**An APL-Excel Interface:
Reference Manual**

Author: David Crossley

Date: September 22, 2011

© David Crossley 2010, 2011

davidwfc@gmail.com

Table of Contents

[Toolkit Namespace 4](#_Toc304487833)

[Function Details 4](#_Toc304487834)

[Notes on function arguments: 4](#_Toc304487835)

[R 🡨 active ’’ 4](#_Toc304487836)

[addsheet ω 4](#_Toc304487837)

[α bcolour ω 5](#_Toc304487838)

[{α} bold ω 5](#_Toc304487839)

[R 🡨 books ’’ 5](#_Toc304487840)

[{α} border ω 6](#_Toc304487841)

[{α} border\_all ω 6](#_Toc304487842)

[R 🡨 cellrange ω 6](#_Toc304487843)

[centre ω 6](#_Toc304487844)

[R 🡨charts ω 7](#_Toc304487845)

[R 🡨 check ω 7](#_Toc304487846)

[R 🡨clean ω 7](#_Toc304487847)

[clear ω 7](#_Toc304487848)

[{α} close ω 7](#_Toc304487849)

[R 🡨 copysheet ω 8](#_Toc304487850)

[R 🡨 {α} dateToDaynum ω 8](#_Toc304487851)

[R 🡨 {α} daynumToDate ω 9](#_Toc304487852)

[deletesheet ω 9](#_Toc304487853)

[R 🡨 dim ‘’ 9](#_Toc304487854)

[endexcel ‘’ 9](#_Toc304487855)

[α font ω 9](#_Toc304487856)

[α fontcolour ω 10](#_Toc304487857)

[α format ω 10](#_Toc304487858)

[{α} frame ω 10](#_Toc304487859)

[freeze ω 11](#_Toc304487860)

[help 11](#_Toc304487861)

[α indent ω 11](#_Toc304487862)

[R 🡨 α lastcell ω 11](#_Toc304487863)

[ljust ω 11](#_Toc304487864)

[R 🡨 lc ω 11](#_Toc304487865)

[R 🡨 {α} open ω 11](#_Toc304487866)

[R 🡨 {α} plot ω 12](#_Toc304487867)

[quitexcel ‘’ 17](#_Toc304487868)

[R 🡨 range ω 17](#_Toc304487869)

[R 🡨 rc ω 18](#_Toc304487870)

[R 🡨 rd ω 18](#_Toc304487871)

[R 🡨 read ω 19](#_Toc304487872)

[R 🡨 readf ω 19](#_Toc304487873)

[R 🡨 readt ω 19](#_Toc304487874)

[rename ω 19](#_Toc304487875)

[rjust ω 20](#_Toc304487876)

[{α} saveas ω 20](#_Toc304487877)

[R 🡨 setbook ω 21](#_Toc304487878)

[R 🡨 setsheet ω 22](#_Toc304487879)

[R 🡨 sheets ω 22](#_Toc304487880)

[R 🡨 {α} startexcel ω 22](#_Toc304487881)

[R 🡨 usedrange ω 23](#_Toc304487882)

[{α} wrap ω 23](#_Toc304487883)

[α write ω 23](#_Toc304487884)

[α writef ω 24](#_Toc304487885)

[R 🡨 α xread ω 24](#_Toc304487886)

[α xwrite ω 24](#_Toc304487887)

[Excel Properties and Methods 25](#_Toc304487888)

[xl – The Application Object 25](#_Toc304487889)

[Excel WorkSheet Functions 25](#_Toc304487890)

[Collection Objects 26](#_Toc304487891)

[wbk – A Workbook Object (returned by *setbook* and *open*) 27](#_Toc304487892)

[sh – A Sheet Object (returned by *setsheet*) 27](#_Toc304487893)

[rs – A Range or Selection Object 28](#_Toc304487894)

[Appendix A 28](#_Toc304487895)

[Appendix B 29](#_Toc304487896)

[Appendix C: *xread* and *xwrite* 32](#_Toc304487897)

[Summary 32](#_Toc304487898)

[Some Simple Examples 33](#_Toc304487899)

[Specifying a Cell or Cell Range 34](#_Toc304487900)

[More Examples 34](#_Toc304487901)

# Toolkit Namespace

For the purpose of the examples presented in this document, the toolkit functions are assumed to reside in the namespace #.XL. If you include #.XL in []PATH, then there is no need to qualify the function names (unless there’s a name conflict) even if your application is in another namespace. However, *startexcel* creates the global objects *xl* and *wb* in #.XL, and so you must qualify those names whenever you refer to them.

If you use a namespace other than #.XL for the toolkit it has no impact on the toolkit itself other than how you qualify *xl* and *wb*. The name of the toolkit namespace is determined dynamically when needed (in the function *endexcel*).

# Function Details

## Notes on function arguments:

1. Unless stated otherwise, “cell range” means a single cell or a cell range. It can be specified as a string (case-insensitive) or as numerical row-column co-ordinates. Blanks in a string are ignored. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| String: | ‘b6’ | ‘G 8’ | ‘b6: g 8’ | ‘H4:A2’ |
| Co-ordinates: | 6 2 | 8 7 | (6 2)(8 7) | (4 8)(2 1) |

 See the description of the function *range* for ways to construct range co-ordinates dynamically.

1. A worksheet can be specified by name or number. Sheets are numbered in the left-to-right order in which they are displayed so moving a sheet will change sheet numbers. Beware that a sheet named ‘Sheet*n*’ (the *n*th sheet in creation order) is **not** necessarily sheet number *n*.
2. An **open** workbook can be specified by its short name, which excludes the path, or by number. The name must be any name in *(books ‘’)[;1]* and a number *n* denotes *(books ‘’)[n;1]*. You can usually omit the suffix but it is important to specify it where there are two or more workbooks whose names differ only by their suffixes. In that case, Excel will choose a workbook for you (experience indicates that it picks the first file with a matching name in the workbooks collection, which is in order of opening) if you omit the suffix.
The result of *(books ‘’)* is always in the order of opening, so when a workbook is closed, the numbers associated with workbooks that were opened subsequently will change. It is therefore not safe to refer to a workbook by number throughout an application, but a number is a useful shorthand for a name during interactive work.
3. An **unopened** workbook must be specified by its full name, which includes the path. Again, the suffix is optional but, as in paragraph (3), specifying it is recommended in case there are two files which differ only by their suffixes. In that case, Excel will choose a file for you if you don’t specify a suffix. Experience indicates that Excel 12.0 will always pick a .xlsx file over any other extension.
4. All character string arguments other than workbook names, sheet names, cell ranges and font names must be lowercase.

### R 🡨 active ’’

R is a 3-element list containing the short name of the active workbook (which omits the path), the full name (which includes the path), and the name of the active sheet. The right argument is ignored.

### addsheet ω

Add sheets to the active workbook. Sheets can be added at the end (the default) or before or after a specified sheet. ω has from 1 to 3 elements:

ω[1] is the number of sheets to add

ω[2] is ‘b’ (Before), ‘a’ (After), or ‘e’ (End, default)

ω[3] (needed only if ω[2]≠‘e’) is the sheet name or number to which ω[2] applies.

Examples:

|  |  |
| --- | --- |
| addsheet 2 | Adds 2 sheets at the end |
| addsheet 1 ‘e’ | Add 1 sheet at the end |
| addsheet 3 ‘b’ 1 | Add 3 sheets before sheet 1 |
| addsheet 1 ‘a’ ‘blotter’ | Add 1 sheet after sheet ‘Blotter’ |

### α bcolour ω (alt: bcolor)

Apply background colour α to the cell range ω in the active sheet. α is an integer in the range 0 to 56. Here are a few common values:

1. default
2. black
3. default colour but removes cell borders (as do all non-zero background settings)
4. red
5. medium green
6. blue
7. yellow
8. mauve

Examples:

3 bcolour 3 4 Set background to red in cell (3 4), ie, D3

0 bcolour (3 4)(5 8) Set background to default in the range (3 4) to (5 8), ie, D3:H5.

*bcolour* sets the Interior.ColorIndex property of a cell. This is is compatible with Excel versions prior to 12, and is supported in version 12 but is not used in the Excel 12 gui. Instead, the gui uses the Interior.Color property (an integer in the range 0 to 16777215) for ‘Standard Colors’ and ‘More Colors’ in the drop-down menu but uses Interior.ThemeColor in conjunction with Interior.TintAndShade for the Theme colours. The following examples respectively set a standard color and a theme colour, both shades of green:

 (xl.Range ’B5’).Interior.Color 🡨 5296274

 (xl.Range ’B5’).Interior.(ThemeColor TintAndShade) 🡨 xl.ThemeColorAccent3 0.6

### {α} bold ω

Set or remove bold attribute for the cell range ω in the active sheet.

α: 1= set bold (default), 0= cancel bold. Example:

bold (2 1)(12 1) ⍝apply bold

### R 🡨 books ’’

R is a 2-column matrix of the names of open workbooks; the row order corresponds to the order in which the books were opened. Column 1 contains the short name, which excludes the path, and column 2 the full name. Example:

books''

scatter.xlsm c:\apl\scatter.xlsm

Warnings:

If *wb* is not defined (see *startexcel*), Value Error occurs and so R is a character string.

Be careful when identifying a workbook by a number – a number is relative to workbooks that are currently open and can therefore identify different workbooks at different times.

### {α} border ω

Set or remove the border on a specific edge for the cell range ω in the active sheet.

α has one or two values:

α[1]: ‘t’, ‘b’, ‘l’, ‘r’ for top, bottom, left or right border

α[2]: any integer from 1 (default) to 13 sets the border; 0=removes the border

This function does not set inside borders. This means that the instruction

 ‘t’ border (19 4)(22 6)

sets only a top border for the cells (19 4) to (19 6), and that

 ‘l’ border (19 4)(22 6)

sets only a left border for the cells (19 4) to (22 4).

You could set all outside borders with

 ‘tblr’ border ¨ ⊂ (19 4)(22 6)

to insert a frame around the whole range but the function *frame* does this more simply.

The line style specified by α[2] is as follows (in Office 2007):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | Solid  | 6 | Bold italic dot dash | 11 | Dot dot dash (=5) |
| 2 | Dotted  | 7 | Solid (=1) | 12 | Double line (=9) |
| 3 | Faint dotted | 8 | Dotted (=2) | 13 | Bold italic dot dash (=6) |
| 4 | Dot dash | 9 | Double line |  |  |
| 5 | Dot dot dash | 10 | Dot dash (=4) |  |  |

A thick single line is not supported directly but you can obtain that by executing *border* with left argument 9 (double line) and then 1 (thin line) in turn.

