on for an explanation of what this does.

**Line 13** – This creates the work sheet. See example 1 for more information.

**Line 16-19** – This group adds a title line to the spreadsheet.

**Line 16** - pulls the system date into a variable. The Hspecs include DATEDIT(\*YMD) which means the date will be in YYYYMMDD format.

**Line 17** – Adds the report title. The call to the #$XLSXChar procedure accepts up to 3 parameters. The first one is the value to add to the cell, the second one is the style and the third one is a number of cells to merge into this one. The style is omitted, style will be covered in Example 3. The merge option is set to 2, this means the cell will span 3 columns. When adding a style to the title in example 3 title will required 3 columns to fit.

**Line 18** – Skips a row. The #$XLSXNull function just skips a columns. Since the title took 3 columns and this column is skipped the cell to write to will column 5 or E in Excel.

**Line 19** – Adds a date. This example uses the #$XLSXYYMD function to add the date. That is because the date is in YYYYMMDD format from line 16. If the date was pulled as a date data type field the program would use procedure #$XLSXDate instead. To make programming more concise the function could have been #$XLSXDate(%date()) which would pull the current date and pass it directly to the add function.

Dates in Excel are stored as a number of days, so without formatting this will look funny for now. Example 3 will discuss date formatting which is accomplished by using styles.

**Line 22** – Skips a line. The #$XLSXNext function moves to the next line, all section after the first one need to start with this function or the data will just keep going right across the spread sheet. This section of code only skips a line, it doesn’t write out any data so since the next section also goes to a new line it will leave an empty column in the spreadsheet.

**Line 25-30** – Adds a header row. The section start with the #$XLSXNext function which moves the cursor to the start of the next row, then it uses the #$XLSXChar function to add column headers.

**Line 34** – Initializes a count variable this variable is used to get the number of detail rows added. Line 52 will use the count to create a formula to calculate the total.

**Line 35,36, 44 and 45** – These line crate a loop to read through the data.

**Line 37** – Increments the count variable to keep track of how many rows are added to the spreadsheet.

**Line 38-43** – Write one detail row to spreadsheet.

**Line 38** – Moves the cursor to the start of the next row of the spreadsheet.

**Line 38-40** – uses the #$XLSXChar function to add an item and description from the input file to the row.

**Line41** – uses the #$XLSXYYMD function to add the setup date to the spreadsheet. Note the setup date in the file is stored as a number in YYYYMMDD format. If it was a date data type variable the #$XLSXDate function would be used. If it was stored as a numeric MMDDYYYY variable the #$XLSXMDYY function would be used. If it was stored as a numeric date in MMDDYY format the #$XLSXMDY function would be used. These function were added to make adding common date formats easy. If a date is stored in any other format it will have to be converted to one of these date formats to be added.

**Line 42 and 43** – These lines add the quantity on hand and price data to the spreadsheet. They use the procedure #$XLSXNumr. This procedure works like the #$XLSXChar procedure except it accepts a numeric value instead of a string. Numbers are stored differently in Excel and using this procedure will flag the data as numeric. If #$XLSXChar(%char(Price)) was used the data would be added as a string and it would be left justified instead of right justified. String can also not be used in formulas like numbers can, so using the #$XLSXNumr function for numeric data is important.

**Lines 48-53** – add a total line, 48 moves the cursor to the next row, 49 skips a cell, 50 adds the constant Total:, 51 skips a cell.

**Line 52** – Adds a total formula. This line uses the #$XLSXForm procedure to add a formula to total the quantity on hand. In Excel the sum function is used to get the total. The sum function totala all values in a range. A range in excel is written as D4:D9 where D4 is the cell location of the upper left hand corner of the range and D9 is the cell location of the lower right hand cell in the range. Since both colums use D the function =sum(D4:D9) will total all the values in column D starting in row 4 through row 9. The program must come up with the formula. In this example we know that the all the quantity one hands are in column D because that is where we wrote them out. We also know the first row will be 4 because we wrote a title, skipped a line and then a header row. We know that the last row will be the number of lines we wrote out plus the 3 line we skipped before writing out the data. Using this information we can generate the formula.

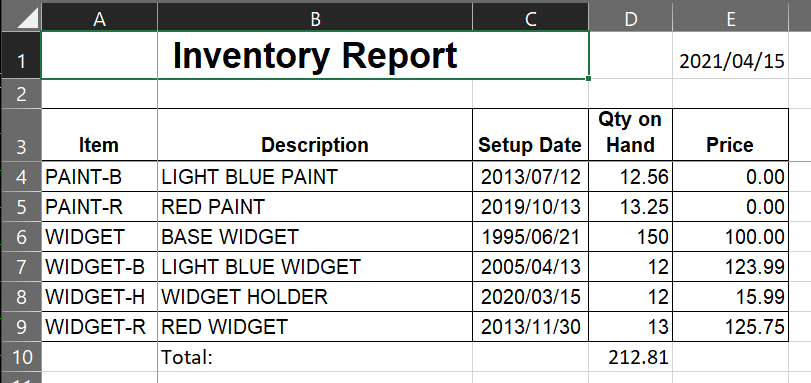
For a shortcut to manually calculating ranges review the #$XLSXCell procedure which can produce a range relative to the current cell of the spreadsheet. It is the better way to generate the range because it take the static values out of the equation and makes everything dynamic in case rows or columns are added later. Covering that function is little more complex than you need in Example 2 though.

**Line 55** – This closes the work book and writes out the Excel file. Nothing can be changed after this.

**Line 57, 58** – This just ends the RPG program.

## Example 3 – Formatting cells

This example will produce the same spreadsheet as we did in example 2 except it will no add formatting. The output will now look like this:



Notice the title is big and bold, the headers are bold and centered, dates are formatted correctly as is the price, there are borders around and although you cannot see it the first row and header data are frozen so they remain visible while the file is scrolled through.

Formatting is done using styles. A style must be setup before a worksheet is opened. The style is given a name and all properties are set on the style. When adding data to the spreadsheet the style can be passed to use on the cell.

Style are added to the spreadsheet after the #$XLSXOpen procedure is used and before the first #$XLSXWkSh procedure is issued. Styles are added via the #$XLSXStyle procedure. The procedure accepts parameters just like the #$XLSXOpen procedure. Each parameter is passed as a keyed pair consisting of a Key and a value. Some keys a can take more than one value though. The only key that is required to create a style is the name. The name is what will be passed in the style parameter of any of the add data type functions.

For example #$XLSXStyle(‘Name:Title’) will create a style named Title. However, since no other options were passed this style would be useless because it would show just like the default of not passing a style. The following code is the style used for the title in the example above:

C CALLP #$XLSXStyle('name:TITLE'

C : 'Font:Arial'

C : 'PointSize:18'

C : 'Alignment:CENTER'

C : 'BoldWeight:BOLD')

Notice the first keyed pair passed is the name. This sets up the style title. Style names are not case sensitive, they are all converted to capital letters in the service program so you can send them in any case you want.

The rest of the parameters each setup an extra property for the formatting of the cell. Most are self-explanatory. The #$XLSXStyle section of this document lists all the options you can use along with appropriate values for each one. Suffice it say for now that the Title style will use the font Arial, point size 18, will wrap text (although it is not used in this example), will be center aligned and bold.

The following is the code for example 3:

0001.00 H DATEDIT(\*YMD) OPTION(\*SRCSTMT:\*NODEBUGIO:\*NOSHOWCPY) DEBUG INDENT('| ')

0002.00 H DFTACTGRP(\*NO) BNDDIR('#$XLSX1.0/#$XLSX') ACTGRP(\*NEW)

0003.00 F\* #$XLSX Example adding styles

0004.00 F#$XLSXINV IF E K DISK

0005.00 D/INCLUDE #$INCXLSX

0006.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0007.00 C\*

0008.00 C\* Start the Excel Work book - MUST BE DONE FIRST

0009.00 C CALLP #$XLSXOpen('OutputName:+

0010.00 C /EMAIL/TMP/#$XLSXE3.xlsx')

0011.00 C\*

0012.00 C\* SETUP STYLES

0013.00 C\*

0014.00 C\* MAKE THE TITLE BIG, BOLD AND CENTERED

0015.00 C CALLP #$XLSXStyle('name:TITLE'

0016.00 C : 'Font:Arial'

0017.00 C : 'PointSize:18'

0018.00 C : 'WrapText:Yes'

0019.00 C : 'Alignment:CENTER'

0020.00 C : 'BoldWeight:BOLD')

0021.00 C\* THE DATE ON THE TITLE ROW NEEDS A DATA FORMAT SO IT IS EDITED CORRECTLY

0022.00 C CALLP #$XLSXStyle('name:TDATE'

0023.00 C : 'DataFormat:yyyy/mm/dd')

0024.00 C\*

0025.00 C\* MAKE THE HEADER ROW, BOLD, CENTERED WITH BORDERS...

0026.00 C CALLP #$XLSXStyle('name:HDR'

0027.00 C : 'Font:Arial'

0028.00 C : 'PointSize:10'

0029.00 C : 'WrapText:YES'

0030.00 C : 'Alignment:CENTER'

0031.00 C : 'BoldWeight:BOLD'

0032.00 C : 'BorderTop:THIN'

0033.00 C : 'BorderBottom:MEDIUM'

0034.00 C : 'BorderLeft:THIN'

0035.00 C : 'BorderRight:THIN')

0036.00 C\*

0037.00 C\* THESE THREE ARE ALL FOR THE DETAIL ROW, THE FIRST ONE IS

0038.00 C\* A GENERAL STYLE, THE NEXT TWO SHOW FORMATING FOR THE DATE

0039.00 C\* AND DOLLAR FIELDS

0040.00 C CALLP #$XLSXStyle('name:DTL'

0041.00 C : 'Font:Arial'

0042.00 C : 'PointSize:10'

0043.00 C : 'BorderTop:THIN'

0044.00 C : 'BorderBottom:THIN'

0045.00 C : 'BorderLeft:THIN'

0046.00 C : 'BorderRight:THIN')

0047.00 C CALLP #$XLSXStyle('name:DATE'

0048.00 C : 'Font:Arial'

0049.00 C : 'PointSize:10'

0050.00 C : 'Alignment:CENTER'

0051.00 C : 'BorderTop:THIN'

0052.00 C : 'BorderBottom:THIN'

0053.00 C : 'BorderLeft:THIN'

0054.00 C : 'BorderRight:THIN'

0055.00 C : 'DataFormat:yyyy/mm/dd')

0056.00 C CALLP #$XLSXStyle('name:DOLLAR'

0057.00 C : 'Font:Arial'

0058.00 C : 'PointSize:10'

0059.00 C : 'BorderTop:THIN'

0060.00 C : 'BorderBottom:THIN'

0061.00 C : 'BorderLeft:THIN'

0062.00 C : 'BorderRight:THIN'

0063.00 C : 'DataFormat:#,##0.00')

0064.00 C\*

0065.00 C\* CREATE A NEW SHEET(TAB), SET THE SHEET NAME, FREEZE THE TOP 3 ROWS

0066.00 C CALLP #$XLSXWkSh('SheetName:#$XLSXT3'

0067.00 C : 'FreezeRows:3'

0068.00 C : 'FreezeColumns:1')

0069.00 C\*

0070.00 C\* SET SOME COLUMN WIDTHS.

0071.00 C\* THESE COULD HAVE BEEN DONE ON THE LINE ABOVE, BUT THEY ARE LEFT

0072.00 C\* HERE TO SHOW HOW ADDITIONAL PROPERTIES CAN BE ADDED BY RE-RUNNING

0073.00 C\* THE #$XLSXWkSh FUNCTION, THE FUNCTION ACCEPTS UP TO 200 PROPERTIES

0074.00 C\* SO THE ONLY REASON THIS WOULD BE DONE IS IF THERE ARE HUNDREDS OF

0075.00 C\* COLUMNS YOU WANT TO SET THE WIDTH FOR.

0076.00 C CALLP #$XLSXWkSh('ColumnWidth:1:11'

0077.00 C : 'ColumnWidth:2:30'

0078.00 C : 'ColumnWidth:3:11'

0079.00 C : 'ColumnWidth:4:8'

0080.00 C : 'ColumnWidth:5:11')

0081.00 C\*

0082.00 C\* Add Title row

0083.00 C\* Notice on the #$XLSChar for the title we have added the style name

0084.00 C\* as the second parameter. There is also a third parameter, that is the

0085.00 C\* number of cells to merge in to the cell we are using. This formats the

0086.00 C\* with the properties from the style and makes it span 3 columns.

0087.00 C\* All add data type functions have style and merge for the 2nd and 3rd parm.

0088.00 C MOVE \*DATE #DATE 8 0

0089.00 C CALLP #$XLSXChar('Inventory Report':'TITLE':2)

0090.00 C CALLP #$XLSXNull()

0091.00 C CALLP #$XLSXYYMD(#DATE:'TDATE')

0092.00 C\*

0093.00 C\* Skip a Line

0094.00 C CALLP #$XLSXNext()

0095.00 C\*

0096.00 C\* Add the header line

0097.00 C CALLP #$XLSXNext()

0098.00 C CALLP #$XLSXChar('Item':'HDR')

0099.00 C CALLP #$XLSXChar('Description':'HDR')

0100.00 C CALLP #$XLSXChar('Setup Date':'HDR')

0101.00 C CALLP #$XLSXChar('Qty on Hand':'HDR')

0102.00 C CALLP #$XLSXChar('Price':'HDR')

0103.00 C\*

0104.00 C\* Loop through file and add each detail line, save count so a formula

0105.00 C\* can be added at the bottom

0106.00 C MOVE \*ZEROS COUNT 5 0

0107.00 C READ #$XLSXInv

0108.00 C DOW NOT %EOF

0109.00 C ADD 1 COUNT

0110.00 C CALLP #$XLSXNext()

0111.00 C CALLP #$XLSXChar(Item:'DTL')

0112.00 C CALLP #$XLSXChar(Desc:'DTL')

0113.00 C CALLP #$XLSXYYMD(Setup:'DATE')

0114.00 C CALLP #$XLSXNumr(QtyOh:'DTL')

0115.00 C CALLP #$XLSXNumr(Price:'DOLLAR')

0116.00 C READ #$XLSXInv

0117.00 C ENDDO

0118.00 C\*

0119.00 c\* Add a total line using a formual to calculate it

0120.00 C CALLP #$XLSXNext()

0121.00 C CALLP #$XLSXNull()

0122.00 C CALLP #$XLSXChar('Total:')

0123.00 C CALLP #$XLSXNull()

0124.00 C CALLP #$XLSXForm('=SUM(D4:D'+%char(count+3)+')')

0125.00 C\*

0126.00 C\* Close the open XLS File

0127.00 C CALLP #$XLSXClose

0128.00 C\*

0129.00 C SETON LR

0130.00 C RETURN

0131.00 C\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I will not document all lines in this program because it uses the same flow as example2. The following will only document lines with changes. It is best to view the source on the aS400 while reading through this explanation.