### {α} border\_all ω

Set or remove top, bottom, left and right borders for each cell (ie, for inside and outside borders) in the cell range ω in the active sheet. α is any integer from 1 (default) to 13 to set a border or 0 to remove it. Example:

9 border\_all (19 4)(22 6)

The line style α has the same meanings as for *border*.

### R 🡨 cellrange ω

R is an array of co-ordinates covering each cell in the cell range ω. Example:

R 🡨 cellrange (2 3)(4 5)

ρR

3 3

R

2 3 2 4 2 5

3 3 3 4 3 5

4 3 4 4 4 5

### centre ω

Centre cell contents in each cell in range ω in the active sheet. ω is one or two pairs of co-ordinates. Example:

centre (1 1)(1 12)

If you apply *indent* then *centre* to the same cell(s), *centre* cancels the effect of *indent* (this is an Excel feature).

### R 🡨charts ω

Return a two-column matrix of chart names and titles in the sheet specified by ω. If ω is empty it implies the active sheet, otherwise ω (case-insensitive) specifies any sheet in the active workbook and therefore must be a name returned by *sheets*. Example:

charts''

Chart 5 line plot

Chart 6 scatter plot

### R 🡨 check ω

Return a boolean array corresponding to the cell range ω in the active sheet. Any 1’s in the array mean that the corresponding cells have an error such as #VALUE! or #DIV/0!. Example:

check ‘b4:f6’

0 0 0 0 0

0 0 1 0 0

0 0 0 0 0

This result implies an error in cell D5.

*check* is useful where *read* results in DOMAIN ERROR. The time fo r executing *check* is proportional to the number of cells in the range so *check* is quite slow for large ranges.

### R 🡨clean ω

Clean one data element before writing it to a sheet. Specifically:

Replace ι0 by []Null

Delete trailing blanks from a character string.

*clean* is invoked by *write* and *writef*.

### clear ω

Clear values and formulas (but not formats) in the cell range ω in the active sheet. However, ω may be a nested vector whose second elemt is ‘end’, in which case the effect is to clear from cell ω[1] to the end of the sheet. Examples:

 clear 20 30 clear the cell (20, 30) only

 clear (20 2)(25 8) clear the specified range

 clear (20 2) ‘end’ clear cell (20,2) and all cells below it and right of it

### {α} close ω

Close a workbook; if ω is an empty vector, it implies the active workbook, otherwise ω is the name (excluding path) of any open workbook and therefore must be one of the names (case-insensitive) returned by *books*; α has one of these values:

‘p’: (default) if there are any unsaved changes, Excel will prompt you; you may need to click the Excel taskbar icon or the session itself in order to make the prompt visible. Beware that the flashing that signals the prompt can be very easy to miss.

‘s’: if there are any unsaved changes, save the workbook first.

‘d’: discard any unsaved changes.

Warning: if α=’s’ and the workbook is previously unsaved (eg, Book1), Excel has its own logic about where it will put the saved workbook. Likely locations are My Documents or a folder that contains a recently-opened workbook. If you can’t find it, look at the definitions for Autorecover and Default file locations. You can find this information in version 12 by clicking the Office button, then Excel Options, then Save.

Examples:

 ‘s’ close ‘’ save and close the active workbook

 ‘d’ close ‘projects.xlsb’ close the specified workbook without saving changes

You can close all open workbooks (relying on Excel to prompt you if necessary) with the expression:

 close ¨ (books ‘’)[;1]

or

{close ω} ¨ wb.Countρ1

If you omit the suffix from ω and you have multiple books open with the same name but different suffixes, experience indicates that Excel closes the first file with a matching name in the result of *books*.

### R 🡨 copysheet ω

Copy or move one or more sheets, either within the active workbook or from the active workbook to another workbook. ω has from 1 to 5 elements:

[1] the source sheet names or numbers: an empty vector implies the active sheet; if more than one sheet is specified the list of names and/or numbers must be parenthesized

[2] (optional; default = empty vector) the target workbook; an empty vector implies the active workbook

[3] (optional; default = ‘c’) the action: ‘c’ = copy or ‘m’ = move

[4] (optional; default = ‘e’) the relative placement in the target workbook: ‘b’ = before, ‘a’ = after, ‘e’ = end

[5] (ignored if ω[4] = ‘e’) the sheet name or number that ω[4] applies to if ‘b’ or ‘a’

Examples:

1. copy the active sheet to the end of the active workbook:

copysheet ‘’

1. move the active sheet to the end of the active workbook:

copysheet ‘’ ‘’ ‘m’

1. move sheet 2 to the end of another workbook (accounts.xlsx):

copysheet 2 ‘accounts.xlsx’ ‘m’

1. copy two sheets to another book, inserting them after sheet ‘Mary’:

copysheet (‘apple’ ‘pie’) ‘recipes.xls’ ‘c’ ‘a’ ‘Mary’

If you specify more than one sheet, the order of the sheets is maintained regardless of the specification order in ω[1]. After either copy or move, the target workbook (ω[2]) becomes the active workbook. The result R is either the help string or an empty matrix.

### R 🡨 {α} dateToDaynum ω

Convert ω, an array of numeric dates in yyyymmdd format, to Excel day numbers. Other date formats are not anticipated because most date-string formats can be written directly to Excel. R has the same rank as ω. Example:

 dateToDaynum 20100615 20100623

40344 40352

If you write day numbers to Excel you will need to set the date format (see *format*), before or after you write the date values (obtained from dateToDaynum), if the date format is not already set.

### R 🡨 {α} daynumToDate ω

Convert day numbers (as returned by *read* for cells containing dates) in the array ω to dates in one of various conventional formats. Values in ω thatare not positive integers are passed through unchanged. R has the same rank as ω. α is an integer that determines the format of R as follows:

0: array of numeric vectors; each element contains (yyyy mm dd day-of-week); default

1: numeric array; each element is constructed as yyyymmdd

2: array of strings; each element is formatted as dd/mm/yyyy

3: array of strings; each element is formatted as mm/dd/yyyy

Examples:

daynumToDate 40344 40352

2010 6 15 1 2010 6 23 2

1 daynumToDate 40344 40352

20100615 20100623

 daynumToDate 40344 40352 25.6 ‘Total’ []null

2010 6 15 1 2010 6 23 2 25.6 ‘Total’ []NULL

You can easily extend this function if you want other format choices.

Excel stores dates as the number of days since 31/12/1899 (though there is a bug due to treating 1900 as a leap year). When you obtain the Value2 property of a cell (as in *read*), the number returned is the day number. This is not generally useful, except for computing differences between two dates, so *daynumberToDate* is provided in order to convert day numbers to conventional dates. If, however, you read the Text property of date cells, as with *readt*, you could perhaps dispense with *daynumberToDate*.

### deletesheet ω

Delete sheet(s) ω from the active workbook. ω is a list of sheet names and/or numbers; numbers must lie in the range ιxl.Sheets.Count and names must occur in the result of *sheets*. Domain Error occurs if a sheet name or number is invalid or if you try to delete all sheets. Example:

deletesheet 3 ‘blotter’

### R 🡨 dim ‘’

Return the dimensions of the sheet. R is a two-element vector comprising the number of rows and columns, which depends on the version of Excel. The right argument is ignored. Example:

dim ‘’

1048576 16384

### endexcel ‘’

Erase Excel objects from two namespaces: the one, such as #.XL, in which it (and, implicitly, all other functions in this package) resides and the calling namespace. *endexcel* does **not** terminate the Excel session itself. The goal is to clean up the workspace before you save it so that when you next load it you avoid a dialog box with the message ‘The following objects could not be rebuilt: ’. However, user-defined Excel objects may exist in other namespaces so you may want to extend the scope of *endexcel* yourself. *endexcel* assumes that any object owned by #.XL.xl (the ‘#.XL’ part of this string is determined dynamically) is an Excel object.

### α font ω

Set the font for the cell range ω in the active sheet. α is the name (case insensitive) of any font that appears in the Font drop-down box. Example:

‘apl385 unicode’ font 3 6

### α fontcolour ω (alt: fontcolor)

Set the font colour for the cell range ω in the active sheet. α is the value of the Font.Color attribute. Example:

¯16776961 fontcolour (1 1)(10 20) use red font

In Excel 12.0 (Office 2007), the values corresponding to the Standard Colours shown in the Font Colour drop-down box are, from left to right:

First one -16777024

Red -16776961

Ochre -16727809

Yellow -16711681

Pale Green -11480942

Dark Green -11489280

Light Blue -1003520

Dark Blue -4165632

Black -10477568 (but 1 and -1 work too)

Purple -6279056

Excel 12.0 allows colour numbers in the range from -2\*31 to (2\*31)-2. Some colours (eg, in the range -2 to -10) are so faint as to be invisible.

### α format ω

Set the NumberFormat property for the cell range ω in the active sheet. α is any legitimate format string for dates or numbers.

Here are some sample formats for α with their []fmt equivalents:

‘#,##0.00’ like ‘CF*n*.2’

‘0.0%’ like ‘F*n*.1,<%>’

‘0.00;[Red]0.00’ like ‘F*n*.2’ but negative values appear in red (without a minus sign)

‘$#,##0.00;[Red]$#,##0.00’ as previous example but with ‘$’ symbol

‘0.00\_ ;[Red]-0.00 ‘ like ‘M<->Fn.2’ but negative values appear in red (with a minus sign)

‘$#,##0.00;[Red]-$#,##0.00’ as previous example but with ‘$’ symbol

 ‘0’ like ‘I*n*’

You can also specify a decorator such as string of digits and/or the character ‘X’ in place of or as well as a $ sign in the foregoing examples (eg, ‘X-999-X#0.00’) but beware that many other characters cause an error.

The next two formats are affected by regional date and time settings specified for the operating system:

‘m/d/yyyy’ display date as ‘mm/dd/yyyy’

‘[$-F800]dddd, mmmm dd, yyyy’ display date as in ‘November 6, 2012’

These formats are independent of operating system settings:

‘dd/mm/yyyy;@’ display date as ‘dd/mm/yyyy’

‘d/m/yy;@’ display date as ‘dd/mm/yy’ but omit leading zeros

‘m/dd/yy;@’ display date as ‘mm/dd/yyyy’ but omit leading zeros

### {α} frame ω

Set or remove all outside borders around cell range ω in the active sheet. α is any integer from 1 (default) to 13 to set the border or 0 to remove it. For example:

6 frame (3 4)(8 12)

The line style α has the same meanings as for *border*.

### freeze ω

Freeze panes. ω is the number of rows and columns that you want to freeze; one or two zeros cancels freezing. For example:

freeze 1 2 freeze row 1 and columns A:B

freeze 2 0 freeze rows 1 and 2

freeze 0 remove freezing, if any

### help

Display one line of information for each function. For every function except *help* and *lc*, the same information can be obtained by executing the function with ‘?’ as the right argument.

### α indent ω

Set the left indent for the cell range ω in the active sheet. α is a small integer such as 0 (no ident), 1, 2 etc. Example:

indent (3 1)(8 1)

*indent* actually sets both left and right margins for the specified cell(s), so applying *ljust* or *rjust* after *indent* aligns the cell contents with the indented left or right margin. *indent* is implicitly cancelled by *centre*.

### R 🡨 α lastcell ω

ω specifies a single cell; α specifies the direction, either ‘l’, ‘r’, ‘u’, or ‘d’ (left, right, up, or down); R contains the co-ordinates of the last cell in a contiguous set of defined cells starting at cell ω, or, if cell ω is empty, starting at the first defined cell after ω, in direction α. Whether or not cell ω is defined, if all cells in direction α are empty, then R 🡨 ω. *lastcell* is used in *range*.

In Excel terms, if cell ω is defined and the adjacent cell in direction α is also defined, the last cell in the range in direction α is determined by placing the cursor at cell ω and pressing Ctrl simultaneously with the appropriate arrow key (eg, Ctrl+🡪). If cell ω is empty, Ctrl+🡪, eg, locates the first defined cell (if not the boundary of the sheet) and pressing Ctrl+🡪 again locates the end of the range (or the boundary of the sheet). Examples:

R 🡨 ‘d’ lastcell 2 1 last cell below (2 1)

R 🡨 ‘r’ lastcell ‘D3’ last cell to the right of ‘D3’

### ljust ω

Left justify text in each cell in the cell range ω in the active sheet. Example:

ljust (3 1)(3 20)

The effect of *ljust* for a given cell reflects the prior use of *indent* for the same cell.

### R 🡨 lc ω

ω is a character array of any dimension. R has the same structure as ω but each uppercase character is replaced by its corresponding lowercase character. Example:

lc 2 5ρ’Mary David’

mary

david

### R 🡨 {α} open ω

Open a workbook and return R as a workbook object. ω is

 An empty vector, implying a new workbook, or

 The full name (including the path) of a saved workbook; the suffix is recommended but optional.

If the second option applies and α is 0 or omitted, the workbook is opened normally. If α is 1 however, the workbook is opened under another name, given by R.Name. Changes to this workbook do not affect the original from which it was copied. This happens automatically (without specifying α) whenever you open a template workbook (extension .xlt or .xltm).

Warning: Domain Error occurs if ω cannot be opened; R is then returned as a character string.

If you omit the file suffix from ω and there is only one file that matches ω, there is no problem. If there are two or more files with the same name but different extensions, Excel picks a file according to its order of preferences; for example, Excel version 12 will open a .xlsx file in preference to one with any other suffix. You can open multiple files with different suffixes as long as you specify the suffix for the second and subsequent files.

Examples:

wbk 🡨 open ‘c:\docs\test.xlsm’

wbk.Name

test.xlsm

wbk 🡨 1 open ‘c:\docs\test.xlsm’

wbk.Name

test1

Illustrating the same file name with different suffixes:

 open ‘c:\mi\tb1’

 open ‘c:\mi\tb1.csv’

 open ‘c:\mi\tb1.xls’

 books ‘’

tb1.xlsx c:\mi\tb1.xlsx

tb1.csv c:\mi\tb1.csv

tb1.csv c:\mi\tb1.xls

Excel will prompt you if you try to open a workbook that is already open and that has unsaved changes. If you then click Cancel, a Domain Error occurs.

See ‘Excel Properties and Methods’ regarding the workbook object.

### R 🡨 {α} plot ω

Make a simple chart in the active worksheet and return R as a chart object, if successful, or else as a character string if there is an error in the specifications. ω is either a numeric array of the plot data **or** a list of alternating property names and property values. α specifies the plot type; if omitted, the chart is a line plot.

The purpose of *plot* is to demonstrate that one can usefully build simple, experimental Excel plots in Apl. There are many plotting options not covered by *plot* but conceptually, at least, *plot* is easily extensible. In an application, however, rather than create a chart from scratch in Apl, it would be preferable to save a chart definition in a workbook and just update properties, such as labels and data, from Apl. Excel supports 14 generic chart types and 73 individual chart types – these are listed in Appendix B. Support for charts with two y axes will be added later. **Warning**: Of all the functions in this package, *plot* has been the one most vulnerable to changes in Excel.

α is an integer or a string. Allowed plot types are listed in xl.XlChartTypes (see Appendix B). α must be one of the constants shown (eg, xl.xlLine, or its value, 4) or else one of the strings listed below, provided for convenience, which map to constants in xl.XLChartTypes:

‘l’ line chart (xl.xlLine)

‘lm’ line chart with markers (xl.xlLineMarkers)

‘sc’ stacked columns (xl.xlColumnStacked)

‘cc’ clustered columns (xl.xlColumnClustered)

‘pie’ pie chart (xl.xlPie)

The following table lists the property names allowed in ω together with the definitions of their associated data:

'data' A vector or matrix of plot data; each element or row respectively represents another observation and hence another value on the x-axis; a null value may be represented by []null or an empty vector.

‘null’ Null handling: 1= omit, 2= use zero, 3=interpolate; (these numbers are the values of the Excel constants xl.xlNotPlotted, xl.xlZero and xl.xlInterpolated).

'datarowscols' The number of rows and columns of data to plot; needed only if ‘data’ is omitted.

'dataposn' The row and column of the cell address which locates the plot data in the sheet; default is (1 1).

'title' Plot title (string).

'xtitle' X-axis title (string).

'ytitle' Y-axis title (string).

‘yrange’ Up to three values controlling labelling and tick marks for the y-axis: the maximum value, the minimum value and the interval size. For plot types whose names end in ‘100’, such as xlAreaStacked100, the values should be between 0 and 1 since they will be multiplied by 100 when applied. Excel’s defaults apply to omitted values and to values specified as an empty vector.

'legend' Legend: single string or nested set of strings (one for each column of plot data); superfluous legends are ignored.

‘legendposn’ Location of the legend: either ‘t’, ‘b’, ‘l’, ‘r’ or ‘c’, meaning top, bottom, left, right, or top-right corner.

'labels' X-axis labels (character matrix, one row per row of data); for a radar plot, labels apply to the spokes.

'labelposn' The cell address which locates the labels in the sheet. **Warning**: if you specify labelposn, the column of cells starting at labelposn must be defined or else you will get a very strange chart.

 ‘plotposn’ The x and y co-ordinates of the top-left corner of the chart frame. The co-ordinate unit of measurement is described later in this section. Either co-ordinate may be an empty vector, in which case the default placement is used for the left or top border respectively. The default placement of a chart is in the centre of the Excel window.

‘scaling’ Up to four values specifying chart scaling factors and the scaling origins. The first two values are the height and width scaling factors: the size of a standard chart is multiplied by these factors. The last two values indicate how to apply the scaling – whether to fix the top or left chart border (value 0), the vertical or horizontal chart centre (value 1), or the right or bottom border (value 2). The defaults are 1 1 0 0 – ie, no scaling and the top left corner as the scaling origin.

Restrictions and Relationships

If ‘data’ is specified, ‘datarowscols’ is derived from the data.

If ‘data’ is omitted, the plot data must be located in the active sheet at ‘dataposn’ and you must also specify ‘datarowscols’.

If you specify ‘labels’, you must also specify ‘labelposn’ (there is no default for this).

If you specify ‘labelposn’ but omit ‘labels’, the labels must be located in the active sheet at ‘labelposn’.

Notes on specific plot types:

**Bar:** The x and y axes are interchanged. Parameters xtitle and labels therefore apply to the vertical axis and ytitle applies to the horizontal axis.

**Bubble**: (1) Labels are not implemented. (2) Excel interprets the columns of Data as follows:

|  |  |
| --- | --- |
| # Cols | Interpretation |
| 1 | Column 1 is the y-value; all bubbles are the same size |
| 2 | Column 1 is the y-value and column 2 is the bubble diameter; the x-axis is labelled from 1 to the number of datapoints |
| 3 | Column 1 is the x-value, column 2 is the y-value (as in a scatter plot) and column 3 is the bubble diameter  |
| 4 | 2 series: Columns 1 & 2 are the y-value and diameter for series 1, columns 3 & 4 are the y-value and diameter for series 2; the x-axis is labelled from 1 to the number of datapoints |
| 5 | 2 series: Column 1 is the common x-value; columns 2 & 3 are the y-value and diameter for series 1; columns 4 & 5 are the y-value and diameter for series 2 |
| 2n (n≥2) | *n* series: Columns 1 & 2, 3 & 4, etc. contain respectively the y-value and diameter for each series 1 though *n*; the x-axis is labelled from 1 to the number of datapoints  |
| 2n+1 (n≥2) | *n* series: Column 1 is the common x-value; columns 2 & 3, 4 & 5, etc. contain respectively the y-value and diameter for each series 1 though *n* |

The diameter values are relative so if you scale the diameter values by a constant, the diameters in the chart do not change.

**Doughnut**: (1) Legend is not implemented; instead, use parameter ‘labels’ and/or ‘labelposn’. (2) If you have multiple columns of data, they are plotted concentrically, from innermost (first series) to outermost (the last). If you use plot type xlDoughnutExploded, only the outermost ring is exploded.

**Pie**: (1) Legend is not implemented; instead, use parameter ‘labels’ and/or ‘labelposn’. (2) xtitle, ytitle and yrange are not implemented. (3) Even if you specify more than one data column, only the first column is plotted.

**Doughnut, Pie, Radar**: Data are shown clockwise from 12 o’clock.

**Scatter**: (1) In general, Excel treats the first data column as x co-ordinates and all following columns as y co-ordinates. Thus if your data has *n* columns, a scatter plot will therefore contain *n-1* series. For that reason, you need only *n-1* legends. However, if your data has only one column, Excel treats the data as y co-ordinates and imputes x co-ordinates as 1, 2, 3, … up to the number of data points. (2) *plot* ignores labels for scatter plots. Although Excel lets you specify labels, the effect of doing so is to re-label the x-axis as 1, 2, 3, … up to the number of data points, which counters the intent of a scatter plot.