**Lines 12-63** setup the style. These ones are a little drawn our examples sake. Notice they come after the #$XLSXOpen procedure but before the #$XLSXWkSh procedure.

**Lines 15-20** create the style for the title. The parameters are pretty self-explanatory. This makes the title big bold and centered.

**Lines 22-32** create style for the date to the right of the header. The only parameter passed in the DataFormat parameter. This parameter tells Excel hoe to format the data. Since this is a data field the format passed is yyyy/mm/dd. This format will show the date like 2021/04/05. The section on the [dataFormat](#_DataFormat_–_Sets) in the #$XLSXStyle section gives a lot of examples on different date formatting. This data format will be used on the price column as well to set the static decimal positions.

**Lines 26-35** set the style for the header row. In this style the WrapText option becomes important. If you look at the example output above you can see that the Qty on Hand column header is split to 2 lines. This would not happen without the WrapText:Yes option. This style also introduces the Border Options. They are set one option at a time in this example. The bottom border is set to medium to produce a thicker line under the headers and the rest of the options are set to thin. All the available border options are listed in the [borders](#_Borders) section under the #$XLSXStyle procedure in this document. Try not to get carried away with too much borders as they can distract form the data in the report.

**Lines 40-46** setup the style for regular detail cells. There is nothing new to report here.

**Lines 47-55** setup the style for the detail date cell. They use the same options as the regular detail but it adds the DataFormat to format the date correctly.

**Lines 56-63** setup the style for the detail line price column. This uses the same options as the regular detail lines except it adds a dataFormat option to force 2 decimal positions and to add commas to the number. Excel stores numeric as just a number, so it you want it display a special way you have to tell it to. The [dataFormat](#_DataFormat_–_Sets) section of this document gives example of commonly used data formats with numbers.

**Lines 66-68** Starts a worksheet. This differs from the previous example because it passes parameters when it starts it. The first parameter is the sheet name, this is just what excel will put on the tab at the bottom where you change sheets.

The next 2 parameters set the number of rows and columns to freeze. Freezing the first row really shouldn’t be done unless the data spreads across the sheet wider than can be seen on the screen. This make it so when you scroll over you can still see what item you are on. Freezing the top 3 columns makes it so when you scroll down the title and header lines stay at the top of the page. This should pretty much be done on all reports.

**Lines 76-80** set the column widths. These rally should have been done on the first call to #$XLSXWkSh but this shows how you can keep setting parameters until the first piece of data is added. Once the first piece of data is added the start of the sheet is actually written to the file so nothing on it can be changed. The #$XLSXWkSh procedure accepts up to 200 options so unless you are adding a whole lot of column width there is not really a reason to call the procedure twice.

The ColumnWidth key is the first one we have used that requires more than one value. The keyed pair is now listed with a Key:value1:value2 format. The columnWidth key uses valuea as the column to set the width for and value2 as the width. The width is set in an approximate number of character when using the base font. So if your item number is an 8 character field you should be able to set the width to 8. However you also need to look at the header and make sure it fits in the width and if you use a different size font you will just have to play around with it to see what works best.

**Line 89** – Add title. To the call to #$XLSXChar we have added the style name TITLE. This style was setup on lines 15-20. It is going to make the title big, bold and centered.

**Line 91** Add run date. To this line we have used the style TDATE which is setup with a dataformat to format the date the way we want.

**Lines 97-102** Add column header. We have added the style HDR to format the header row.

**Lines 111,112 and 114** Detail lines, we added the style DTL to add borders and set the font and size.

**Line 113** adds the date. It now includes the style DATE which is similar the style DTL except it has the dataFormat option to format the date.

**Lin115** adds the price. It now includes the style DOLLAR which adds a data format to force tow decimal positions and add commas for thousands separators.

The remainder of the code is the same as example 2.

# Procedure Details

This section covers each procedure individually. It documents all options for each procedure.

## #$XLSXOpen = Create a New Workbook

* Initialize all data
* Setup temporary folder and subfolders
* Set initial style with a name of blank

This function must be called first before any other functions can be used. This function accepts a list arguments. The arguments are just text strings that normally contain matched pairs of a key and a value, however some keys require multiple values. Colons are used for separators. For instance the value ‘OutputName:/EMAIL/TMP/#$XLSXTS.xlsx’ designates the location and name of the Excel file produced. The key is OutputName and the value is /EMAIL/TMP/#$XLSXTS.xlsx.

The OutputName example above is the only required parameter, all other are optional.

The keys are not case sensitive so OutputName, OUTPUTNAME and outputname will do the same thing. Value however can be case sensitive depending on their use. Each argument option is detailed below and lists what each key does and any specific details about it.

### OutputName – tells #$XLSX what to name the output file

This key value requires one value to be included. The value is case sensitive so that the program can handle how the excel file will be named. The value passed should include the full or relative path to the folder the file should be created in as well as the file name. If the name part ends in anything but .xslx, the .xlsx extension will be added. This is a required argument.

Examples:

* OUTPUTNAME:/EMAIL/TMP/Test - Creates the file Test.xlsx in the folder /EMAIL/TMP.
* OUTPUTNAME:Test - Creates the file Test.xlsx in the current directory for the job.
* OUTPUTNAME:TMP/Test.XLSX - Creates the file Test.XLSX in the subfolder TMP in the current directory for the job.

### Debug – Generates the xml with tabs and end of line characters.

The debug option is meant to be used by developers working on the #$XLSX program. It adds indenting and line breaks to the XML output that makes up the Excel file. These increase the size of the file but makes it much easier to debug the code. If not passed the value will be Off by default.

This key accepts most values generally used for Boolean type data. The values that can be used are Y, Yes, On, \*on, true and 1. These values are not case sensitive. If any other value is passed debug will remain the default of off.

Example

* Debug:on – Turns on debug option

### Author (Creator) – Sets the author of the file

The Author option can be sent as either Author or Creator, they both set the same value in the file. Both options are supported because different programs will list the value as one or the other and it was deemed helpful to support either syntax. If not passed no author will be set. The value is case sensitive so that names will look better.

Example

* Author:Tim Tognazzini – set the author to Tim Tognazzini
* Creator:Tim Tognazzini – sets the author to Tim Tognazzni

### Company – Set the company property for the spread sheet

The Company option just sets the company name in the properties of the Excel file. The value is case sensitive.

Example:

* Company:Tognazzini, Inc.

### Title – sets the title property for the Excel file

The Title option sets the title property of the spread sheet.

Example:

* Title:Physical Inventory Count List

### Subject – sets the subject property for the file

The Subject key set the subject property of the workbook.

Example:

* Subject:Some Subject

### Manager – sets the manager property for the file

The manager key sets the manager property of the Excel file.

Example

Manager:Tim Tognazzini

### Category – sets the category property for the file

The category key sets the category property of the work book.

Example:

* Category:Inventory Control

### Status – sets the status property for the file

The status option sets the status property for the spreadsheet.

Example:

* Status:Open

### Comments – Adds comments to the file

The comment key sets the comments for the work books. Multiple lines of comments can be entered by inserting the end of line characters. If including #$include in your program the variable EOR will already be declared which contains the end of line characters. You are limited to 2048 characters per passed argument, so the total length of the comment with the end of line characters can only be 2039, because the key and semicolon (“Comments:”) takes up 9 characters. EOR takes up 2 characters per use.

If #$include is not being used, EOR can be defined in the program or the constant of x'0D25' can be used which is an end of line character followed by carriage return.

Examples:

* Comment:One Line Comment
* ‘Comment:First line’ + EOR + ‘Second Line’
* ‘Comment:First Line’ + x'0D25' + ‘Second Line’

### HyperLinkBase – Adds the Hyper Link Base property

This adds the hyper base link property to the file. The hyper link base is a default URL that gets used for any hyperlink added to the document. Not really sure what you would use this for, but’s it’s there if you need it.

Example:

* HyperLinkBase:somelink.com

### Tags (Keywords) – Adds tags or keywords to the file

Allows adding tags to the file. Tags and the Keywords are used interchangeably so the function will accept either option, but it does the same thing in the file. Multiple tags can be entered, they are just separated by semicolons.

Examples:

* Tags:tag
* Tags:tag1;tag2;
* Keyword:tag1;keyword2;

### CustomProperty – Adds custom properties to the file

Most other arguments in this section are used to handle the default properties that Excel supports. However, Excel also allows entry of custom properties. These properties just allow the user to enter the name, type and value of the property. The #$XLXS system allows entry of up to 30 customer properties for a workbook.

The format of a custom property requires using the key value of CustomerProperty and then supplying the name, type and value of the property. The name type and value are separated by a colon just like the key and other values are. So CustomProperty:Book Date:Date:20200321 will add a custom property of Book Date with a type of Date and a value of 20200321.

The name can be up to 128 characters long, anything else will be truncated. The type must be Text, Date, Number, YesOrNo or Boolean. Boolean just does the same thing as yes or no.

The value parameter must match the type. Text type can have any characters in the value. The date type must be a numeric date in YYYYMMDD format without any separator. The number type must be numeric data. It can include a leading negative sign and a decimal point.

The YesorNo/Boolean type must be a value that translates to yes or no. These values are allowed for yes: yes, y, on, \*on, 1 and true. These values are allowed for no: no, n, off, \*off,0 and false. The values for YesOrNo are not case sensitive so TRUE, True and true all equate to Yes.

Examples;

* CustomProperty:Market:Text:Nasdaq
* CustomProperty:Days Old:Number:57
* CustomProperty:Negative Number with decimals:Number:-57.34
* CustomProperty:Book Date:Date:20200221
* CustomProperty:Reviewed:YesOrNo:false
* CustomProperty:Paid:boolean:Yes

### Buffering – Sets the file buffering option

The service program uses file buffering by default. This causes it to write data to memory instead of directly to file till it is all accumulated or the max buffer size is reached, which increases program efficiency.

Turning buffering off will make the program run a lot slower. In testing it took over 30 times as long to write directly to disc than it did using buffering.

All Boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so ON, On and on will all turn on buffering.

Buffering is on by default, so you do not need to do anything to turn it on. However, if you have the need to pass the parameter anyways you can use any of these options to turn it on:

* Buffering:On
* Buffering:\*on
* Buffering:Yes
* Buffering:y
* Buffering:1
* Buffering:True

If you want to turn off buffering, you can do so with any of the following options.

* Buffering:Off
* Buffering:\*off
* Buffering:no
* Buffering:N
* Buffering:o
* Buffering:FALSE

## #$XL2Style = Create a style

* Create a style to use when adding data to a cell
* Must be done after #$XLSOPEN and before#$XLSWkSh

Styles are a set of options used to format a cell. Each style contains all formatting options for the cell. A style can be applied to as many cells as needed, but the style only needs to be defined once. Styles allow changing the font, size, color, fill, border options and more of a cell in a workbook.

This function is used to define the style. Each style must be given a name, all other properties are optional, however at least one other property should be passed or there is no point in using a style. See Example 3 for information on using styles, it demonstrates their use.

All options are passed using a character field containing a key and at least one value. The key and values are separated by a semi colon. For instance the name of the style is passed like this Name:Style1 where Name is the key and Style1 is the value. The function can accept a list of up to 200 parameters. If a parameter is duplicated each addition instance overrides the value from the previous instance. For instance if the following keys are passed the font will be Courier.

#$XLSStyle(‘Name:Style1’:’Font:Arial’:’PointSize:12’:’Font:Courier’);

Key names are not case sensitive, so FONT, Font and font will all set the font value. Values may be case sensitive depending on which key is passed. The key values listed below will dictate if a value is case sensitive. For the most part the font is the main one that has to match the case.

The following is a list of all the key values that can be passed and what values they accept.

### Name – The style name

Sets the name of the style. This name can be passed to any of the functions that add data to cell. The name is used to tie the style to the cell. This is a required parameter. The Value is not case sensitive so Name:Style1 and Name:STYLE1 will do the same thing. Likewise a style can be named Style1 in this function and then passed as STYLE1 in one of the add data functions.

The name is limited to 50 characters. Any additional characters will be truncated.

### BoldWeight – Sets font to bold or not

Set the font to bold or not. The only two values used are Bold and Normal. They are not case sensitive so you can use BOLD, Bold or bold.

Use the following parameter to set the font to bold: BoldWeight:Bold

The default is not bold, so by just not passing a bold weight parameter it will not be bold. However if you feel the need to pass the bold weight and do not want it bold you can use the normal option like this: BoldWeight:Normal

### PointSize – Font Point Size

This option set the point size of the font. The point size is passed as a number ranging between 1 and 409. Any other value will be ignored and an inquiry message will be sent.

The following example sets the point size to 12: PointSize:12

### Font – Font Name

This function sets the font name. These are the exact names shown in the drop down font box in Excel. These name are case sensitive, so if you see Arial in Excel you cannot pass arial, because it will not be found.

To set the font to Arial pass the argument like this: Font:Arial

For Times New Roman it would look like this: Font:Times New Roman

When using font you must remember that your computer may have custom fonts loaded that will not be on everyone else’s computer. If you select a font that another person does not have installed, Excel will use the default font instead. It is best to stick to the standard fonts like Arial, Times New Roman or Courier.

#$XLSX does not have the ability to embed fonts, mostly because the fonts do not exist on the server creating the spread sheet.

### Color – Font Color

This sets the color of the text in a cell. Colors can be passed using 3 different systems. The options are indexed, RBG or theme.

Indexed colors are passed as a string. For instance Color:Red will set the cell color to red. [Appendix 1](#_Appendix_1_–) has the list of index color strings. If you pass a name that is not on the list then the color will be left the default.

RBG colors are passed as a Hexadecimal value relating to the RBG index. TODO this still needs to be done.

Theme colors are passed as a reference to a part of the theme. When using themes all colors in the spread sheet can be changed by changing the theme. TODO figure this out and make it work.

### Italic – Set cell to italic.

This option sets the text in the cell to italic or not. It accepts a Boolean value, which equates to yes or no. All Boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so On, On and on will all turn on italics.

To turn on italics use any of these options:

* Italics:On
* Italics:\*on
* Italics:Yes
* Italics:y
* Italics:1
* Italics:True

Italics are off by default so there is no reason to pass the italics parameter if you do not want them on. However, if you have the need to pass the parameter anyway you can set them to off with any of the following options.

* Italics:Off
* Italics:\*off
* Italics:no
* Italics:N
* Italics:0
* Italics:FALSE

### StrikeOut – Set Cell to have a line through the text

This option sets the text in the cell to have a line through it or not. It accepts a boolean value, which equates to yes or no. All boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so On, On and on will all turn on the strike out option.

To turn on the strike out option use any of these options:

* StrikeOut:On
* StrikeOut:\*on
* StrikeOut:Yes
* StrikeOut:y
* StrikeOut:1
* StrikeOut:True

The strike out option is off by default so there is no reason to pass this parameter if you do not want it on. However, if you have the need to pass the parameter anyway you can set it to off with the any of the following options.

* StrikeOut:Off
* StrikeOut:\*off
* StrikeOut:no
* StrikeOut:N
* StrikeOut:0
* StrikeOut:FALSE

### TypeOffset – Set text to superscript or subscript.

This option sets the text in the cell to be subscript or super script. Subscript is where the text is smaller and lower in the cell while superscript is smaller and higher in the cell. These option kind of mess with the vertical alignment so use them with caution and test what it looks like. The typeoffset argument accepts a handful of values that equate to superscript of sub script. The options are listed below. Do include the argument if you do not want the style to be subscript or superscript. The argument is not case sensitive so SUPER, Super and super all do the same thing.

To turn on the subscript option use any of these options:

* TypeOffset:1
* TypeOffset:sub
* TypeOffset:subscript

To turn on the superscript option use any of these options:

* TypeOffset:2
* TypeOffset:sup
* TypeOffset:super
* TypeOffset:superscript

Although not passing the option is the preferred way to not use it, the system does also except the following values to not use subscript or superscript.

* TypeOffset:0
* TypeOffset:None

### Underline – Sets underline style

This option sets the text in the cell to be underlined. Excel hands a single or double underline. The values are not case sensitive so Single, SINGLE and single will all turn on a single underline. The following is the list of values.

* Underline:None
* Underline:Yes - Same as Single
* Underline:Single
* Underline:Double

### Alignment/VerticalAlignment – Sets Horizontal/Vertical Alignment

The system allows three ways to set the vertical and horizontal alignment in a cell. Vertical and horizontal alignment can be set individually using the keys VerticalAlignment and Alignment, or two parameters can passed to the Alignment key to set horizontal alignment and then the vertical alignment in one argument. For instance “Alignment:Left:Top” does the same thing as “Alignment:Left” and “VerticalAlignment:Top”.

The options for horizontal alignment are:

* CENTER
* CENTER\_SELECTION
* FILL
* GENERAL
* JUSTIFY
* LEFT
* RIGHT

The options for vertical alignment are:

* TOP, 0
* CENTER, 1
* BOTTOM, 2
* JUSTIFY, 3
* DISTRIBUTED, 4

### DataFormat – Sets the Data formatting option.

The data format is used for formatting numeric data. It can be used to format dates, times or numbers. The best way to figure out how the format needs to be entered is to play with it in Excel. However, some of the common entries are listed below. This should only be used on numeric columns. It will be ignored on non-numeric fields. Date and Time fields are stored as number in Excel and the data format is what converts them to human readable dates or times.

#### Numeric Formats

Number formats can get little complicated, but the examples below should give you a good idea of how to use them. By default, Excel does not show leading zeros or trailing zeros after the decimal point. This makes dollar columns look bad because the decimal points do line up for whole numbers. It is suggested to always include a data format for dollar amounts so they line up nicely.

In Excel you can right click a cell, select Format Cells, and make sure the Number tab is selected, then click Custom to see some of the standard custom formats. Playing with these it becomes easier to see how to make negatives show as brackets or red as well as some other common features.

Below are some commonly used formats.

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Description | Value | Formatted |
| #,##0 | Hide decimals and show commas | 123456.789 | 123,456 |
| #,##0.00 | Show 2 decimals and commas | 123456.789 | 123,456.78 |
| $ #,##.00 | Include dollar sign, 2 decimals and commas | 123456 | $ 123,456.00 |
| 0.00 | Show 2 decimals and not commas | 123456.7 | 123456.70 |
| 0 | Hide decimals and no commas | 123456.789 | 1234567 |
| 000000-00 | Format a fabric number with the hyphen W/leading 0 | 123456 | 01234-56 |
| 0-00 | Format a fabric number with hyphen W/out leading 0 | 123456 | 1234-56 |

#### Date Formats

Date formats are pretty simple they use a combination of M,MM,D,DD,YY,YYYY and separators. D differs from DD in that DD includes a leading zero and DD does not, this works the same way with M and MM. YY is a 2 digit year and YYYY is a four digit year. All the examples bellow use / as the separator, but it can be replaced with any symbol like a – or period.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format | Value | Formatted | Value 2 | Formatted 2 |
| yyyy/mm/dd | 20200101 | 2020/01/01 | 20201010 | 2020/10/10 |
| yyyy/m/d | 20200101 | 2020/1/1 | 20201010 | 2020/10/10 |
| yy/mm/dd | 20200101 | 20/01/01 | 20201010 | 20/10/10 |
| yy/m/d | 20200101 | 20/1/1 | 20201010 | 20/10/10 |
| m/d/yy | 20200101 | 1/1/20 | 20201010 | 10/10/20 |
| mm/dd/yy | 20200101 | 01/01/20 | 20201010 | 10/10/20 |
| m/d/yyyy | 20200101 | 1/1/2020 | 20201010 | 10/10/2020 |
| mm/dd/yyyy | 20200101 | 01/01/2020 | 20201010 | 10/10/2020 |

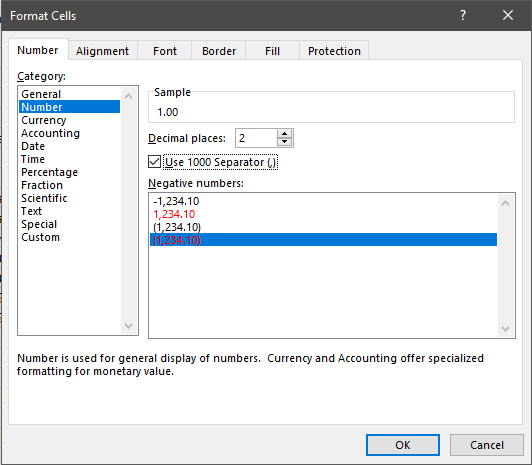
#### Time Formats

Time formats are pretty easy as well. They use H for the hour, MM for minutes and SS for seconds. By default they are 24 hour times, appending AM/PM to the end converts it a 12 hour time. See the examples below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format | Value | Formatted | Value 2 | Formatted 2 |
| h:mm | 134103 | 13:41 | 110535 | 11:05 |
| h:mm:ss | 134103 | 13:41:03 | 110535 | 11:05:03 |
| h:mm AM/PM | 134103 | 1:31 PM | 110535 | 11:05 AM |
| h:mm:ss AM/M | 134103 | 1:31:03 PM | 110535 | 11:05:03 AM |

#### Default Formats

The default formats in Excel all equate to their own number formats. For instance, if you click a cell with a number in it and click format you get the following screen.



If you select 2 decimal positions, use columns separators and show negative number in parenthesis in red, you are doing the same thing as entering “#,##0.00;[Red](#,##0.00)” as the data format. This format says to format the data with comma separators at every 3 characters, include 2 decimal positions and if it is negative do the same thing but add the color red and parenthesis.

[Appendix 3](#_Appendix_3_-) contains information on setting up data formats.

TODO create this: The following table includes the default options with example of the number formats to create them.

### Borders

There are a handful of arguments that affect how borders are displayed. Each border needs a style. Then the color can be set. Diagonal borders also have the option to select up, down or both. The following 3 sections discusses each of these options.

#### Border Styles

Border styles can be set using individual keys for each side or using the short cut key Border that can set them all at once. The following list shows what keys set the border styles, following that is a list of the supported styles.

* Border + 1 style, sets all borders to the same style.
* Border + 2 styles, sets the top and bottom to the first style and the right and left borders to the second style.
* Border + 4 styles, sets the Top, Bottom, Left and Right in that order
* BorderTop + style, sets the top border
* BorderBottom + style, sets the bottom border
* BorderLeft + style, sets the Left border
* BorderRight + style, sets the Right border
* BorderDiagonal + style, sets the style of the diagonal border, you must also set the diagonal option which is discussed in the next section.

The following is the list of available border styles

* DASH\_DOT
* DASH\_DOT\_DOT
* DASHED
* DOTTED
* DOUBLE
* HAIR
* MEDIUM
* MEDIUM\_DASH\_DOT
* MEDIUM\_DASH\_DOT\_DOT
* MEDIUM\_DASHED
* THIN
* NONE

Examples:

* Border:Thin – sets all border to thin
* Border:Medium:Thin – Sets the top and bottom border to Medium and the left and right border to thin.
* Border:None:Double:Thin:Thin – Sets no top border, a double bottom border and thin left and right borders.
* BorderTop:Dashed – sets the top border to dashed.
* BorderBottom:Dashed\_Dot – sets the bottom border to dashed\_dot.
* BorderLeft:Hair – sets the left border to hair.
* Borderright:Medium – sets the right border to medium.
* BorderDiagonal:Thin – sets the diagonal borders to thin, requires a diagonal option to also be set.

#### Diagonal Option

The Diagonal option sets which diagonal borders are shown. The BorderDiagonal style must also be set for this to work. The diagonal border allows turning on the downward diagonal border, the upward diagonal border or both. Both basically creates an x with the borders. The following are the three options you can use.

* DiagonalOption:None – not Diagonal borders are used
* DiagonalOption:Both – both diagonal borders are used
* DiagonalOption:Down – the diagonal border starting in the upper left corner going down to the lower right corner is used.
* DiagonalOption:Up – the diagonal border starting in the lower left corner and going up to the upper right corner is used.

#### Border Color

The border color property sets the color of the border. Excel only handles one color per cell, so all borders will be the same color. Colors can be passed using 3 different systems. The options are indexed, RBG or theme.