**Surface**: Two data columns are required; additional columns are ignored.

**Examples**

Excel lets you specify plot data as a series per column (the default) or a series per row. *plot*, however, assumes a series per column, thus you would define 8 values for each of 2 series as:

data 🡨 ?8 2 ρ32

The simplest command is:

c 🡨 plot data

which generates a line plot in the active sheet; *data* is a vector or matrix. The structure of the right argument is an exception to the normal rules and is implemented just for convenience; the equivalent full command is:

c 🡨 ‘l’ plot ‘data’ data

*c* is a chart object and Excel lets you change its properties. For example, you can change the chart to a column chart by re-defining the chart type:

 c.ChartType 🡨 xl.xlColumnClustered

but beware that this may not work for every chart type (because for some chart types you must specify the type before specifying the data).

You can add a title (but you have to set HasTitle before you can define the title itself):

 c.HasTitle 🡨 1

 c.ChartTitle.Text 🡨 ‘My First Chart’

Once a title is defined, you can also edit it in Excel by clicking in the title.

For later reference, the name of the plot is:

 c.Name

Sheet1 Chart 1

This result includes the name of the sheet. Alternatively:

 c.Parent.Name

Chart 1

The explanation for this is that *c.Parent* is a shape object (eg, xl.ActiveSheet.Shapes.Item 1), whereas c is the object xl.ActiveSheet.Shapes.(Item 1).Chart. For some reason the name of a shape object excludes the sheet but the name of a chart item includes it. The function *charts* returns the names of the shape objects that own charts, thus the result excludes the sheet name.

Note that *plot* first writes *data* to the active sheet. By default, *data* is placed with cell (1 1) as the top left corner, unless you specify an alternative position via *dataposn*. For example:

c 🡨 ‘l’ plot ‘data’ data ‘dataposn’ (12 3)

If your plot data is already in the sheet, then you can omit the ‘data’ paramater and specify instead the parameters *dataposn* (unless the default (1 1) is valid) and *datarowscols*. The latter is equivalent to ρ*data*. In theory, *datarowscols* could be determined dynamically but because of the possibity of adjacent non-plot data and/or nulls in the data, it’s preferable to state the data size explicitly. Thus you can build a plot using existing data by:

 c 🡨 ‘cb’ plot ‘dataposn’ (12 3) ‘datarowscols’ (8 2)

Note that these parameters are nested.

You can add titles and a legend as in:

c 🡨 ‘l’ plot ‘data’ data ‘xtitle’ ‘Month’ ‘ytitle’ ‘Number of Faults’ ‘legend’ (‘Type A’ ‘Type B’)
 ‘title’ ‘Faults by Type and Month’

Legend strings must be nested if there are more than one. See the note above about scatter plots.

To use x-axis labels in place of Excel’s default numbering, define an array with one row per observation and specify the ‘labels’ and ‘labelsposn’ parameters (‘labelsposn’ is analagous to ‘dataposn’):

 labels 🡨 8 3 ρ ‘JanFebMarAprMayJunJulAug’

 c 🡨 ‘l’ plot ‘dataposn’ (12 3) ‘datarowscols’ (8 2) ‘labels’ labels ‘labelposn’ (12 6)

You can use the same labels in subsequent plots by specifying ‘labelposn’ but omitting ‘labels’. You may also use a numeric vector for the labels instead of a character matrix.

You can override the default y-axis labels and tick marks by specifying ‘yrange’. Here’s an example of specifying the upper tick mark at 30, the lower one at 10 and the interval at 5 (meaning a tick mark and every 5 units):

 c 🡨 ‘l’ plot ‘dataposn’ (12 3) ‘yrange’ (30 10 5)

You will see labels at 10, 15, 20, 25 and 30. If, however, you specify the interval as 6, you will see tick marks at 10, 16, 22 & 28 – the upper value of 30 is ignored. If you have data values that are outside the range implied by the first two values, the chart will be truncated. If you specify one or two values only, Excel’s defaults apply to the omitted attributes. You can also specify any of the three values as an empty vector to mean Excel’s default value. For percentage plots, such as xlAreaStacked100, use yrange values between 0 and 1.

By default, a chart is centred in the Excel window, but it will be truncated at the right and bottom if the window is too small. You can place a chart elsewhere via the parameter plotposn, which locates the top left corner of the chart. Empirically, and assuming that all rows have the same height and that all columns have the same width, the plotposn units are column width and row height, as returned by (on my pc):

 xl.Columns.(Item 1).Width

48

 xl.Rows.(Item 1).Height

15

**Warning**: these values may differ from what Excel reports when you click on the top or left border and view the column width or row height; for example, Excel shows column width as 8.43, in contrast to 48 reported above.

Therefore, if you execute

 (w h) 🡨 xl.(Columns.(Item 1).Width Rows.(Item 1).Height)

you can then place the top left corner of a chart in cell (r,c) by specifying plotposn as ((w,h)×(c,r)-1) – note that you specify the x-dimension first. For example, where (r,c) is (8, 12):

 c1 🡨 ‘l’ plot ‘data’ data ‘plotposn’ ((w,h) × (12 8 - 1))

You should therefore use plotposn when you want to create several charts on the same sheet. Here’s an example of stacking two charts vertically:

 c1 🡨 ‘l’ plot ‘data’ data1 ‘dataposn’ (1 1) ‘plotposn’ ((w,h) × (6 8 - 1))

 c2 🡨 ‘l’ plot ‘data’ data2 ‘dataposn’ (21 1) ‘plotposn’ ((w,h) × (6 30 - 1))

You can also move a chart after creating it, as in:

 c2.ChartArea.(Top Left) 🡨 200 400

In both Excel and Apl, c.ChartArea.(Top Left) always returns 4 4, so it seems that you can’t read the position.

You can change the size of the chart by specifying scaling factors. In the next example, the chart height is increased by 50% and the width by 30%:

c 🡨 ‘l’ plot ‘data’ data ‘scaling’ (1.5 1.3)

By default, the top and left borders do not change when scaling is applied. However, you can specify the scaling origin as top border, centre or bottom border for the height and left border, centre or right border for the width. For each dimension, 0=top or left (the default), 1=centre, 2=bottom or right. Thus you can fix the right and bottom borders with the command:

c 🡨 ‘l’ plot ‘data’ data ‘scaling’ (1.5 1.3 2 2)

If you want to scale one dimension only, specify factor 1 for the other dimension. The default scaling values are 1 1 0 0, which apply respectively to height, width, height, width. If you supply less than 4 values, the defaults apply to the omitted attributes.

Excel supports three options for dealing with nulls in the data:

1. Omit the point (there will be a gap) – this is the default
2. Substitute zero
3. Interpolate between the preceding and following points provided that there are non-null points on both sides of the null point.

In Apl, a null value can be represented as []null or ⊂''; ⊂ι0 is allowed but only because *write* converts it. To specify the third option, for example, include the parameter ‘null’ and the option number, as in:

c 🡨 ‘l’ plot ‘dataposn’ (12 3) ‘datarowscols’ (8 2) ‘labels’ labels ‘labelposn’ (12 6) ‘null’ 3

After creating a chart, you can change the null-handling option as in:

 c.DisplayBlanksAs 🡨 2

You can activate a chart via, eg,

xl.ActiveSheet.Shapes.(Item ‘chart 5’).Select ι0.

Now you can access the chart’s properties and methods via xl.ActiveChart; for example, xl.ActiveChart.Name returns the name of the active chart.

Although xl.ActiveChart nominally supports the method Delete, it fails, so instead delete the active chart with the command:

 xl.ActiveChart.Parent.Delete

or c.Parent.Delete

where c is returned by *plot*.

You can also delete any chart with the command:

 xl.ActiveSheet.Shapes.(Item *n*).Delete

where *n* is a number (meaning the *n*th item returned by *charts*) or the chart name (without the sheet name). You can delete all charts with the command

{xl.ActiveSheet.Shapes.(Item 1).Delete}¨ι⊃ρcharts’’

since the charts are implicitly renumbered after you delete the first one.

### quitexcel ‘’

quitexcel terminates the Excel session and erases all Excel objects in #.XL. *quitexcel* invokes *endexcel*.

Warnings:

* If you invoke *endexcel* then *quitexcel*, *quitexcel* cannot now terminate Excel.
* If there is an unsaved workbook when you execute *quitexcel*, Excel will display a dialog box (“Do you want to save …”). Apl does not know about this so if you subsequently execute *startexcel* while the dialog box is open, *startexcel* will fail.

### R 🡨 range ω

R is two co-ordinate pairs of diagonally opposite corners of the cell range ω, which has a special, unique definition. The intent of *range* is to let Excel determine a range dynamically by scanning the sheet from the given start point (which may be an empty cell) until it reaches a defined cell followed by an empty cell. You can specify the number or rows and/or columns in the range, or you can let Excel determine both. You can also specify the directions to scan instead of the default down and right. ω is a vector of from 1 to 5 elements but the first element must be explicitly enclosed if it is the only element:

[1] single cell specifying any corner of the range; ‘’ implies the active cell

[2] # of rows (positive or negative) in the range **or** empty vector

[3] # of columns (positive or negative) in range **or** empty vector

[4] direction if [3] is empty: ‘d’=down (default) or ‘u’=up; ‘’ implies ‘d’

[5] direction if [4] is empty: ‘r’=right (default) or ‘l’=left; ‘’ implies ‘r’

**Warning**: if ω[1] contains numerical co-ordinates, they must be parenthesized if other arguments follow or explixitly enclosed if no arguments follow.

Where ω[3] is empty, *lastcell* determines the extent of the range in the row dimension according to ω[4], and where ω[4] is empty, *lastcell* determines the extent of the range in the column dimension according to ω[5]. If ω[2] or ω[3] is an integer, the sign implies the direction: positive means down or right, negative means up or left. Which set of corners are present in R (top-left and bottom-right or bottom-left and top-right) and the order of the co-ordinates depends on how ω is specified.

Examples:

The first grid below represents a section of a spreadsheet. The second shows the results of applying *range* to the first grid; the active cell is assumed to be B3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|   | **A** | **B** | **C** | **D** | **E** |
| **1** | 3 | 31 | 32 | 12 |  |
| **2** | 6 | 27 | 24 | 15 |  |
| **3** | 22 | 14 | 23 |   | 36 |
| **4** | 34 | 13 | 15 | 2 |  |
| **5** | 23 |  | 1 | 7 |  |
| **6** |  | 15 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Range Selected | Result | Comments |
| range 'a1' 4 3 | A1:C4 | (1 1) (4 3) | Select 4 rows and 3 columns down and right from A1  |
| range (1 1) 4 3 | A1:C4 | (1 1) (4 3) | Identical to previous line |
| range ⊂ 1 1 | A1:D5 | (1 1) (5 4) | Starting at A1, select across row 1 to the first defined cell followed by an empty cell (D1) and down column A to the first defined cell followed by an empty cell (A5) |
| range (1 1) ⍬ ⍬ | A1:D5 | (1 1) (5 4) | Identical to previous line |
| range ⊂ '' | B3:C4 | (3 2) (4 3) | Starting at B3 (the active cell), select down and right to the first cell followed by a blank cell |
| range 'D5' ⍬ ⍬ 'u' 'l' | C4:D5 | (5 4) (4 3) | Starting at D5, select up column D and left across row 4 |
| range '' ⍬ ⍬ ‘u’ 'l' | A1:B3 | (3 2) (1 1) | Starting at B3 (the active cell), select up column B and left across row 3 to the edge of the grid |
| range 'b2' ⍬ 5 '' 'l' | A2:F4 | (2 2) (4 6) | Starting at B2, select down column B to the first blank cell and 5 columns right across row 2; direction ‘l’ is ignored because the number of columns is specified |
| range 'b2' ⍬ ¯5  | A2:B4 | (2 2) (4 1) | Starting at B2, select down column B to the first blank cell and 5 columns left across row 2 (to edge of grid) |
| range (4 5) ¯3 ¯2 | D2:E4 | (4 5) (2 4) | Starting at D4, select 3 rows up and 2 columns left |

*range* is particularly useful where you want to read or format an array of unknown size. For example:

frame range 'D3' '' 5

r 🡨read range ⊂ ''

### R 🡨 rc ω

R is one or two row-column co-ordinate pairs derived from the conventional (case-insensitive) address of the cell (range) string ω. Embedded spaces in ω, which would cause an error, are removed. A numerical argument is returned unchanged. Examples:

rc 'F8'

8 6

rc 'F 8'

8 6

rc 'a4:g6'

 4 1 6 7

### R 🡨 rd ω

R is a range definition string for the cell range ω, which is one or two row-column co-ordinate pairs. A string argument is returned unchanged. Examples:

rd 8 6

$F$8

rd (4 1)(6 7)

 $A$4:$G$6

### R 🡨 read ω

R is an array of data obtained from the cell range ω in the active sheet. R has rank 0 or 1 for a single cell (1 implies a string) and rank 2 for multiple cells. Example:

R 🡨 read 'a4:g6'

ρR

3 7

It does not matter how you specify the range, hence the following three commands return identical results:

read 'a4:g6'

read 'a6:g4'

read 'g6:a4'

Empty cells are returned as []NULL and dates (unless stored as strings) as day numbers (see *daynumToDate*).

**Warning**: A character string (beginning ‘Domain Error:’) is returned in the event of Domain Error, which occurs if you attempt to read a range containing an error such as #VALUE! or #DIV/0!. If you wish you can execute *check* to determine which cells are in error.

### R 🡨 readf ω

R is an array of formulas or formatted values obtained from range ω in the active sheet. For any cell in range ω, the corresponding element of R contains the formula, if one is defined, or else the formatted value, if a value is defined, or else an empty vector. Thus R contains strings only. R is rank 1 for a single cell or rank 2 for multiple cells. Example:

ρ[]🡨readf 'b15:c15'

 =SUM(B2:B12) =SUM(C2:C12)

 1 2

### R 🡨 readt ω

R is a matrix of formatted values (the Text property of each cell) obtained from range ω in the active sheet. All elements of R are strings. In contrast to *read* and *readf*, *readt* returns what’s displayed in the sheet rather than the underlying values or formulas. Thus where *read* returns the values 12345.6789 and 41002, *readt* might return ‘$12,345.68’ and ‘03/04/2012’ respectively (dates are stored as day-numbers). Like *readf*, *readt* returns strings only. R is rank 1 for a single cell or rank 2 for multiple cells. Example:

ρ[]🡨readt 'b15:c15'

 $1,474.00 $4,222.00

1 2

Note that the Text property is not defined for an Excel Range object; instead, you have to obtain the Text property for one cell at a time. Compare *readt* with *read* to see the different techniques.

### rename ω

Rename sheets in the active workbook; ω is a list of the form:

existing-sheet-*i* new-name-*i* [existing-sheet-*j* new-name-*j* … ]

Existing-sheet-*i* is either the name or number of an existing sheet (an empty vector denotes the active sheet) and new-name-*i* is its intended new name (note that a number is converted to a string). The order of the existing sheets is unimportant and names are case-insensitive. New names must be legitimate (beware that some special characters are prohibited) and *rename* removes leading and trailing blanks. *rename* has no impact the designation of the active sheet.

Example: Rename the active sheet to ‘Purchases’ and sheet 1 to ‘Sales’:

rename '' 'Purchases' 1 'Sales'

### rjust ω

Right justify text in each cell in range ω in the active sheet. Example:

rjust (3 4)(5 8)

The effect of *rjust* for a given cell reflects the prior use of *indent* for the same cell.

### {α} saveas ω

Save an open workbook under another name or file type. ω has two or three elements:

ω[1]= the name of an open workbook (excluding the path); ‘’ denotes the active workbook

ω[2]= the name of the file to write (usually including path and suffix)

ω[3]= the file format (optional).

In general you can specify the file suffix in ω[2] to imply the file format and omit ω[3]. Example:

 saveas '' 'c:\projects\businessplan.xlsx'

In such a case, the file suffix in the following table of common suffixes is mapped to the value in column 3 and this value is supplied as the format type in the Excel ‘saveas’ command.

|  |  |  |
| --- | --- | --- |
| Suffix | Constant | Value |
| xls | xl.xlWorkbookNormal | -4143 |
| xla | xl.xlAddIn | 18 |
| xlt | xl.xlTemplate | 17 |
| csv | xl.xlCSV | 6 |
| xml | xl.xlXMLSpreadsheet | 46 |
| html | xl.xlHtml | 44 |
| xlsb | xl.xlExcel12 | 50 |
| xlsx | xl.xlOpenXMLWorkbook | 51 |
| xlsm | xl.xlOpenXMLWorkbookMacroEnabled | 52 |
| xlam | xl.xlOpenXMLAddIn | 55 |
| xltx | xl.xlOpenXMLTemplate | 54 |
| xltm | xl.xlOpenXMLTemplateMacroEnabled | 53 |

Note that the 4-character suffixes apply to Excel version 12.0 and the 3-character ones to earlier versions.

If you omit the suffix from ω[2], as in:

 saveas '' 'c:\projects\businessplan'

then Excel determines the file type from the file ω[1], in this case the active workbook. Excel then appends the appropriate suffix to the file name when the file is saved. Thus if you don’t specify a suffix in ω[2] or a file format (ω[3]) then the saved file will have the same format as file ω[1].

If you omit the path from ω[2], Excel will use a default location, possibly c:\Documents and Settings.

In the rare case when you want to specify a file format other than those listed above, use one of the constants (or its value) listed in xl.XlFileFormat (of which the table above is a subset), shown in Appendix B; you can then omit the file suffix from ω[2].

Note the following:

* Even if you are using Excel 12.0, you can still save a workbook in the format of an earlier version.
* If *saveas* appears to hang, it’s probably because Excel has issued a prompt but the prompt window is obsucred. In that case, click the Excel session or the task bar icon for the relevant workbook in order to see the it. Some causes of an Excel prompt are:
	+ you specify both a file suffix and a file format and they are inconsistent
	+ some formatting isn’t supported by the file format you selected
	+ the file specified in ω[2] already exists.

The last two situations (and possibly others) can be anticipated by defining α. α is optional and is either ‘p’ (prompt; default) or ‘f’ (force). In the situation where you want to write the file regardless, ‘f’ forces the file write whereas ‘p’ causes Excel to prompt you. If you click ‘Cancel’ in a prompt window, you will then see a dialog box indicating that the saveas method failed.

Examples: (*books* is used to illustrate the effect of *saveas*)

1. Open a new workbook, write to it, and save it as a Windows csv file:

open ''

books ''

Book1 Book1

 (?12 12 ρ200) write 'a2'

 saveas '' 'c:\mi\test1.csv'

 books ''

test1.csv c:\mi\test1.csv

1. Creat another workbook and save it as a csv file for the Macintosh:

open ''

books ''

test1.csv c:\mi\test1.csv

Book2 Book2

 (?12 12 ρ200) write 'a2'

 saveas '' 'c:\mi\test2' xl.xlCSVMac ⍝suffix is omitted from ω[2]

 books ''

test1.csv c:\mi\test1.csv

test2.csv c:\mi\test2.csv

1. Re-save test1.csv as an xls file; force over-write if there is an existing file:

'f' saveas 'test1' 'c:\mi\test1.xls' ⍝suffix is omitted from ω[1]

 books ‘’

test1.xls c:\mi\test1.xls

test2.csv c:\mi\test2.csv

### R 🡨 setbook ω

There are two situations:

* If ω is the number or name (case-insensitive name returned by *books*) of an open workbook, *setbook* designates ω as the active workbook and returns R as the object for workbook ω.
* If ω is an empty vector, *setbook* just returns R as the active-workbook object.

Examples:

R 🡨 setbook ‘accounts.xlsb’

R 🡨 setbook ‘’

See ‘Excel Properties and Methods’ for a description some of the properties and methods of the workbook object.

### R 🡨 setsheet ω

There are two situations:

* If ω is the number or name (case-insensitive name returned by *sheets*) of a sheet in the active workbook, *setsheet* designates ω as the active worksheet and returns R as the object for sheet ω.
* If ω is an empty vector, *setsheet* just returns R as the active-sheet object.

Examples:

R 🡨 setsheet ‘clients’

R 🡨 setsheet ‘’

See ‘Excel Properties and Methods’ for a description some of the properties and methods of the workbook object.

### R 🡨 sheets ω

R is a list of the sheets in the workbook specified by ω. ω is the number or name (case-insensitive name returned by *books*) of an open workbook or else an empty vector, implying the active workbook. The sheets are listed in R in the same order as they appear in the Excel session. Examples:

R 🡨 sheets ‘accounts.xlsb’

R 🡨 sheets ‘’

### R 🡨 {α} startexcel ω

Where ω is an empty vector, *startexcel* uses OLE to attach to an existing Excel session, if possible, or else starts a new Excel session. Where ω is ‘new’, *startexcel* starts a new Excel session even if there are existing sessions. α is 1 (default) to make the session visible or 0 to hide it. R is 0, indicating success, or the ‘help’ string. *startexcel* creates two objects of type OLEClient: *xl*, the Excel application object, through which all Excel properties and methods are accessible, and *wb*, the workbooks collection object. See ‘Excel Properties and Methods’ re *xl* and *wb*.