Indexed colors are referenced by index name. The names are basically just the color name, like “Red”. See the [Colors](#_Appendix_1_–) section of the appendix for the list of usable colors. The examples below use each of the color types.

* BorderColor:Red – The borders for the cell will be red.
* BorderColor:FFFF0000 – Same as red with RBG color.
* BorderColor:<TODO> - Theme color

### WrapText – Flags the cell to wrap text

Set the text to wrap in a cell. This mean that if the data in the cell does not fit it will wrap down to a new line and increase the height of the row. It accepts a boolean value, which equates to yes or no. All boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so On, On and on will all turn on the strike out option.

To turn on the text wrapping use any of these options:

* WrapText:On
* WrapText:\*on
* WrapText:Yes
* WrapText:y
* WrapText:1
* WrapText:True

The text wrapping option is off by default so there is no reason to pass this parameter if you do not want it on. However, if you have the need to pass the parameter anyways you can set it to off with the any of the following options.

* WrapText:Off
* WrapText:\*off
* WrapText:no
* WrapText:N
* WrapText:0
* WrapText:FALSE

### FillForegroundColor – Set the fill foreground color

The fill foreground color property sets the color of the fill. The fill foreground color is still the background color of the cell. However, there are 3 properties to set the fill, the foreground color background color and pattern. If you just want a solid color fill, you only need to use the FilleForegroundColor key. If you want fill pattern you must also designate the FillBackgroundColor and the FillPattern. Colors can be passed using 3 different systems. The options are indexed, RBG or theme.

Indexed colors are referenced by index name. The names are basically just the color name, like “Red”. See the [Colors](#_Appendix_1_–) section of the appendix for the list of usable colors. The examples below use each of the color types.

* BorderColor:Red – The borders for the cell will be red.
* BorderColor:FFFF0000 – Same as red with RBG color.
* BorderColor:<TODO> - Theme color

### FillBackgroundColor – Sets the fill back ground color

If using a Fill pattern, you must use the fill background color as well. The pattern is made using fill foreground and background colors. This key accepts one of the 3 colors documented in the FillForegroundColor section.

Example making a striped red and yellow background:

* FillBackgroundColor:RED
* FillForegroundColor:Yellow
* FillPattern:HorizontalStripe

### FillPattern – Sets the fill pattern

The fill pattern is used along with the fill foreground and background colors to make a pattern in the background of the cell. See the fill Background color section for an example.

There are 3 naming conventions for the fill pattern. One is the Excel names which are shown in Excel when selecting the pattern. The next is a list of names used when creating an XML file to open in Excel. These names were retained for conversions purposes. The third option is an index number used within the #$XLSX service program, the indexes can be used the same as the Excel names. The following table lists all the available options, each value in a row does the same thing.

|  |  |  |
| --- | --- | --- |
| Excel Name | XML Name | Index |
| None | None | 0 |
| Solid | Solid | 1 |
| 75%GRAY | LIGHTGRAY | 2 |
| 50%GRAY | MEDIUMGRAY | 3 |
| 25%GRAY | DARKGRAY | 4 |
| 12.5%GRAY | GRAY125 | 5 |
| 6.25%GRAY | GRAY0625 | 6 |
| HORIZONTALSTRIPE | DARKHORIZONTAL | 7 |
| VERTICALSTRIPE | DARKVERTICAL | 8 |
| REVERSEDIAGONALSTRIPE | DARKDOWN | 9 |
| DIAGONALSTRIPE | DARKUP | A |
| DIAGONALCROSSHATCH | DARKGRID | B |
| THICKDIAGONALCROSSHATCH | DARKTRELLIS | C |
| THINHORIZONTALSTRIPE | LIGHTHORIZONTAL | D |
| THINVERTICALSTRIPE | LIGHTVERTICAL | E |
| THINREVERSEDIAGONALSTRIPE | LIGHTDOWN | F |
| THINDIAGONALSTRIPE | LIGHTUP | G |
| THINCROSSHATCH | LIGHTGRID | H |
| THINDIAGONALCROSSHATCH | LIGHTTRELLIS | I |

## #$XLSXWkSh = Create a New Worksheet

* Must be done after #$XLSOPEN and before #$XLSCLOSE
* Closes the prior sheet if one is open
* Initialize worksheet data structure, add to workbook data structure sheet array.
* Allow changing properties until the first cell is written
* Properties include: AutoBreaks, Print Scale

TODO describe it

Options:

### Header/Footer – Set page headers or footers

The Header and Footer key options allow setting all options for the page header and footer. The key value of header will set header options and the key value of footer will set footer options. The parameters for both options are the same so they are included together in this documentation. All options will be listed for header, but every options also exists for the footer key.

Headers and footers get a little complicated because there can technically be up to 9 headers and 9 footers. This happens because the worksheet can be setup for different first page and different odd and even page headings. Those options are set using the DifferentOddAndEvenPages and DifferentFirstpage key words. Each heading and footing can also have a left, center and right option. This will be covered when discussing the First/Odd/Even option and the Left/Center/Right options.

The header and footer key allows the passing of up to 13 parameters. These are separated by semicolons. The option can have anywhere from 1 to 13 parameters and the order of the first 3 are fixed. The remaining four options just set specific options and can be in any order.

#### Parameter 1 – Text

The text parameter is where the value to be printed is passed. It can be any text you want. However if you need a semicolon anywhere in the text, the entire text string needs to be included in double quotes. If you actually want double quotes around the text then you can start and end the string with two double quotes, only the first and last one will be removed. If only one option is passed, all other values will be defaulted. Each additional value will list its’ default in the section that covers it.

The text also allows for special values. These values get replaced by a keyword in Excel that brings in the correct value. For example if you want page numbers you can use \*PAGE in the footer and in the Excel document the actual page number will be shown. The following table shows the special values you can use. Each value must be in uppercase.

|  |  |  |
| --- | --- | --- |
| Special Value | Description | Example |
| \*NAME | The name of the sheet | Sheet 1 |
| \*FILE | The name of the file | Physical Inventory.xlsx |
| \*PATH | The full path to the file | C:\Users\userName\Desktop\ |
| \*PAGE | The current page number | 2 |
| \*PAGES | The total number of pages | 10 |
| \*DATE | The current date | 2/12/2020 |
| \*TIME | The current time | 8:30 am |

Examples:

|  |  |
| --- | --- |
| Parameter | Heading |
| Header:Inventory Listing | Inventory Listing |
| Header:”Inventory Listing: \*date” | Inventory Listing: 10/23/2020 |
| Header:””Inventory Listing: \*date”” | “Inventory Listing: 10/23/2020” |
| Header:””Inventory Listing:” \*date” | “Inventory Listing:” 10/23/2020 |

#### Parameter 2 – Font

The font option allows you to use a specific font for the header or footer. It has to be in the second parameter. Fonts are passed as name. These are the names that are in the drop down list in Excel. They are case sensitive so Arial will work but arial will not. Remember that you may have custom fonts loaded that not everyone has. Be careful to only use common fonts like Arial, Times New Roman or Courier to ensure that the spreadsheet will look the same on everyone’s computer.

#### Parameter 3 – Font Size

This parameter allows you set the font size for the header or footer. The size is passed as an integer value between 1 and 409. Any other value will be ignored.

#### Parameter 4-13 – Options

After the first three parameters you can include up to 10 options. These options allow you set bold, italics, left/center/right position and first/odd/even page. These options are not case sensitive so BOLD, Bold and bold all do the same thing.

Bold and italic can be sent in any position and it will just make the header or footer bold or italicized.

Underline or DoubleUnderline can be sent to underline the entire text with a single or double underline.

Color can be passed, if it is, the actual color has to be passed in the very next parameter. For instance …:Color:Red:underline works but …:Color:underline:Red will create an error. See the [color section under styles](#_Color_–_Font) to see how colors can be passed.

Strikethrough can be passed to put a line through all the text.

SupperScript or Subscript can be passed to make all the text a supper script or subscript.

Left, Center or Right can be sent to dictate which location of the header or footer you are updating. Only one of these options can be passed. If not passed the location will be set to Center.

First, Odd or Even can be passed to dictate whether the heading or footing is displayed on the first page, or pages or even pages. Only one of these options can be passed. These options only work if you specify the [DifferentFirstpage](#_DifferentFirstpage_–_Set) and/or the [DifferentOddAndEvenPages](#_DifferentOddAndEvenPages_–_Set) option. The default value is odd, odd will be used on every page if neither of the different options are passed. The following table shows how the page Odd/Even/First options will be used based on the different options.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Different First | Different Odd/Even | First | Odd | Even | Description |
| Off | Off | N/A | All Pages | N/A | The same header and footer on all pages, no need to include the first, odd or even parameter. |
| Off | On | N/A | Odd Pages | Even Pages | Different header and footer on od and even pages. Each header or footer should include the Odd or Even option. |
| On | Off | First Page | All Other Page | N/A | The first page will have its’ own options, all other pages have the same header or footer. Each header or footer entry should contain the First or Odd option |
| On | On | First Page | Odd Pages | Even Pages | The headers on the first page, odd pages and even pages will be different. Each header and footer option should include either the First, Odd or Even option. Page 1 will be first, page 2 even, 3 odd, 4 even and so on. |

#### Example 1 – Simple heading and footing

This example assumes you just want a page heading centered on each page with the file name on the bottom left, a page number of pages on the bottom center and date at the top right. To make it pretty we are going to bold the title and make it bigger. The excel file name we are going to make smaller because it is just for reference.

These options would be used:

|  |  |
| --- | --- |
| Parameter | Description |
| Header:Inventory Report:Arial:24:Bold | Adds a header with the title “Inventory Report” in Arial Font, 24 point and bold. Defaults to Center |
| Header:\*DATE:Arial:12:Right | The \*DATE special value tells Excel to use the current date, the font is Arial, 12 point and the position is right. |
| Footer:\*FILE:Arial:8:Left | The file name will be included in the left footer in smaller font. |
| Footer:\*PAGE of \*PAGES:Arial:12 | The current page and number of pages will be listed like “1 of 10”, using Arial 12point and defaulting to the center position. |

#### Example 2 – Complex headings and footings

This example is to show about the most complex example that can be done. For this example we are turning on the DifferentFirstPage and the DifferentOddAndEvenPage options. The goal is to create a report that will have the headings and footing different on the first page then be assumed that it will printing duplexed and that we want the page numbers on the outside of each sheet which will be different on the odd and even pages.

All pages will have the title centered at the top, in larger font and bold. The first page will include the file path in the bottom left corner and the page 1 of XXX in the right corner. The odd pages will have the …

TODO finish this and test it. Waiting on new options so they can be added.

|  |  |
| --- | --- |
| Parameter | Description |
| DifferentOddAndEvenPages:True | Makes the header different for the odd and even pages. |
| DifferentFirstpage:True | Makes the headers and footers different for the first page. |
| Header:Inventory Report:Arial:24:Bold | Adds a header with the title “Inventory Report” in Arial Font, 24 point and bold. Defaults to Center |
| TODO Finish this. |  |
| Header:\*DATE:Arial:12:Right | The \*DATE special value tells Excel to use the current date, the font is Arial, 12 point and the position is right. |
| Footer:\*FILE:Arial:8:Left | The file name will be included in the left footer in smaller font. |
| Footer:\*PAGE of \*PAGES:Arial:12 | The current page and number of pages will be listed like “1 of 10”, using Arial 12point and defaulting to the center position. |

### DifferentOddAndEvenPages – Set headings and footing to be different for odd and even pages

This option sets the worksheet to use different headings and footings on odd and even pages. It accepts a boolean value, which equates to yes or no. All boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so On, On and on will all turn on the different for odd and even option.

To turn on the different for odd and even option use any of these options:

* DifferentOddAndEvenPages:On
* DifferentOddAndEvenPages:\*on
* DifferentOddAndEvenPages:Yes
* DifferentOddAndEvenPages:y
* DifferentOddAndEvenPages:1
* DifferentOddAndEvenPages:True

The different for odd and even option is off by default so there is no reason to pass this parameter if you do not want it on. However if you have the need to pass the parameter anyways you can set it to off with the any of the following options.

* DifferentOddAndEvenPages:Off
* DifferentOddAndEvenPages:\*off
* DifferentOddAndEvenPages:no
* DifferentOddAndEvenPages:N
* DifferentOddAndEvenPages:0
* DifferentOddAndEvenPages:FALSE

Setting the headings and footing based on this parameters is covered in the [Header/Footer](#_Header/Footer_–_Set) section.

### DifferentFirstpage – Set headings and footing to be different for the first page

This option sets the worksheet to use different headings and footings on the first page. It accepts a boolean value, which equates to yes or no. All boolean parameters in the system can accept any of these values for yes, Yes, Y, On, \*on, 1 or true and any of these values for no, No, N, off, \*off, 0 or False. The values are not case sensitive so On, On and on will all turn on the different first page option.