If there is no current Excel session, then

startexcel ''

and startexcel 'new'

are equivalent; both start a new Excel session and define *xl* and *wb* for that session.

If there is one existing Excel session that is not connected to the current Apl session, then

 startexcel ''

connects to that session and defines *xl* and *wb* for that session. If there are multiple Excel sessions but none is connected to Apl, it is unpredictable which session Apl will connect to.

If there is already a connection to an Excel session, then:

*startexcel ''*  has no impact (except perhaps to change the Visible attribute)

*startexcel 'new'* starts a new Excel session, connects to it, and defines *xl* and *wb* for the new session. The previous session survives but can no longer be reached from Apl.

If *startexcel* fails, first determine whether Excel is busy or awaiting input, either in a cell or a modal dialog box. (This situation can interfere with any function and, if it applies, usually generates a dialog box containing the message ‘Error: RPC\_E\_CALL\_REJECTED’.) In other situations, first execute *endexcel ‘’* (to erase all Excel objects from your workspace) before trying again, then, if necessary, manually terminate Excel (check the Task Manager to make sure that there are no hidden instances of Excel). In extreme cases, re-boot.

If *startexcel* starts a new Excel session, as opposed to connecting to an existing session, Excel add-ins are not loaded. You need to start Excel independently in order to obtain add-ins.

### R 🡨 usedrange ω

R is a two-element nested vector for the sheet specified by ω. ω is the number or name (case-insensitive name returned by *sheets*) of a sheet in the active workbook or else an empty vector, implying the active sheet. The elements of R are:

R[1] the address of the UsedRange property

R[2] an array of the data in the cells coverd by R[1].

Exceptions:

* If there are no defined cells in the sheet, R[1] is an empty vector (even though the UsedRange property would return ‘$A$1’ in that case) and R[2] is 0 0ρ0.
* If R[1] covers more than one cell, R[2] has rank 2, but if R[1] covers one cell only, R[2] is a numeric scalar or a string vector.
* DOMAIN ERROR occurs if a cell contains an error such as #N/A or #DIV/0!.

Specifying a sheet other than the active sheet does not change the designation of the active sheet. Example:

R 🡨 usedrange 'clients'

 ρ¨R

1. 3 2

 R

|  |  |  |
| --- | --- | --- |
| '$A$1:$B$3'  | MaryTomPeter | X CorpY CorpZ corp |

### {α} wrap ω

Set or remove text wrapping (the WrapText attribute) for range ω in the active sheet. α is 1 (default) to set text wrapping or 0 to cancel it. Example:

wrap (1 1)(1 10) ⍝set wrapping for ‘A1:A10’

### α write ω

Write a single data item or data array α to the active worksheet; ω contains the co-ordinates of the top-left corner of the target range. Both ρρα and (|*depth* α) must be 2 or less and each element must be a numeric scalar or a string. In general, if a string begins with ‘=’, it i treated as a formula and evaluated, and, if it matches a date pattern, it is converted to a day number, but see *writef* for exceptions. If α is a string it is treated as a unit and written to a single cell; if α is a character matrix, each row is written to a single cell. A vector other than a string is written horizontally. Empty cells in α can be specified as ‘’, ι0, or []Null. (An attempt to write ι0 to a cell would cause Domain Error, but *write* invokes *clean* which replaces ι0 by ‘’ and also removes trailing blanks from strings.) When you read a cell written as ‘’, []Null is returned. Examples:

'mary' write 'e1' write a string to E1

('alpha' 'bravo' 'charlie') write 'e1' write 3 strings to F1:H1

(↑'alpha' 'bravo' 'charlie') write 2 6 write 3 strings to F2:F4

(2 3 ρι6) write 2 2 write a matrix to B2:D3

See section “Assigning values to a range or selection**”** under Excel Properties and Methods for other ways of assigning values.

### α writef ω

Write a single formula or date or array of formulas and/or dates α to the active worksheet; ω contains the co-ordinates of the top-left corner of the target range. α has the same constraints and structures as for *write*. Example:

'=sum(b1:b12)' '' '=sum(d1:d12)' writef 'b14'

In fact, both *write* and *writef* allow you to include data, date strings and formulas in α. However, where α has more than one element and contains strings only, formulas and date strings in α won’t be evaluated unless you use *writef*. In general, therefore, use *writef* for formulas and date strings. ‘Evaluating a date’ means that Excel coverts the date to a day number and formats it according to whatever format pattern is set for the cell. In some cases evaluating a date isn’t necessary but it has advantages – for example, you can’t change the format of a date that’s a string.

### R 🡨 α xread ω

R is an array of data obtained from the workbook, worksheet and range specified in ω. x*read* invokes *startexcel* to start the Excel interface if necessary, and opens or activates the workbook and activates the worksheet. The flags in α control whether *xread* starts a new Excel session and whether it closes the excel interface and the workbook.

ω is a nested array specifying:

ω [1]: workbook

ω [2]: worksheet

ω [3]: cell range.

α is an optional character string; allowed values (other than blank, which is ignored) are:

‘n’: start a new Excel session

‘c’: close the workbook

‘e’: invoke *endexcel*

‘q’: invoke *quitexcel*

See Appendix C for more information.

### α xwrite ω

Write a single data item or data array (ω[4]) to the workbook, worksheet and cell specified in ω[ι3]. xw*rite* invokes *startexcel* to start the Excel interface if necessary, and opens, creates or activates the workbook and activates the worksheet. The flags in α control whether or not *xwrite* closes the excel interface and saves and closes the workbook.

ω is a nested array specifying:

ω [1]: workbook

ω [2]: worksheet

ω [3]: cell; the top left corner of the target range

ω [3]: data; scalar, vector, or matrix; note that a string and each row of a character matrix are treated as a single entity.

α is an optional character string; allowed values (other than blank, which is ignored) are:

‘n’: start a new Excel session

‘c’: close the workbook

‘s’: save the workbook (without closing it)

‘e’: invoke *endexcel*

‘q’: invoke *quitexcel*

See Appendix C for more information.

# Excel Properties and Methods

You can explore Excel’s properties and methods via the PropList and MethodList properties of any Excel object in the Apl environment. The lists below show a few useful ones – knowing a few of them is helpful when interacting with Excel from Apl. Note that some properties are settable, others are read-only. Also note that some methods fail. If you can’t get a method to work, the best way to understand it is to search Excel’s help or simply try to use the method in a macro (you will at least see a tooltip indicating the arguments) but sometimes even that doesn’t help.

## xl – The Application Object

**Properties**

xl.DisplayAlerts usally 1; set to 0 to avoid prompts

xl.Selection.(Row Column) co-ordinates of the top left corner of the current selection

xl.ThousandsSeparator eg, ‘,’ or ‘.’

xl.UserName

xl.Version

xl.Visible set to 1 (visible) or 0 (hidden)

xl.Width session width in characters (based on the standard font)

Note that there are also many constants (eg, xl.xlRight, xl.xlNone, xl.xlContinuous) which are not exposed as properties. Some of these are used in the functions described in this document.

**Methods**

xl.Quit terminate the Excel session

xl.Run ‘*macro*’ run an Excel macro\*

xl.Selection.Address ‘’ the address of the current selection (null argument is required)

xl.trim ‘*string’* returns a two-element array of the trimmed string (or number) and the original

\*In Excel 12, you first need to click the Office button then select Excel Options, Trust Center and Macro Settings, then check the box ‘Trust access to the VBA developer model’.

## Excel WorkSheet Functions

Worksheet functions are implemented as methods of xl.WorkSheetFunction. You can apply worksheet functions directly to apl variables as well as to Excel ranges. For example:

 (3+ι5) write 1 1 ⍝range A1:E1

 xf 🡨 xl.WorksheetFunction

 xf.StDev 3+ι5

1.58113883

 xf.StDev xl.Range ‘A1:E1’

1.58113883

 xf.Match 7 (xl.Range ‘A1:E1’) 0

4

xf.IsNumber xl.Range ‘C1’

1

xf.IsNumber¨ 34 'apple' 55.1 'pear'

1 0 1 0

IsNumber reports Domain Error for arguments other than simple scalars and strings.

## Collection Objects

All collection objects have Count (the number of members) and Item (specify a member) properties. Thus:

wb.Count returns the number of open workbooks (wb is defined in *startexcel*)

wbk🡨wb.Item ‘book1’ defines wbk as the workbook object for the member Book1

wbk🡨wb[⊂‘book1’] is an alternative to using the Item property

wbk.Sheets.Count returns the number of sheets in Book1

xl.Sheets.Count returns the number of sheets in the active workbook

sh🡨xl.Sheets[1] defines sh as the sheet object for the sheet 1.

Note that *startexcel* sets []WX🡨1, which is essential for enabling both Item and indexing methods to access collection members. If []wx🡨3, use of Item is disabled.

If you use indexing, the index may be an enclosed name, a number, or a list of names and/or numbers. In some cases, however, use of Item is preferable; as an example see *setsheet*, which invokes the method Activate, which may apply only to a scalar object. Item allows a string, a scalar and a one-element vector as arguments but use of indexing in *setsheet* would require analysis of the argument in order to enclose a string but not a one-element vector.

For specifying multiple members with the Item property, use “each”, as in xl.Sheets.(Item¨ι3).Name.

Some other collections are Cells (includes all cells in a spreadsheet), Rows (the rows of a spreadsheet), Columns (the columns of a spreadsheet), Names (names of defined ranges and tables in a sheet or workbook) and Shapes (charts, comments and GUI objects in a sheet). For the active sheet, it sufficient to use xl.Cells, xl.Rows etc but the form xl.ActiveSheet.Cells and xl.ActiveSheet.Rows works too.

There is a Names collection for the workbook as well as for each sheet, so use xl.Sheets[i].Names or xl.ActiveSheet.Names for a sheet and xl.Names or xl.ActiveWorkbook.Names for the active workbook or wb[i].Names for any other workbook. *rc* does not support named ranges (it returns the co-ordinates of one corner only) so in order to obtain the address of the named range ‘sales’, for example, in the active sheet, you need to do:

addr🡨 xl.(Range ‘sales’).Address’’

The collection xl.Cells can be indexed but it is simpler and more intuitive to use the Item property. A collection, by definition, is a list, so xl.Cells represents a list of (×/dim ‘’) cells. Since the list is the order row 1, row 2, etc, xl.Cells[ι3] therefore refers to the range A1:C1 and, given 16,384 columns in Excel 12, xl.Cells[1+16384×0,ι3] refers to the range A1:A4. You could convert row-column co-ordinates to a single number via an expression like *col+(row-1)×16384* but it would be cumbersome and unintuitive. For comparison, here are two equivalent methods for reading three non-contiguous cells (assuming origin 1):

1. R 🡨 (xl.Cells.Item *¨*(3 1)(7 3)(8 6)).Value2
2. cellIndex🡨 {ω[;2]+(ω[;1]-1)×16384}

 R 🡨 xl.Cells[cellIndex↑(3 1)(7 3)(8 6)].Value2

Beware that both will fail if any of the specified cells is empty. In that case you could use the Text or Formula property instead of Value2.

Notes:

1. In Dyalog 10.1, indexing collections is not implemented.
2. In Dyalog 12.1 (and prior versions where []WX defaults to 3) []WX must be set to 1 globally. Dyalog incorrectly uses the global value of []WX instead of the local value set in *startexcel*.
3. In Dyalog 13, changing []WX has no impact on how you can interact with Excel – the effective value is always the local value set in *startexcel*.
4. The Shapes collection cannot be indexed in Dyalog versions prior to 13.
5. For all collections, Item is normally elided in VBA but is always required in Apl if you don’t use indexing.

## wbk – A Workbook Object (returned by *setbook* and *open*)

**Properties**

wbk.FileFormat returns a format number from xl.XlFileFormat (see Appendix B)

wbk.FullName

wbk.Name

wbk.ReadOnly 0= read & write, 1= read only

wbk.Saved 0=there are unsaved changes; 1=no outstanding changes

wbk.Sheets.Count the number of sheets in the workbook

**Methods**

wbk.Activate

wbk.Save

wbk.Close ‘’ note the required argument

wbk.SaveCopyAs file (argument is the file to be saved, including path and extension)

If you have closed the workbook to which wbk refers, trying to access a method or property (excluding PropList & MethodList) of wbk will result in a dialog box with the message ‘RPC\_E\_DISCONNECTED’.

## sh – A Sheet Object (returned by *setsheet*)

**Properties**

(sh.Columns.Item 1).Width width of column 1 (Item ‘A’ is allowed too) but the value differs from what you get in Excel (presumably different units are being used)

(sh.Rows.Item 3).Height height of row 3 (see parameter ‘plotposn’ in the section on *plot*)

sh.PageSetup.PrintArea print area address

sh.Name sheet name

**Methods**

sh.Activate (re-)activate the sheet

sh.Delete delete the sheet (but Excel will prompt you unless you set xl.DisplayAlerts🡨0)

sh.Copy copy the sheet; example: sh.Copy ⊂ (‘After’ (xl.Sheets.Item ‘test’))

sh.Move move the sheet; example: sh.Move ⊂ (‘Before’ (xl.Sheets.Item 1))

sh.Next activate the next sheet and return it as an object (sh.Next.Next is allowed)

sh.Paste the final action of a cut/copy and paste sequence

sh.Previous.Activate activate the sheet before sh.Name

sh.Next.Activate activate the sheet after sh.Name

sh.UsedRange.Address ‘’ return the (smallest) address range that covers all defined cells

## rs – A Range or Selection Object

You can define a range object in code by, for example,

 rs🡨 xl.Range ’A5:C8’

In fact, you can define several range objects. A selection object is a special type of range object; both have the same properties and methods. The difference is that there can be only one selection object, which you can define in code, as in

xl.(Range’A5:C8’).Select

or, in Excel, by dragging the mouse over some cells, rows, or columns in the sheet or by clicking the top left corner to select the entire sheet. Both the command xl.(Range’A5:C8’).Select and the mouse action define the object xl.Selection; note that the Select instruction returns the value 1.

In the following sample properties and methods, rs represents either a range object, as defined above, or xl.Selection.

**Properties**

rs.BorderAround🡨n n=1 sets a continuous line, n=2 and n=3 sets a dotted line, n=4 sets a dot-dash line, etc.

rs.WrapText🡨 Boolean

rs.Value2🡨 Array (see note “Assigning values to a range” below)

**Methods**

rs.Clear clear values and formats

rs.ClearContents clear values only

**Assigning values to a range or selection** works a little differently from the way *write* works, which constructs a range that matches the shape of the data provided. Here are some examples:

1. To assign a constant to each cell in the range, execute

rs.Value2🡨 25

1. To assign the same values to each row, assign a vector or 1-row matrix; Excel will truncate the vector if it is too long (compared to the number of columns in the range) or extend it with #N/A’s if it is too short:

rs.Value2🡨ι5

1. To assign the same values to each column, assign a 1-column matrix; as with the previous example, Excel will extend or truncate as necessary:

rs.Value2🡨 4 1ρι4

1. To assign an array, there are also no size constraints; Excel extends or truncates each dimension as necessary:

rs.Value2🡨 3 2ρ6?20

# Appendix A

**Practical Tips**

In working with Excel, it can be helpful to define various book or sheet objects so that you can easily re-activate them. For example:

wbk1 🡨 open ‘c:\projects\clients.xlsx’

wbk2 🡨 open ‘c:\projects\expenses.xlsx’

expenses.xlsx is now the active workbook. To revert to clients.xlsx, just execute

 wbk1.Activate

and later you can execute

 wbk2.Activate

The same principle applies to worksheets and other collections.

**General Errors**

Sometimes dialog boxes occur when the connection with Excel is broken. For example, you may see a dialog box with just a short alpha-numeric string (about 8 characters), or with the message ‘RPC\_E\_Disconnected’. These can arise in two situations:

* You terminate Excel but do not execute *endexcel*.
* You define an object (eg, bk 🡨 setbook’’) then close the workbook from Excel, then refer to a property such as bk.Name.

In both cases you have an Apl object that nominally exists but which has no connection to an Excel object. In the first case, the solution is to execute *endexcel* then *startexcel*.

You can trap this type of error in Apl (error 11) but you can’t avoid the display of the dialog box.

In other cases of broken connection you may encounter Value Error or Syntax error depending on the context. For example, if there are no open workbooks (wb.Count = 0), then xl.ActiveSheet returns zero and so xl.ActiveSheet.Name causes Syntax Error. However, referring to xl.Sheets and various other objects that depend on a workbook being open usually results in a dialog box with the message DISP\_E\_EXCEPTION.

Note that closing Excel from the Task Manager has a stronger effect than closing the Excel session via the File menu or the Close button or executing xl.Quit. If you perform these actions:

startexcel ’’

close Excel from the Task Manager

open ‘c:\apl\testwbk.xlsx’

you will see a dialog box with the message ‘Error: 800706BA’ (so execute *endexcel* then *startexcel*). However, if you close Excel by any other method mentioned above, the *open* command will work, though you will need to do

xl.Visible 🡨 1

# Appendix B

The Excel object includes many constants. No constants appear in any PropList, but xl.TypeList lists 231 methods in Excel 12, each of which returns a table of names of constants and their values that relate to a particular Excel feature. For example:

5↑xl.TypeList

Constants XlCreator XlChartGallery XlColorIndex XlEndStyleCap

 3↑[1] xl.Constants

xlAll ¯4104

xlAutomatic ¯4105

xlBoth 1

Two of these tables are shown below. Note that all such tables have 3 columns but column 2 is always empty.

**xl.XlChartType**

This is organized by generic chart type.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Constant** | **Value** | **Type** | **Constant** | **Value** |
| **Area** | xlArea | 1 | **Line** | xlLine | 4 |
| xlAreaStacked | 76 | xlLineMarkers | 65 |
| xlAreaStacked100 | 77 | xlLineMarkersStacked | 66 |
| xl3DArea | -4098 | xlLineMarkersStacked100 | 67 |
| xl3DAreaStacked | 78 | xlLineStacked | 63 |
| xl3DAreaStacked100 | 79 | xlLineStacked100 | 64 |
| **Bar** | xlBarClustered | 57 | xl3DLine | -4101 |
| xlBarStacked | 58 | **Pie** | xlPie | 5 |
| xlBarStacked100 | 59 | xlPieExploded | 69 |
| xl3DBarClustered | 60 | xlPieOfPie | 68 |
| xl3DBarStacked | 61 | xl3DPie | -4102 |
| xl3DBarStacked100 | 62 | xl3DPieExploded | 70 |
| **Bubble** | xlBubble | 15 | xlBarOfPie | 71 |
| xlBubble3DEffect | 87 | **Pyramid** | xlPyramidBarClustered | 109 |
| **Column** | xlColumnClustered | 51 | xlPyramidBarStacked | 110 |
| xlColumnStacked | 52 | xlPyramidBarStacked100 | 111 |
| xlColumnStacked100 | 53 | xlPyramidCol | 112 |
| xl3DColumn | -4100 | xlPyramidColClustered | 106 |
| xl3DColumnClustered | 54 | xlPyramidColStacked | 107 |
| xl3DColumnStacked | 55 | xlPyramidColStacked100 | 108 |
| xl3DColumnStacked100 | 56 | **Radar** | xlRadar | -4151 |
| **Cone** | xlConeBarClustered | 102 | xlRadarFilled | 82 |
| xlConeBarStacked | 103 | xlRadarMarkers | 81 |
| xlConeBarStacked100 | 104 | **Scatter** | xlXYScatter | -4169 |
| xlConeCol | 105 | xlXYScatterLines | 74 |
| xlConeColClustered | 99 | xlXYScatterLinesNoMarkers | 75 |
| xlConeColStacked | 100 | xlXYScatterSmooth | 72 |
| xlConeColStacked100 | 101 | xlXYScatterSmoothNoMarkers | 73 |
| **Cylinder** | xlCylinderBarClustered | 95 | **Stock** | xlStockHLC | 88 |
| xlCylinderBarStacked | 96 | xlStockOHLC | 89 |
| xlCylinderBarStacked100 | 97 | xlStockVHLC | 90 |
| xlCylinderCol | 98 | xlStockVOHLC | 91 |
| xlCylinderColClustered | 92 | **Surface** | xlSurface | 83 |
| xlCylinderColStacked | 93 | xlSurfaceTopView | 85 |
| xlCylinderColStacked100 | 94 | xlSurfaceTopViewWireframe | 86 |
| **Doughnut** | xlDoughnut | -4120 | xlSurfaceWireframe | 84 |
| xlDoughnutExploded | 80 |  |  |