To turn on the different for first page option use any of these options:

* DifferentFirstpage:On
* DifferentFirstpage:\*on
* DifferentFirstpage:Yes
* DifferentFirstpage:y
* DifferentFirstpage:1
* DifferentFirstpage:True

The different for first page option is off by default so there is no reason to pass this parameter if you do not want it on. However, if you have the need to pass the parameter anyway you can set it to off with the any of the following options.

* DifferentFirstpage:Off
* DifferentFirstpage:\*off
* DifferentFirstpage:no
* DifferentFirstpage:N
* DifferentFirstpage:0
* DifferentFirstpage:FALSE

Setting the headings and footing based on this parameter is covered in the [Header/Footer](#_Header/Footer_–_Set) section.

### Margins – setting page margins

Margins can be set for the following options:

* LeftMargin
* RightMargin
* TopMargin
* BottomMargin
* HeaderMargin
* FooterMargin

They can be set individually using the options above or via the Margin keyword which is a shortcut to set multiple margins at the same time. Using the Margin keyword make coding margins easier. There are 5 ways to use the margin keyword. They are listed below.

All margins are in inches and handle 2 decimal positions. The program converts the text to a numeric value and formatting doesn’t really matter. For instance 0.3, 0.30, .3 and .30 will all do the same thing. The program does not validate the number passed, but Excel will get an error if the margins do not fit on the page. Since #$XLSX does not know paper sizes, there is no way to validate that a margin fits on a page, however a top limit of 21 inches has been set to ensure that the value is at least somewhat logical.

#### Single parameter

Sets all margins to the same size. This is useful if you just want to quickly set all margins to say a quarter of an inch. The following command does that is one step.

* Margin:0.25

This is the exact same as using all these keywords:

* LeftMargin:0.25
* RightMargin:0.25
* TopMargin:0.25
* BottomMargin:0.25
* HeaderMargin:0.25
* FooterMargin:0.25

Using the single parameter option makes for weird header and footer margins, but that is only an issue if headers and footers are being used. You can always issue a separate HeaderMargin keyword after the single parameter margin key word to override the header margins. For example the following two keywords sets all margins to 0.25 inches, but then sets the header margin to 1 inch.

* Margin:0.25
* HeaderMargin:1

This example works because keywords are always processed in arrival order, so the last keyword that updates a property is what the property will be.

#### Two parameters

The two parameter option sets the top and bottom margins and the left right margins. The first value will set the top and bottom margin and the second value will set the side margins.

#### Three parameters

Sets the top, right and left and the bottom margins. Right and left are both set to the middle parameter.

#### Four Parameters

Sets the top, right, bottom and left margins in that order.

#### Six parameters

Sets the top, right, bottom, left, header and footer margins in that order.

#### SideMargins

The SideMargin keyword can be used to set both the right and left margins at the same time.

#### Examples

The following table shows examples of the margins being used. The description column first lists the initial defaults, then each entry after that shows a Keyword and its values with how the margins will be changed from the defaults.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Description | Top | Right | Bottom | Left | Header | Footer |
| Initial defaults | 0.75 | 0.7 | 0.75 | 0.7 | 0.3 | 0.3 |
| LeftMargin:0.25 | 0.75 | 0.7 | 0.75 | 0.25 | 0.3 | 0.3 |
| RightMargin:0.25 | 0.75 | 0.25 | 0.75 | 0.7 | 0.3 | 0.3 |
| TopMargin:0.25 | 0.25 | 0.7 | 0.75 | 0.7 | 0.3 | 0.3 |
| BottomMargin:0.25 | 0.75 | 0.7 | 0.25 | 0.7 | 0.3 | 0.3 |
| HeaderMargin:1 | 0.75 | 0.7 | 0.75 | 0.7 | 1 | 0.3 |
| FooterMargin:0.5 | 0.75 | 0.7 | 0.75 | 0.7 | 0.3 | 0.5 |
| SideMargin:0.25 | 0.75 | 0.25 | 0.75 | 0.25 | 0.3 | 0.3 |
| Margin:0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |
| Margin:0.5:0.25 | 0.5 | 0.25 | 0.5 | 0.25 | 0.3 | 0.3 |
| Margin:0.5:0.25:0.3 | 0.5 | 0.25 | 0.3 | 0.25 | 0.3 | 0.3 |
| Margin:0.5:0.45:0.40:0.25 | 0.5 | 0.45 | 0.4 | 0.25 | 0.3 | 0.3 |
| Margin:0.1:0.2:0.3:0.4:0.5:0.6 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 |

### SheetName

Sets the name of the sheet. This is what is displayed in the tab at the bottom of the worksheet.

### AutoBreaks

Dictates whether to show the page margins. By default, page margins are not shown unless a print preview is done. Setting this option to Yes shows those breaks as soon as the sheet is open.

TODO make this work and test it.

### ColumnWidth

The column width option sets the width of a column. The Keyword requires two parameters, the first for the column number and the next for the width of that column. For example to set the second column to a width of 30 use: 'ColumnWidth:2:30'.

Since Excel labels columns with letter instead of numbers, you can also pass a letter or letter combination as the second parameter. For instance, the previous example used column 2, this could have also been sent as B. So 'ColumnWidth:2:30' and 'ColumnWidth:B:30' will do the exact same thing.

Excel labels columns A-Z, then AA-AZ, BA-BZ and so on. Column A=1, B=2,… Z=26, AA=27, AB=28, AZ=54 and so on.

TODO test the alpha columns.

TODO allow widths to be set in pixels or Excels width value, something like 'ColumnWidth:2:25Px' and 'ColumnWidth:2:9.75Width'

### Zoom

Set the zoom for the page. Requires one parameter for the zoom percent. The value must be between 10 and 500.

### Repeating Columns/Row

TODO Make this work

Set the number of columns or rows to repeat on each page. This is a printing feature. It allows for printing a header row or column on each page.

Either function accepts a single row/column or a range for rows/columns. For instance if you want the first 3 rows of a work sheet printed at the top of each page you can use ‘RepeatingRows:1:3’, if you only want the first row repeated you can use ‘RepeatingRows:1’.

### Freeze Columns

Sets the number of columns to freeze when scrolling left and right on a worksheet. It must include a number between 1 and the number of columns that fit on one screen. Since the size of columns is not known it is up to the programmer to figure out what works best here. Just be aware that funny things will happen if you freeze more columns than fit on the screen.

### Freeze Rows

Sets the number of rows to freeze when scrolling up and down in a spread sheet. This is normally used to keep column header displayed while rolling in a spread sheet. It accepts one parameter for the number of rows to freeze.

Example Data: 'FreezeRows:3'

### Print Orientation

Sets the page orientation to portrait or land scape. The default is portrait, and the only other option is landscape. You may pass an L or the word landscape to set it to land scape any other value will set it to portrait.

Examples:

* ‘PrintOrientation:LandScape’
* ‘PrintOrientation:L’
* ‘PrintOrientation:Portrait’
* ‘PrintOrientation:P’

### Print Scale

Sets the print scale. It must be a value from 1 to 400. It is a percent. So, 50 will print half the original size and 200 will print twice as big as normal.

Example: 'PrintScale:50' , 'PRINTSCALE:200'

### Filter

Auto filters can be turned on however there are some issues that can cause the spreadsheet to no longer work. If there are any merged cells in the sheet then the auto filters will cause Excel to return an error. Only use the filters if the sheet contains data and no merged cells.

Filters add the drop down arrow at the top to filter the data displayed. They exist across a header row and will allow filtering of the data below that row.

There are 2 ways to send the filter parameter. The first way is send the key Filter with 2 values, the first value is the starting cell and the second value is the ending cell. The other option is to send the key filter and one value. If one value is sent the starting cell will always be A1, so one value can only be used if the headers are in row 1.

Cells are passed in excel notation, so the second column of the third row is B3. No conversion will be done, and no error checking is currently done. If you pas anything wrong the workbook may not be usable.

Examples:

|  |  |
| --- | --- |
| Key |  |
| Filter:A4 | Filters will be added from A1 to A4 |
| Filter:B3:D3 | Filters will be added from C2 to C5, this should only be done if the data starts on row 4 |
| Filter:B4 | This will blow up because the starting cell will be A1 which is not in the same row as B4 |
| Filter:B2:A4 | This will also blow up because the filters are not in the same row. |

### Paper Size

This option will set the paper size for the work sheet. The default size is Letter. You can use any value from the drop down list in Excel. The value is case sensitive and must match what Excel has exactly. Some standard sizes are listed below.

|  |  |
| --- | --- |
| Size | Dimensions |
| Letter | 8.5” by 11” |
| Legal | 8.5” by 14” |
| Tabloid | 11” by 17” |
| Executive | 7.25 by 10.5 |

### Print area

Sets the area on a sheet that will be printed. The parameter requires 4 values, the first is starting column, the next is ending column, then starting row and ending row.

TODO come up with an example and test this.

### Print Quality

TODO

### Print Center Vertically

TODO

### Print Center Horizontally

TODO

### Print Page Order

TODO

## #$XLSXChar = add a character field to an Excel File

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data

TODO – finish documenting

## #$XLSXNumr = add a value field to Excel file

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data

TODO – finish documenting

## #$XLSXDate = add a date field to Excel file

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data
* Date Passed as a Date data type field

This function adds a date field to the next cell of the spreadsheet. It accepts a value passed as a date data type. If your program has dates stored in numeric formats try using #$XLSXYYMD, #$XLSXMDY or #$XLSXMDYY functions to add numeric date formats without having to convert them to date data types.

All data adding functions must be called after a spreadsheet has been started and a work sheet has been added to it. See the example programs for a walk through of how a spread sheet is created.

Once any of the add data type functions like this one are used, properties can no longer be changed on the open work sheet.

Parameters

1. Date, Required, passed as an RPG Date data type field
2. Style: Optional, Omittable, passed as a character string, max length 50 characters
3. Merge: Optional, Omittable, passed as a constant value

The Style parameter is a character field, normally passed as a literal string. This is used to apply styling options to the cell. While not required good practice is to always use a style on a date field. Excel stores dates as a numeric field where the number is the essential the number of days past 1900/01/01 this means by default they display as a number. The style has a property for a data format that tells Excel how to display the date. For instance a style with a data format setup like ‘DataFormat:yyyy/mm/dd’ will display a data like this 2021/01/01. If a style is not passed the date will look like this 44197, which not many people can understand.

The Merge parameter is used to merge this cell into a number of cells to the right. For instance a merge value of 1 will create a cell that spans 2 columns.

Examples:

* #$XLSYYMD(DATE:’S\_DATE’); // were DATE is a variable defined as a date data type

## #$XLSXYYMD = add a date field to Excel file

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data
* Date passed as a YYYYMMDD numeric field

This function was added to prevent date data from having to be converted in the end program before calling the #$XLSXDate function. The end result is the same as the #$XLSXDate function, it just converts the date from a decimal field to a date field first. Please review the [#$XLSXDate](#_#$XLSXDate__=) section of this document for information on the when and how this function can be used.

The input date needs to be a Dec(8,0) field and contain a date in the YYYYMMDD format. If the date is invalid an 0 date cell will be added and an error message will be logged.

Example usage:

* #$XLSYYMD(20210101:’S\_DATE’);
* #$XLSYYMD(DATE:’S\_DATE’); // were DATE is a variable containing the value

## #$XLSXMDY = add a date field to Excel file

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data
* Date Passed as a MMDDYY numeric field

This function was added to prevent date data from having to be converted in the end program before calling the #$XLSXDate function. The end result is the same as the #$XLSXDate function, it just converts the date from a decimal field to a date field first. Please review the [#$XLSXDate](#_#$XLSXDate__=) section of this document for information on the when and how this function can be used.

The input date needs to be a Dec(6,0) field and contain a date in the MMDDYY format. If the date is invalid an 0 date cell will be added and an error message will be logged.

Example usage:

* #$XLSMDY(010121:’S\_DATE’);
* #$XLSMDY(10121:’S\_DATE’); // since it is numeric the leading zero does nothing
* #$XLSMDY(DATE:’S\_DATE’); // were DATE is a variable containing the value

## #$XLSXMDYY = add a date field to Excel file

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data
* Date passed as a MMDDYYYY numeric field

This function was added to prevent date data from having to be converted in the end program before calling the #$XLSXDate function. The end result is the same as the #$XLSXDate function, it just converts the date from a decimal field to a date field first. Please review the [#$XLSXDate](#_#$XLSXDate__=) section of this document for information on the when and how this function can be used.

The input date needs to be a Dec(8,0) field and contain a date in the MMDDYYYY format. If the date is invalid an 0 date cell will be added and an error message will be logged.

Example usage:

* #$XLSMDYY(01012021:’S\_DATE’);
* #$XLSMDYY(1012021:’S\_DATE’); // since it is numeric the leading zero does nothing
* #$XLSMDYY(DATE:’S\_DATE’); // were DATE is a variable containing the value

## #$XL2Null = Skips a column, but does not create the cell

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data

TODO – finish documenting

## #$XLSXForm = add a formula field to a Excel File

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data

TODO – finish documenting

## #$XLSXNext = Start a new row

* Must be done after a sheet is opened, after sheet properties are set and before the sheet is closed.
* Constitutes writing cell data

TODO – finish documenting

## #$XLSXCell – Returns a Cell or Range Relative to the Current Cell

With no parameters, this just returns the current cell. Optional parameters will be passed in keyed pairs like the rest of the functions except these will have shortcuts so they fit easily into the #$XLSXForm function using string concatenation. The parameters are used for pulling relative cells or cell ranges relative to the current cell location.

The intent is to use these when creating functions. It prevents the need to keep track of current locations in a program. It also provides a benefit when rearranging a spreadsheet. If the spread sheet was hard coded and a column or row is added all the formulas would have to be reworked. When using relative locations, the formulas should remain intact.

Optional Parameter list (shortcut):

* Range(R) – Produces a range with both vertical and horizontal offsets and widths.
* VerticalRange(VR) – Produces a vertical range relative to current cell.
* HorizontalRange(HR) – Produces a horizontal range relative to the current cell.
* Offset(O) – Produces a single cell location with both a vertical and horizontal offset.
* VerticalOffset(VO) – Produces a single cell location relative to the current cell, moves back by an offset.
* HorizontalOffset(HO) – Produces a single cell location relative to the current cell, moves back by an offset.

**Note:** Generally only one parameter should ever be passed. See the multiple parameter section for information on multiple parameters at the same time and how to avoid doing it.

### Optional Parameters

#### Range, Shortcut R, Parameters (VR,HR,VO,HO)

The Range parameter can return a range that spans multiple rows and columns or a single cell address. This can be used to get the same results as HR or VR if you pass 0 for one of the one of the parameters. If the range values are left 0 it will return a single cell instead of a range. This option can be used to produce all possible results available from the #$XLSXCell function. It requires 4 parameters.

The parameters are:

1. Vertical Range
2. Horizontal Range
3. Vertical Offset
4. Horizontal Offset

The following table illustrates the parameters required to select the same area based on what cell you are entering the formula in. Each cell selects the values in the blue rectangle.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| 1 | Item | October Sales | November Sale | December Sale |  |  |
| 2 | Plain Widget | 6,543 | 5,897 | 7,613 | R:4:3:-3:1 | R:4:3:-3:2 |
| 3 | Green Widget | 2,654 | 3,197 | 2,765 |  |  |
| 4 | Yellow Widget | 3,564 | 2,493 | 3,195 |  |  |
| 5 | Brown Widget | 1,684 | 1,296 | 1,864 | R:4:3:0:1 | R:4:3:0:2 |
| 6 |  | R:4:3:1:-2 |  | R:4:3:1:0 | R:4:3:1:1 | R:4:3:1:2 |
| 7 |  | R:4:3:2:-2 |  | R:4:3:2:0 | R:4:3:2:1 | R:4:3:2:2 |
| 8 |  |  |  |  |  |  |

Notice the vertical and horizontal ranges are always 4 and 3. This is because no matter where the selection is going it is calculating a range that is 4 tall (vertical) and 3 wide (horizontal). The vertical and horizontal offsets change based on the position of the cell to the range that needs to be captured.

All vertical offsets use a positive number to go up in the spreadsheet. This was done because totals formulas are generally after the data. The goal is to count the number of detail rows written and use that number for the vertical range. For instance, there are 4 detail lines in this example, so the vertical range is 4.

Since the option in E2 is above the end of the data the vertical offset is entered as a negative. This is harder to produce when writing a spreadsheet because you would need to know the count of the detail lines before adding them to the spreadsheet, then you would have to include the formula only on the first line. This is not too hard if you are using SQL, but doing it with record level access means that you have to read through the data twice, once to get the count and then a second time to write out the data.

All Horizontal offsets are based on a positive number moving left in the spreadsheet. This is again done because horizontal totals are generally done at the end of rows. Again, you can use a negative offset to move right in the spreadsheet. Since columns are generally fixed in report type spreadsheets, using negatives for the horizontal offset is probably more likely than using them for the vertical offset.

Notice the offset is used to get the bottom right corner of a range. After that the upper right corner is based on the vertical and horizontal range. All the range options in this example will produce B2:D5, the range of the highlighted values.

To get just a vertical range you would leave the horizontal range value 0. For instance in cell C6 if you want to get the range C2:C5 for a total formula you could use R:4:0:1:0. This will use the offset to go up one cell for the ending range and then calculate back 4 rows for the starting cell. Since there is no horizontal offset or range all the column letters will remain C. This the same as doing a VR:4. If you wanted the total to be in row seven you would use R:4:0:2:0,. Since you are one more row down from the data you have to increase the offset by 1. This would be the same as doing a VR:4:1.

To get just a horizontal range only use the horizontal options, leave the vertical options 0. For example if you want to total across a row you can use r:0:3:0:1. Using this in E3 will result in B3:D3. This example offsets the range horizontally back one space and then does a range of 3. This can also be done using just HR:3 which defaults the vertical numbers to 0 and the horizontal offset to 1. If you want the same range in F3 for maybe an average formula you can use R:0:3:0:2 which just increases the offset by one since you are one more column away from the data. This would be the same as using HR:3:1.

If the parameters passed to range only produce a single cell, only a single cell will be returned. Single cells work for ranges in Excel, it will just assume the starting and end of the range is the same. For instance in the above example this formula =sum(B2) would produce 6,543. This can happen by accident if your data changes. For instance in the above example if the input data was changed to only include one row then the vertical range would be 1. If you had a total line at the end of the data for each column the range would now only produce a single cell so only the cell will be returned.

If you pass 0 for both the vertical and horizontal range offsets it will always return a single cell. The cell returned would be based on the offsets. Since the original function with no parameters already returns the current cell, passing R:0:0:0:0 would do the same thing. The offsets would be 0 and the range would be 0 leaving the result the current cell. Including either range value of 1 will produce the same results as a range of 1 still only produces one cell.

#### VerticalOffset, Shortuct VO, Parameters (VO)

The vertical offset parameter moves the position up a number of rows from the current cell. For instance if you are in cell G13 and you want G12 you can use VO:1. There is only one parameter and it is the number of rows to move back. This can be used in conjunction with the horizontal offset, but instead it is better to use the Offset parameter which allows passing both a vertical and horizontal offset. This can also be used with one of the range parameters, however it is not suggested as each range parameter offers an additional parameter to enter an offset. See the multiple parameter section for information on using this with other parameters.

#### HorizontalOffset, Shortuct HO, Parameters (HO)

The vertical offset parameter moves the position back a number of columns from the current cell. For instance if you are in cell G13 and you want F12 you can use HO:1. There is only one parameter and it is the number of columns to move back. This can be used in conjunction with the vertical offset, but instead it is better to use the Offset parameter which allows passing both a vertical and horizontal offset. This can also be used with one of the range parameters, however it is not suggested as each range parameter offers an additional parameter to enter an offset. See the multiple parameter section for information on using this with other parameters.

#### Offset, Shortcut O, Parameters (VO,HO)

This option produces a cell location with both a vertical and horizontal offset to the current cell. The format is O:1:2 where 1 is the vertical offset and 2 is the horizontal offset. The result is the same as passing both a vertical and horizontal offset, so O:1:2 is the same as passing VO:1 and HO:2. See the multiple parameter section for information on using this with other parameters.

#### VerticalRange, Shortcut VR, Parameters (VR, [VO])

The first parameter is a number of cells to include starting above the current cell. For instance if the current cell is c47 and you do a range:12 the function will return C35:C46.

The second parameter is optional. It is an offset to go up before starting the range. It is defaulted to one if it is not passed. This can be used if you skipped a line before the total. An example will be if you are in cell C47 and you skipped Cell C46 to leave a blank line before a total; if you do a VR:12:1 it will return C34:C45. Notice the beginning row also went back by one. This is because we expect the 12 parameter to be a count of rows in the detail which would not include the skipped row. A practical use of this function is if you wanted a total row then an average row. The total row could use VR:12 and the average row could use VR:12:2. This would return the same cell range since the average row would be one line further down then the total line. This could continue to be used for additional function if you just keep incrementing the second parameter.

#### HorizontalRange, Shortcut HR, Paramteres(HR, [Ho])

This will work similar to the vertical range except it is horizontal. It must be called with 1 or 2 parameters. It can be used to add total at the end of a column, or grand totals at the bottom by totaling multiple columns on the previous row.

The first parameter is required. It creates a range from previous columns to the current cell minus 1. For instance if you are currently adding to cell G14 if you use HR:5 it will return G9:G13

The second parameter is optional, it defaults to 0 if not passed. This parameter provides an offset for the range. Continuing the above example if you wanted a total to be skipped out a column you would use #$XLSXNull to skip over a cell leaving the current cell G15. The function would need to return the same range so you would need to offset the range by 1 from the current cell. Using HR:5:1 it would return G9:G13 just like before even though you are currently in G15.

#### Multiple Parameters

This section was included to show how multiple parameters work together. For the most part you should not be using multiple parameters. There is always a function that provides the exact parameter options you need. For instance the Range parameter allows entering all parameters together so it alone can be used to create any result.

Overall there are only 4 values ever passed to the function. They are vertical offset, horizontal offset, vertical range and horizontal range. Each parameter set passes one or more of these values and some default one of the values. When using additional parameters they overrides any value set on a previous parameter.

The system will return a range if the either of the vertical or horizontal range values is not 0 or 1, otherwise it will return a single cell location. The following example shows how the parameters can be used together.

**Using range to set every parameter**

Range can be used to set every parameter at once when using 4 parameters. For instance R:0:0:0:0 will set all parameters to 0 and be the exact same as not sending any parameters at all. This will also override any previous parameters sent. For instance if all these parameters are passed:

HorizontalRange:1

VO:2

Range:0:0:0:0

The end result will be all parameters equal 0 so only the current cell will be returned.

The following example shows how individual parameters can be used to set each value one at a time.

Using these parameters:

VerticalOffset:1

HorizontalOffset:2

VerticalRange:3

HorizontalRange:4

Is the exact same as:

Range(3:4:1:2)

Likewise passing a VerticalOffset and HorizontalOffset is the same passing them both in the Offset parameter.

The vertical and horizontal range options also default the corresponding offset to 1. This was done to make the functions easier to use. For instance you never want to enter a range that includes the current cell. This would create a circular reference which creates an error in Excel. The regular Range function does not do this so you should never pass a 0 in the third and fourth parameters if you are using it generate a range.

### Examples:

To illustrate how to use this function we will look at the following table representing an Excel sheet with totals. All the total cells will be detailed below showing how to use the #$XLSXCell in conjunction with #$XLSXForm to get the desired result.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| 1 | Item | October Sales | November Sale | December Sale | Quarter Totals | Month Average |
| 2 | Plain Widget | 6,543 | 5,897 | 7,613 | 20,053 | 6,684 |
| 3 | Green Widget | 2,654 | 3,197 | 2,765 | 8,616 | 2,872 |
| 4 | Yellow Widget | 3,564 | 2,493 | 3,195 | 9,252 | 3,084 |
| 5 | Brown Widget | 1,684 | 1,296 | 1,864 | 4,844 | 1,615 |
| 6 | All items | 14,445 | 12,883 | 15,437 | 42,765 | 14,255 |
| 7 | Item Average | 3,611 | 3,221 | 3,859 | 10,691 | 3,564 |
| 8 |  |  |  |  |  |  |
| 9 | Quarter 4 Totals | | |  |  |  |
| 10 | Total Sales: | | 41,049 |  |  |  |
| 11 | Monthly Averages: | | 3,421 |  |  |  |
| 12 |  | |  |  |  |  |

All data in rows 6 and 7 and columns E and F are calculated using formulas. The following sections each discuss what functions can be used to create the formulas. As a reminder the #$XLSXForm function is used to insert a function into a cell. In the RPG program we are going to assume we kept count of the number of detail lines added in a variable called COUNT.

Program #$XLSXE9 demonstrates this example. The test file can be generated using program #$XLSXE9PF.

#### Totals in B6-D6

These are just total functions formulas. They need to equal “=sum(B2:B5)” where B is the current column. Since the range needs to end at the previous row we can use the single parameter options of the vertical range parameter. I will use the shortcut VR to make the code smaller. The #$XLSXForm function already adds the equals sign so it is not needed.

#$XLSXForm('sum('+#$XLSXCell('VO:'+%char(COUNT))+ ')');

Now let’s break this down. The #$XLSXFunc function adds a function to a cell. For instance when we are in cell B6, meaning we just added the constant “All Items” to cell A6, we use the add formula function to insert the formula.

The parameter passed to the add function is 'sum('+#$xlxsCell('VR:'+%char(COUNT))+ ')'. This is three strings concatenated together. The first and last strings are just constants and the middle string is returned from the #$XLSXCell function. The parameter passed to #$XLSXCell is 'VR:4', getting the 4 from the COUNT variable. This is the shortcut parameter for VerticalRange. VerticalRange says to start with the current cell, go up one row and use the number of cells from the parameter. The result of #$XLSXCell will be B2:B5 making the full string equal 'sum(B2:B5)'. Since the #$XLSXCell function uses relative positioning the exact same code can be used in C6 and D6 as well.