**xl.XlFileFormat**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Value** | **Variable** | **Value** |
| xlAddIn | 18 | xlWJ2WD1 | 14 |
| xlCSV | 6 | xlWK1 | 5 |
| xlCSVMac | 22 | xlWK1ALL  | 31 |
| xlCSVMSDOS | 24 | xlWK1FMT  | 30 |
| xlCSVWindows | 23 | xlWK3  | 15 |
| xlDBF2 | 7 | xlWK4  | 38 |
| xlDBF3 | 8 | xlWK3FM3  | 32 |
| xlDBF4 | 11 | xlWKS | 4 |
| xlDIF  | 9 | xlWorks2FarEast | 28 |
| xlExcel2  | 16 | xlWQ1 | 34 |
| xlExcel2FarEast | 27 | xlWJ3  | 40 |
| xlExcel3  | 29 | xlWJ3FJ3 | 41 |
| xlExcel4  | 33 | xlUnicodeText | 42 |
| xlExcel5  | 39 | xlHtml  | 44 |
| xlExcel7  | 39 | xlWebArchive  | 45 |
| xlExcel9795  | 43 | xlXMLSpreadsheet  | 46 |
| xlExcel4Workbook  | 35 | xlExcel12  | 50 |
| xlIntlAddIn | 26 | xlOpenXMLWorkbook  | 51 |
| xlIntlMacro  | 25 | xlOpenXMLWorkbookMacroEnabled | 52 |
| xlWorkbookNormal  | ¯4143 | xlOpenXMLTemplateMacroEnabled | 53 |
| xlSYLK | 2 | xlTemplate8  | 17 |
| xlTemplate  | 17 | xlOpenXMLTemplate  | 54 |
| xlCurrentPlatformText | ¯4158 | xlAddIn8  | 18 |
| xlTextMac  | 19 | xlOpenXMLAddIn  | 55 |
| xlTextMSDOS  | 21 | xlExcel8  | 56 |
| xlTextPrinter  | 36 | xlWorkbookDefault  | 51 |
| xlTextWindows  | 20 |  |  |

# Appendix C: *xread* and *xwrite*

## Summary

The functions *xread* and *xwrite* are intended for users who want all-in-one higher-level interface functions, thereby simplifying the interaction with Excel. They are appropriate in situations where you just want to open a workbook, read or write one sheet, then close the workbook and, in that case, they could be the only functions in this package that you would explicitly use (but some other functions are invoked). For many reads and writes and for other operations, other functions in this package are more suitable or required.

*xread* and *xwrite* include extensive argument validation (all errors are trapped and reported as error number 500) and include Excel operations such as:

 starting an Excel session if necessary and optionally closing it afterwards

 opening or activating an existing workbook, and, in *xwrite*, creating a new workbook

 optionally saving (*xwrite* only) and closing a workbook.

Both functions invoke *startexcel*, which may start a new Excel session or connect to an existing session; *startexcel* creates two objects (*xl* and *wb*) in the APL environment which provide the interface to Excel. Both functions may open a workbook or activate an existing workbook and, in addition, *xwrite* may create a new workbook. Both functions optionally close the workbook (*xwrite* saves it first) and invoke either *endexcel* to erase the Excel objects from the workspace or *quitexcel* to terminate the Excel session as well.

Before using *xread* & *xwrite*, please review the documentation of *startexcel*, *endexcel* and *quitexcel*. The description of function arguments under ‘Function Details’ in the main body of this document applies also to *xread* and *xwrite*, but these functions tolerate both the long and short form (ie, with and without the path) of workbook names when referring to an open workbook.

***xread***

Syntax: R 🡨 {flags} xread (Workbook Worksheet Cell-Range)

The result *R* contains the values for the specified book, sheet and range. *R* has rank 0 or 1 (1 implies a character vector) where Cell-Range is a single cell or 2 for multiple cells. *flags* is a character vector which may contain any of the following (upper or lower case):

‘n’ create a new Excel session even if a session is already open

‘c’ close the workbook (without saving it)

‘e’ invoke *endexcel*

‘q’ invoke *quitexcel* (which itself invokes *endexcel*)

Note that *xread ‘?’* displays several lines of help text.

***xwrite***

Syntax: {flags} xwrite (Workbook Worksheet Cell Data)

The effect is to write the contents of Data to the specified book and sheet where Cell (a single cell) specifies the location of the first item of Data. Data may be a single number, a string, a character matrix, or a matrix of strings and/or numbers. A string, and each row of a character matrix, is allocated to a single cell. If Data is a vector, it is written to a single row of the sheet. *flags* is a character vector which may contain any of the following (upper or lower case):

‘n’ create a new Excel session even if a session is already open

‘c’ save and close the workbook (‘s’ is unnecessary if ‘c’ is specified)

‘s’ save the workbook (but leave it open unless ‘c’ is specified)

‘e’ invoke *endexcel*

‘q’ invoke *quitexcel* (which itself invokes *endexcel*)

Note that *xwrite ‘?’* displays several lines of help text.

## Some Simple Examples

1. Read a specified range:

R 🡨 xread ‘c:\apl\loan.xlsx’ ‘loan08’ ‘c7:g12’

 R

 Payment Interest Fee Principal Amount

 [Null] [Null] (1.5%) Reduction Owed

 ¯25000 [Null] [Null] [Null] 25000

 1000 625 9.38 365.62 24634.38

 1000 615.86 0 384.14 24250.24

 1500 606.26 0 893.74 23356.5

In this case, *xread* starts the Excel interface if necessary, opens or activates the workbook, activates the sheet and reads the cell range. No flags are specified so the workbook remains open and the Excel objects (*xl* and *wb*) survive in the workspace. This situation is helpful if you plan to use the workbook again.

1. Read a range but close the workbook and Excel afterwards:

R 🡨 ‘cq’ xread ‘c:\apl\loan’ ‘loan09’ ‘c7:g12’

The objects *xl* and *wb* have been erased, the workbook has been closed and the Excel session has been terminated. Specifying ‘c’ is technically unnecessary – when Excel shuts down, the workbook is closed anyway. However, if there are unsaved changes in this or any other workbook, Excel will prompt you by flashing the taskbar icon, for example. This is sometimes hard to see, but if you specify ‘c’ then *xread* will quit with an error if there are unsaved changes in the active workbook (but no other workbooks); this is a condition you can’t miss.

1. Start a new Excel session, write some data to the first sheet in a workbook and save it:

p 🡨 4 2 ρ‘ABT’ 125 ‘CSCO’ 400 ‘ORCL’ 350 ‘DIS’ 250 ⍝table of stock symbols and numbers of shares

‘ns’ xwrite ‘c:\docs\portfolio.xlsx’ 1 ‘b3’ p

Since ‘b3’ locates the first element of *p*, the portfolio in this example covers the range ‘b3:c6’.

If the workbook doesn’t already exist then *xwrite* creates it, in which case the ‘s’ flag is unnecessary. (That’s because of the way Excel works: a new workbook is named ‘Book’ followed by a numeric suffix; it’s only when it is saved that it acquires the specified file name.) The workbook remains open and the objects *xl* and *wb* remain active.

If portfolio.xlsx is an existing file, you can omit the suffix, but *xwrite* wil terminate with an error if you omit the suffix and the file doesn’t exist.

1. Read two different ranges from the same book and sheet:

R1 🡨 xread ‘c:\accounts\ledger.xlsx’ ‘2011’ ‘a4:h20’

R2 🡨 xread ‘’ ‘’ ‘aa4:ah20’

The first use of *xread* establishes the active book and sheet; the second example can therefore refer to them by empty vectors.

1. Identify workbooks by number:

 books’’

loan.xlsx c:\apl\loan.xlsx

portfolios.xlsx c:\Docs\portfolios.xlsx

 r1 🡨 xread 1 2 ‘a5:f16’ ⍝specify loan.xlsx, sheet 2

 r2 🡨 xread 2 3 ‘a1:g40’ ⍝specify portfolios.xlsx, sheet 3

1. Write data to a new, unsaved workbook, implied by the specification “book”:

xwrite ‘book’ 1 ‘a1’ (?12 3ρ24)

You can see the workbook that was created by this command by executing:

 books ’’

loan.xlsx c:\apl\loan.xlsx

portfolios.xlsx c:\Docs\portfolios.xlsx

book1 book1

The new book will be the last item shown; an unsaved workbook has neither path nor extension. When you specify ‘book’, flags ‘s’ and ‘c’ are ineffective; you will need *saveas* to save the book later.

## Specifying a Cell or Cell Range

There are three special range specifications that apply only to *xread*:

|  |  |
| --- | --- |
| ‘\*’ | The Used Range, an Excel term which means the smallest rectangle that covers every defined cell in the sheet (this can include empty but formatted cells) |
| ‘’ | The current selection, ie, the area you select by dragging the mouse over some cells, rows or columns or by clicking the top left corner to select the whole sheet |

You can also specify a (parenthesized) argument that is appropriate to the function *range* (please see the description in the main section of this document).

There is one special cell definition that applies to *xwrite*:

|  |  |
| --- | --- |
| ‘’ | The active cell; if you have selected a range, the active cell is the corner that you first selected before extending the range with the mouse or the keyboard |

## More Examples

1. Read the whole sheet:

R 🡨 xread ‘loan.xlsx’ ‘sheet2’ ‘\*’

1. To read a selection, drag your mouse over the cells you want to read, then read them by:

R 🡨 xread ‘loan’ 1 ‘’

Because a selection is a property of a sheet, if you activate another sheet or workbook, which probably has a different selection, then execute this example again, the result will be the same as before.

1. The act of making a selection with the mouse implicitly determines the active workbook and worksheet, so you can read the current selection via:

R 🡨 xread ‘’ ‘’ ‘’

If you select rows, columns, or the whole sheet, trailing empty rows and/or columns are removed from the result. (This does not apply if you specify ‘\*’ to select the used range.)

1. Write data to the active sheet using the active cell as the anchor point:

xwrite ‘’ ‘’ ‘’ (?4 2ρ24)

1. Write data to the active sheet using cell co-ordinates for the anchor point:

xwrite ‘’ ‘’ (5 1) (?4 2ρ24)

1. Read the data written in the last example:

R 🡨 xread ‘’ ‘’ (5 1 4 2)

Equivalently:

R 🡨 xread ‘’ ‘’ (‘A5’ 4 2)

1. Read a dynamic range (see the function *range*) but fix the number of columns:

R 🡨 xread ‘’ ‘’ (‘A5’ ⍬ 2)

Equivalently:

R 🡨 xread ‘’ ‘’ ((5 1) ⍬ 2)

1. Read a dynamic range based on D10:

R 🡨 xread ‘’ ‘’ (⊂ ’D10’)