#### Totals in E2-E5

These are just total functions formulas. They need to equal “=sum(D2:D5)” where D is the current row. Since the range needs to end at the previous row we can use the single parameter options of the horizontal range parameter. I will use the shortcut HR to make the code smaller. The #$XLSXForm function already adds the equals sign so it is not needed.

#$XLSXForm('sum('+#$XLSXCell('HR:3')+ ')');

Now let’s break this down. The #$XLSXForm function adds a formula to a cell. For instance when we are in cell E2, meaning we just added the value 7613 to cell D2, we use the add formula function to insert the formula.

The parameter passed the add function is 'sum('+#$xlsxCell('HR:3')+ ')'.This is three strings concatenated together. The first and last strings are just constants and the middle string is returned from the #$xlsxCell function. The parameter passed to #$xlsxCell is 'HR:3', the 3 is a constant because the report shows the months for one quarter so it will always be exactly 3 months. This is the shortcut parameter for HorizontalRange. HorizontalRange says to start with the current cell, go left one row and use the number of cells from the parameter. The result of #$xlsxCell will be B2:D2 making the full string equal 'sum(B2:D2)'. Since the #$XLSXCell function uses relative positioning the exact same code can be used in E3 through E5.

#### Total in E6

E6 can be entered using three different formulas. The obvious one is just adding the quarterly totals together with =sum(E2:E5). This option would use the exact same functions as B6-D6 and is definitely what this author would use. Another option would be to add the month totals together. This option would use the same formula used in E2-E5 and is an equally good option. However for the sake of making an example we will be using the formula =sum(B2:D5). This formula calculates the total from each individual detail entry. To get this formula we will need to use the range parameter of #$XLSXCell. We will use the R shortcut instead of spelling it out. The full line to create the formula is:

#$XLSXForm('sum('+#$XLSXCell('R:'+%char(COUNT)+':3:1:1'')+ ')');

To break it down we are just going to look at the #$XLSXCell function which needs to return B2:D5. The COUNT variable is 4 so the parameter passed is R:4:3:1:1. This calculates the Range based on 4 rows, 3 columns and a vertical and horizontal offset of 1. The Horizontal Range value of 4 says to include 4 rows and Vertical Range value of 3 tells it to use 3 columns. The offsets are important here because we need the whole range to be up one row and left one column from the current cell.

#### Averages in B7-D7

TODO

#### Average in E7

TODO

#### Averages in F2-F5

TODO

#### Average in F6

TODO

#### Average in F7

TODO

#### Reason to use relative positions

You may have noticed that this example doesn’t have a report title. So when you get everything calculating the way you want you need to add one and probably a line for Quarter Ending 2021. Had you used manual formula calculations every formula would have to be manually changed. With relative formulas you can add the headers and Item number columns and all your formulas will shift with them. In this example you may still have to slightly modify the totals at the bottom since they do not line up with the detail but not if you used the save cell method of duplicating the totals.

The source code for this example is in member #$XLSXE9. There is an indicator for ShowTitle, you can try running it with it on and off. It demonstrates that the formulas all work the same even when you add extra rows and columns.

## #$XLSXC = Shortcut for #$XSLSCell

After using the #$XLSXCell function in some programs I realized that the length of the command and BIF’s required for building the input string did not fit nicely in the programs.

For example look at this command from one of the examples above:

#$XLSXForm('sum('+#$XLSXCell('R:'+%trim(%char(COUNT))+':3:1:1'')+ ')');

You can see that 2 BIF’s are required to convert the COUNT variable into the input string for the call to #$XLSXCell. After doing this on a whole lot of formula entries in a complex spreadsheet I thought I could make it a little easier. My solution was to create #$XLSXC function that allows passing the parameters separately and then builds the text string and calls #$XLSXCell. This function can be called with 0 to 5 parameters. Calling it 0 parameters is the same as calling #$XLSXCell with no parameters.

The first parameter is the Text option from #$XLSXCell. For instance it can be R, Range, Vo, VerticalOffset and so on. The next 4 optional parameters are the number values that would follow the first option. The procedure just concatenates the parameters together and calls the #$XLSCell function. The primary benefit is that the counts can be passed as number instead having to be converted to a trimmed character string.

The example above can now be written as:

#$XLSXForm('sum('+#$XLSXC('R’:COUNT:3:1:1)+ ')');

In this example #$XLSXC will just concatenate the 5 input fields to the character string ‘R:35:2:1:1’ and call #$XLSXCell, assuming the value of count is 35.

## #$XLSXClose = Close the open Excel File

* Must be done last, nothing else can be done after this is run
* Closes the last sheet if still open
* Writes out all files
* Zips the files into the final excel file
* Deletes the temporary folder.

# #$ZIP Service Program

The #$ZIP service program is used by #$XLSX to do the data compression. An Excel .XLSX file is just a zip archive of XML files. The #$ZIP service program streams data directly into a zip archive handling the compression as it goes. It is documented here in case anyone needs to use it separately for their own project.

There are only 5 procedures exported from the service program. The following tables lists each one with a brief definition of their function.

|  |  |
| --- | --- |
| Procedure | Function |
| #$ZipOpen | Open/create a zip file |
| #$ZipFile | Add file inside a zip file |
| #$ZipWrite | Write data into a file inside a zip file |
| #$ZipWriteB | Same as write but for a buffer |
| #$ZipClose | Close an open zip file |

The #$zip file increases performance drastically over writing out files and then zipping them. There is a lot of overhead when writing files to the IFS. It has to convert the data from EBCDIC to ASCII and move the data from memory to the physical hard drive. If you write out a non-zipped file then use a zip program you are essentially triplicating the work. This is because the data is moved from RAM to disk, then from Disk to RAM and then back to disk. Also, since the data being written and reread from the disk is in an un-compressed format those steps require a lot more resources than just writing the compressed data directly to disk.

All prototypes and constants are defined in member #$ZIP\_H. This needs to be copied or included in your program to use the functions. The service program is in the bound directory #$ZIP so it can be used by including BNDDIR('#$ZIP') on an H spec in fixed format or as a Ctl-Opt in free format.

## Example Programs

There are two example programs that are also used for testing. The programs have comments to show what they are doing, so they will not be included here. The source for the programs are #ZIPTS and #$ZIPTS2. If you want the simplest example start with #ZIPTS2 it only writes out one field to the zip file.

## Ordering

The ordering of the functions is important. Since all data is being streamed directly into the Zip file they must be done in the following order or it will not work.

1. #$ZipOpen – Open the Zip file. This function must be called first. If any other function is called before a file is opened an exception message will be issued.
2. #$ZipFile - Add file inside a zip file. This must be done after the archive is opened and before any data is written to the file. If an existing file is already opened in the zip file, it is closed and a new file is added.
3. #$ZipWrite/#ZipWriteB - Write data into a file inside a zip file. This must be done after a file in the Zip Archive is opened.
4. #$ZipClose - Close an open zip file archive. This must be done last. If it is not done the zip will not be useable.

## #$ZipOpen – Open the Zip file

This function must be called first. If any other function is called before a file is opened an exception message will be issued. This procedure creates, overrides or appends to an existing zip file in the IFS.

Parameters

1. File – the path and name of the zip file. Required, max length 1024, passed as a constant.
2. AppendOption – Optional int(10) defaults to Create/Replace
3. Compression – Optional int(1), options 1-9, 1 is no compression, 9 is highest compression. Defaults to 5 if not passed. Higher compression levels take longer to work, lower levels are faster but create a larger file.

Append Options

The append option is passed as a numeric value, but there are constants setup for each option. The constants are defined in the #$Zip\_h include file. The following table describes each option.

|  |  |  |
| --- | --- | --- |
| Value | Constant | Description |
| 0 | APPEND\_CREATE | Creates the file, if the file already exists an error message is thrown |
| 1 | APPEND\_CREATEAFTER | Not sure, never used it |
| 2 | APPEND\_ADDINZIP | Opens an existing zip archive so additional files can be added to it. The file must exist or an exception message will be sent. |
| 3 | APPEND\_CREATE\_REPLACE | Creates a new zip file and replaced the existing one if there is one. This is the default. |

If the append option needs to update or replace an existing file and you do not have authority to the file or the file is locked and exception error will be issued.

## #$ZipFile - Add file inside a zip file

This has to be done after the archive is opened and before any data is written to the file. If an existing file in the zip archive is already opened it is closed and a new file is added.

This procedure only accepts one parameter and it is the file name with path. The max length is 1024 character.

Files in the zip archive can be at the root level or in a folder. To create them in a folder simply add the folder to the path. The following table shows examples.

|  |  |
| --- | --- |
| Parameter | Result |
| Test.txt | This file will be in the root directory meaning when you open the archive you will see it. |
| Folder/Test.txt | This file will be a folder called folder. When you open the zip archive you will see a directory called folder, when you open it you will see the file Test.txt |
| Folder/Test/Test.txt | This file will be in the folder Folder, then in a sup folder named Test. |

## #$ZipWrite - Write data into a file inside a zip file

This has to be done after a file in the Zip Archive is opened.

This procedure only accepts one parameter. It is varying length field with a max size of 32,766. It is the data to write into the opened file. You can pass whatever kind of date you want but it must be in character field. For instance if want to add a number use the %char or one of the %Edit functions to convert it to character data.

If you have a pointer to data use the #$ZipWriteB procedure.

## #ZipWriteB - Same as write but for a buffer

This has to be done after a file in the Zip Archive is opened.

This function works like #$ZipWrite but it pulls the data from a memory location using a pointer and length. If you already have a pointer to your data this is more efficient than using #$ZipWrite.

Parameters

1. Pointer – An RPG pointer to a memory location. Required.
2. Length – The length of data to stream into the opened zip file. Required.

This is quicker because the C function in the underlying C service program that uses the ZLib package only accepts a pointer and length. When you use #$ZipWrite it accepts the data, moves it into a temporary field and passes the pointer and length of the temporary field. This is not a lot of overhead and makes calling the function from an RPG program easier if your data is in a variable, already so use whatever works best for you.

The following example code shows how the #$ZipWrite and #$ZipWriteB functions can each be used to write the same data in variable to the open file.

C\*

C MOVE 'DATA' DATA 10

C CallP #$ZipWrite(DATA)

C CallP #$ZipWriteB(%addr(DATA):%len(%trimr(DATA)))

C\*

You can see how using #$ZipWrite is easier to code than the #$ZipWriteB procedure.

## #$ZipClose - Close an open zip file archive

This has to be done last. If it is not done the zip will not be useable.

There are no parameters to this command.

# Appendix 1 – Colors

The color system allows for enumerated numbers these are generally the colors in the quick drop down box in Excel. They tend to be easier to use because the #$XLSX system has constants defined for them. For instance number 10 is for red and there is constant defined as RED that contains the number 10. This means that you can pass RED as a color and it will work.

The following table contains all the enumerated colors:

|  |  |  |
| --- | --- | --- |
| Name | Index | RGB |
| AQUA | 49 | 33CCCC |
| BLACK | 0,8 | 000000 |
| BLUE | 12, 4, 39 | 0000FF |
| BLUE\_GREY | 54 | 666699 |
| BRIGHT\_GREEN | 11, 3 | 00FF00 |
| BROWN | 60 | 993300 |
| CORAL | 29 | FF8080 |
| CORNFLOWER\_BLUE | 24 | 9999FF |
| DARK\_BLUE | 18, 32 | 000080 |
| DARK\_GOLD | 51 | FFCC00 |
| DARK\_GREEN | 58 | 003300 |
| DARK\_RED | 16, 37 | 800000 |
| DARK\_TEAL | 46 | CC99FF |
| DARK\_YELLOW | 19 | 808000 |
| GREEN | 17 | 008000 |
| GREY\_25 | 22 | C0C0C0 |
| GREY\_40 | 55 | 969696 |
| GREY\_50 | 23 | 808080 |
| GREY\_80 | 63 | 333333 |
| INDIGO | 62 | 333399 |
| LAVENDER | 41 | FFCCFF |
| LEMON\_CHIFFON | 26 | FFFFCC |
| LIGHT\_BLUE | 48 | 3366FF |
| LIGHT\_CORNFLOWER\_BLUE | 31 | CCCCFF |
| LIGHT\_GREEN | 42 | CCFFCC |
| LIGHT\_ORANGE | 52 | FF9900 |
| LIGHT\_TURQUOISE | 27 | CCFFFF |
| LIGHT\_YELLOW | 43 | FFFF99 |
| LIME | 50 | 99CC00 |
| MAROON | 25 | 993366 |
| NAVY | 56 | 003366 |
| OLIVE\_GREEN | 59 | 333300 |
| ORANGE | 53 | FF6600 |
| ORCHID | 28 | 660066 |
| PALE\_BLUE | 44 | 99CCFF |
| PINK | 14, 6, 33 | FF00FF |
| PLUM | 61 | 993366 |
| RED | 10, 2 | FF0000 |
| ROSE | 45 | FF99CC |
| ROYAL\_BLUE | 30 | 0066CC |
| SEA\_GREEN | 57 | 339966 |
| SKY\_BLUE | 40 | 00CCFF |
| TAN | 47 | FFCC99 |
| TEAL | 21, 38 | 008080 |
| TURQUOISE | 15, 7, 35 | 00FFFF |
| VIOLET | 20, 36 | 800080 |
| WHITE | 1,9 | FFFFFF |
| YELLOW | 13, 5, 34 | FFFF00 |

# Appendix 2 – Theme Color constants

TODO – None of the style options work yet. This was just added because I saw them listed somewhere

<accent1> (Accent 1 Theme Color Reference)

Specifies a reference to the <accent1> theme color in the document's Theme part.

<accent2> (Accent 2 Theme Color Reference)

Specifies a reference to the <accent2> theme color in the document's Theme part.

<accent3> (Accent 3 Theme Color Reference)

Specifies a reference to the <accent3> theme color in the document's Theme part.

<accent4> (Accent4 Theme Color Reference)

Specifies a reference to the <accent4> theme color in the document's Theme part.

<accent5> (Accent5 Theme Color Reference)

Specifies a reference to the <accent5> theme color in the document's Theme part.

<accent6> (Accent 6 Theme Color Reference)

Specifies a reference to the <accent6> theme color in the document's Theme part.

<dark1> (Dark 1 Theme Color Reference)

Specifies a reference to the <dk1> theme color in the document's Theme part.

<dark2> (Dark 2 Theme Color Reference)

Specifies a reference to the <dk2> theme color in the document's Theme part.

<followedHyperlink> (Followed Hyperlink Theme Color Reference)

Specifies a reference to the <folHlink> theme color in the document's Theme part.

<hyperlink> (Hyperlink Theme Color Reference)

Specifies a reference to the <hlink> theme color in the document's Theme part.

<light1> (Light 1 Theme Color Reference)

Specifies a reference to the <lt1> theme color in the document's Theme part.

<light2> (Light 2 Theme Color Reference)

Specifies a reference to the <lt2> theme color in the document's Theme part.

# Appendix 3 - Data Format rules

These are copied from some Microsoft docs found on the internet. It is a lot of data, but if you ever need to figure something out specifically you can probably find out how to do it here.

Up to four sections of format codes can be specified. The format codes, separated by semicolons, define the formats for positive numbers, negative numbers, zero values, and text, in that order. If only two sections are specified, the first is used for positive numbers and zeros, and the second is used for negative numbers. If only one section is specified, it is used for all numbers. To skip a section, the ending semicolon for that section shall be written.

1. The first section, "Format for positive numbers", is the format code that applies to the cell when the cell value contains a positive number.
2. The second section, "Format for negative numbers", is the format code that applies to the cell when the cell value contains a negative number.
3. The third section, "Format for zeros", is the format code that applies to the cell when the cell value is zero.
4. The fourth, and last, section, "Format for text", is the format code that applies to the cell when the cell value is text.

The & (ampersand) text operator is used to join, or concatenate, two values.

The following list describes the different symbols that are available for use in custom number formats.

0 - Digit placeholder.

**Example:** If the value 8.9 is to be displayed as 8.90, use the format #.00

#-Digit placeholder.

This symbol follows the same rules as the 0 symbol. However, the application shall not display extra zeros when the number typed has fewer digits on either side of the decimal than there are # symbols in the format.

**Example:** If the custom format is #.##, and 8.9 is in the cell, the number 8.9 is displayed.

?-Digit placeholder.

This symbol follows the same rules as the 0 symbol. However, the application shall put a space for insignificant zeros on either side of the decimal point so that decimal points are aligned in the column.

**Example:** The custom format 0.0? aligns the decimal points for the numbers 8.9 and a column.

. (period)-Decimal point.

%-Percentage.

If the cell contains a number between 0 and 1, and the custom format 0% is used, the application shall multiply the number by 100 and add the percentage symbol in the cell.

, (comma)-Thousands separator.

The application shall separate thousands by commas if the format contains a comma that is enclosed by number signs (#) or by zeros. A comma that follows a placeholder scales the number by one thousand.

**Example:** If the format is #.0,, and the cell value is 12,200,000 then the number 12.2 is displayed. end example

E- E+ e- e+ -Scientific format.

The application shall display a number to the right of the "E" symbol that corresponds to the number of places that the decimal point was moved.

**Example:** If the format is 0.00E+00, and the value 12,200,000 is in the cell, the number 1.22E+07 is displayed. If the number format is #0.0E+0, then the number 12.2E+6 is displayed.

$-+/():space - Displays the symbol.

If it is desired to display a character that differs from one of these symbols, precede the character with a backslash (\). Alternatively, enclose the character in quotation marks.

**Example:** If the number format is (000), and the value 12 is in the cell, the number (012) is displayed.

\ - Displays the next character in the format.

The application shall not display the backslash.

**Example:** If the number format is 0\!, and the value 3 is in the cell, the value 3! is displayed.

\* - Repeats the next character in the format enough times to fill the column to its current width.

There shall not be more than one asterisk in one section of the format. If more than one asterisk appears in one section of the format, all but the last asterisk shall be ignored.

**Example:** if the number format is 0\*x, and the value 3 is in the cell, the value 3xxxxxx is displayed. The number of x characters that are displayed in the cell varies based on the width of the column.

\_ (underline) - Skips the width of the next character.

This is useful for lining up negative and positive values in different cells of the same column.

**Example:** The number format \_(0.0\_);(0.0) aligns the numbers 2.3 and the column even though the negative number is enclosed by parentheses.

"text" - Displays whatever text is inside the quotation marks.

**Example:** The format 0.00 "dollars" displays 1.23 dollars when the value 1.23 is in the cell.

@ - Text placeholder.

If text is typed in the cell, the text from the cell is placed in the format where the at symbol (@) appears.

**Example:** If the number format is "Bob "@" Smith" (including quotation marks), and the value "John" is in the cell, the value Bob John Smith is displayed.

## Text and spacing

### Display both text and numbers

To display both text and numbers in a cell, enclose the text characters in double quotation marks (" ") or precede a single character with a backslash (\). Single quotation marks shall not be used to denote text. Characters inside double quotes, or immediately following backslash shall never be interpreted as part of the format code lexicon; instead they shall always be treated as literal strings. Remember to include the characters in the appropriate section of the format codes.

**Example:** Use the format $0.00" Surplus";$-0.00" Shortage" to display a positive amount as "$125.74 Surplus" and a negative amount as "$-125.74 Shortage."

The following characters are displayed without the use of quotation marks.

* $ - Dollar sign
* - - Minus sign
* + - Plus sign
* / - Slash mark
* ( - Left parenthesis
* ) - Right parenthesis
* : - Colon
* ! - Exclamation point
* ^ - Circumflex accent (caret)
* & - Ampersand
* ' - Apostrophe
* ~ - Tilde
* { - Left curly bracket
* } - Right curly bracket
* < - Less-than sign
* > - Greater-than sign
* = - Equal sign
* Space character

### Include a section for text entry

If included, a text section shall be the last section in the number format. Include an "at" sign (@) in the section, precisely where the cell’s text value should be displayed. If the @ character is omitted from the text section, text typed in the cell will not be displayed. To always display specific text characters with the typed text, enclose the additional text in double quotation marks (" ").

**Example:** If “June” is typed into the cell, and the text format is "gross receipts for "@ , then the cell will display “gross receipts for June”.

If the format does not include a text section, text entered in a cell is not affected by the format code.

### Add spaces

To create a space that is the width of a character in a number format, include an underscore, followed by the character.

**Example:** When an underscore is followed with a right parenthesis, such as \_), positive numbers line up correctly with negative numbers that are enclosed in parentheses because positive numbers are displayed with a blank space after them exactly the width of the right parenthesis character.

### Repeat characters

To repeat the next character in the format to fill the column width, include an asterisk (\*) in the number format.

**Example:** Use 0\*- to include enough dashes after a number to fill the cell, or use \*0 before any format to include leading zeros.

## Decimal places, spaces, colors, and conditions

### Include decimal places and significant digits

To format fractions or numbers with decimal points, include the following digit placeholders in a section. If a number has more digits to the right of the decimal point than there are placeholders in the format, the number rounds to as many decimal places as there are placeholders. If there are more digits to the left of the decimal point than there are placeholders, the extra digits are displayed. If the format contains only number signs (#) to the left of the decimal point, numbers less than 1 begin with a decimal point.

# (number sign) displays only significant digits and does not display insignificant zeros.

0 (zero) displays insignificant zeros if a number has fewer digits than there are zeros in the format.

? (question mark) adds spaces for insignificant zeros on either side of the decimal point so that decimal points align when they are formatted with a fixed-width font, such as Courier New. ? can also be used for fractions that have varying numbers of digits.

|  |  |  |
| --- | --- | --- |
| To display | As | Use this code |
| 1234.59 | 1234.6 | ####.# |
| 8.9 | 8.900 | #.000 |
| .631 | 0.6 | 0.# |
| 121234.568 | 12.01234.57 | #.0# |
| 44.398102.652.8 | 44.398102.65 2.8(with aligned decimals) | ???.??? |
| 5.255.3 | 5 1/45 3/10(with aligned fractions) | # ???/??? |

### Display a thousands separator

To display a comma as a thousands separator or to scale a number by a multiple of 1,000, include a comma in the number format.

|  |  |  |
| --- | --- | --- |
| To display | As | Use this code |
| 12000 | 12,000 | #,### |
| 12000 | 12 | #, |
| 12200000 | 12.2 | 0.0,, |

### Specify colors

To set the text color for a section of the format, use the name of one of the following eight colors in square brackets in the section. The color code shall be the first item in the section.

* [Black]
* [Blue]
* [Cyan]
* [Green]
* [Magenta]
* [Red]
* [White]
* [Yellow]

Instead of using the name of the color, the color index can be used, like this [Color3] for Red. Numeric indexes for color are restricted to the range from 1 to 56, which reference by index to the legacy color palette.

### Specify conditions

To set number formats that are applied only if a number meets a specified condition, enclose the condition in square brackets. The condition consists of a comparison operator and a value. Comparison operators include: = Equal to; > Greater than; < Less than; >= Greater than or equal to, <= Less than or equal to, and <> Not equal to.

**Example:** The following format displays numbers that are less than or equal to a red font and numbers that are greater than a blue font.

[Red][<=100];[Blue][>100]

If the cell value does not meet any of the criteria, then pound signs ("#") are displayed across the width of the cell.

## Currency, percentages, and scientific notation

### Include currency symbols

To include currency symbols, place the currency symbol in the location it should when displayed.

### Display percentages

To display numbers as a percentage of 100 include the percent sign (%) in the number format.

**Example:** To display .08 as 8% or 2.8 as 280%

### Display scientific notations

To display numbers in scientific format, use exponent codes in a section.

**Example:** E-, E+, e-, or e+

If a format contains a zero (0) or number sign (#) to the right of an exponent code, the application displays the number in scientific format and inserts an "E" or "e". The number of zeros or number signs to the right of a code determines the number of digits in the exponent. "E-" or "e-" places a minus sign by negative exponents. "E+" or "e+" places a minus sign by negative exponents and a plus sign by positive exponents.

## Dates and times

### Display days, months, and years

|  |  |  |
| --- | --- | --- |
| To display | As | Use this code |
| Months | 1–12 | M |
| Months | 01–12 | Mm |
| Months | Jan–Dec | Mmm |
| Months | January–December | Mmmm |
| Months | J–D | mmmmm |
| Days | 1–31 | D |
| Days | 01–31 | Dd |
| Days | Sun–Sat | Ddd |
| Days | Sunday–Saturday | Dddd |
| Years | 00–99 | Yy |
| Years | 0000-9999 | Yyyy |

### Month versus minutes

If "m" or "mm" code is used immediately after the "h" or "hh" code (for hours) or immediately before the "ss" code (for seconds), the application shall display minutes instead of the month.

### Display hours, minutes, and seconds

|  |  |  |
| --- | --- | --- |
| To display | As | Use this code |
| Hours | 0–23 | h |
| Hours | 00–23 | hh |
| Minutes | 0–59 | m |
| Minutes | 00–59 | mm |
| Seconds | 0–59 | s |
| Seconds | 00–59 | ss |
| Time |  | h AM/PM |
| Time |  | h:mm AM/PM |
| Time | 4:36:03 P | h:mm:ss A/P |
| Time | 4:36:03.75 | h:mm:ss.00 |
| Elapsed time (hours and minutes) | 1:02 | [h]:mm |
| Elapsed time (minutes and seconds) | 62:16 | [mm]:ss |
| Elapsed time (seconds and hundredths) | 3735.80 | [ss].00 |

### Minutes versus month

The "m" or "mm" code shall appear immediately after the "h" or "hh" code or immediately before the "ss" code; otherwise, these will display as the month instead of minutes.

### AM and PM

If the format contains AM or PM, the hour is based on the 12-hour clock, where "AM" or "A" indicates times from until and "PM" or "P" indicates times from until midnight. Otherwise, the hour is based on the 24-hour clock.

### Illegal date and time values

Cells formatted with a date or time format and which contain date or time values which do not meet the requirements specified shall show the pound sign ("#") across the width of the cell.