Title

stata.com

xl() — Excel file I/O class

Description

The `xl()` class allows you to create Excel 1997/2003 (`.xls`) files and Excel 2007/2013 (`.xlsx`) files and load them from and to Mata matrices. The two Excel file types have different data size limits that you can read about in the technical note Excel data size limits of [D] **import excel**. The `xl()` class is supported on Windows, Mac, and Linux.

Syntax

If you are trying to import or export an Excel file to or from Stata, see [D] **import excel**. If you are trying to export a table created by Stata to Excel, see [P] **putexcel**.

The syntax diagrams below describe a Mata class. For help with class programming in Mata, see [M-2] **class**.

Syntax is presented under the following headings:

Step 1: Initialization

B = `xl()`

Step 2: Creating and opening an Excel workbook

(void) *B*`.create_book("`*filename*`", "`*sheetname*`" [ , {"xls"|"xlsx"}, "`*locale*`"])`

(void) *B*`.load_book("`*filename*`" [ , "`*locale*`"])`

(void) *B*`.clear_book("`*filename*`")`

(void) *B*`.set_mode("open"|"closed")`

(void) *B*`.close_book()`

Step 3: Setting the Excel worksheet

(void) *B*.add_sheet("*sheetname*")

(void) *B*.set_sheet("*sheetname*")

(void) *B*.set_sheet_gridlines("*sheetname*", {"on"|"off"})

(void) *B*.set_sheet_merge("*sheetname*", *real vector row*, *real vector col*)

(void) *B*.clear_sheet("*sheetname*")

(void) *B*.delete_sheet("*sheetname*")

(void) *B*.delete_sheet_merge("*sheetname*", *real vector row*, *real vector col*)

string colvector *B*.get_sheets()

Step 4: Reading and writing data from and to an Excel worksheet

string matrix *B*.get_string(*real vector row*, *real vector col*)

real matrix *B*.get_number(*real vector row*, *real vector col* [, {"asdate"|"asdatetime"}])

string matrix *B*.get_cell_type(*real vector row*, *real vector col*)

(void) *B*.put_string(*real scalar row*, *real scalar col*, *string matrix s*)

(void) *B*.put_number(*real scalar row*, *real scalar col*, *real matrix r* [, {"asdate"|"asdatetime"|"asdatenum"|asdatetimenum}])

(void) *B*.put_formula(*real scalar row*, *real scalar col*, *string matrix s*)

(void) *B*.put_picture(*real scalar row*, *real scalar col*, "*filename*")

(void) *B*.set_missing([*real scalar num*|*string scalar val*])

Step 5: Formatting cells in an Excel worksheet

(void) *B*.set_number_format(*real vector row*, *real vector col*, "*format*")

(void) *B*.set_vertical_align(*real vector row*, *real vector col*, "*align*")

(void) *B*.set_horizontal_align(*real vector row*, *real vector col*, "*align*")

(void) *B*.set_border(*real vector row*, *real vector col*, "*style*" [, "*color*"])

(void) *B*.set_left_border(*real vector row*, *real vector col* "*style*" [, "*color*"])

(void) *B*.set_right_border(*real vector row*, *real vector col*, "*style*" [, "*color*"])

(void) *B*.set_top_border(*real vector row*, *real vector col*, "*style*" [, "*color*"])

(void) *B*.set_bottom_border(*real vector row*, *real vector col*, "*style*" [, "*color*"])

(void) *B*.set_diagonal_border(*real vector row*, *real vector col*, "*direction*", "*style*" [, "*color*"])

(void) *B*.set_fill_pattern(*real vector row*, *real vector col*, "*pattern*", "*fgcolor*" [, "*bgcolor*"])

(void) *B*.set_column_width(*real scalar col1*, *real scalar col2*, *real scalar width*)

(void) *B*.set_row_height(*real scalar row1*, *real scalar row2*, *real scalar height*)

Step 6: Formatting text in an Excel worksheet

(void) *B*.set_font(*real vector row*, *real vector col*, "*fontname*", *real scalar size* [, "*color*"])

(void) *B*.set_font_bold(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_font_italic(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_font_strikeout(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_font_underline(*real vector row*, *real vector col* {"on"|"off"})

(void) *B*.set_font_script(*real vector row*, *real vector col*, {"sub"|"super"|"normal"})

(void) *B*.set_text_wrap(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_shrink_to_fit(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_text_rotate(*real vector row*, *real vector col*, *real scalar rotation*)

(void) *B*.set_text_indent(*real vector row*, *real vector col*, *real scalar indent*)

(void) *B*.set_format_lock(*real vector row*, *real vector col*, {"on"|"off"})

(void) *B*.set_format_hidden(*real vector row*, *real vector col*, {"on"|"off"})

Step 7: Formatting cell ranges in an Excel worksheet

real scalar	*B*.add_fmtid()
(void)	*B*.set_fmtid(*real vector row*, *real vector col*, *real scalar fmtid*)
(void)	*B*.fmtid_set_number_format(*real scalar fmtid*, "*format*")
(void)	*B*.fmtid_set_vertical_align(*real scalar fmtid*, "*align*")
(void)	*B*.fmtid_set_horizontal_align(*real scalar fmtid*, "*align*")
(void)	*B*.fmtid_set_border(*real scalar fmtid*, "*style*" [, "*color*"])
(void)	*B*.fmtid_set_left_border(*real scalar fmtid*, "*style*" [, "*color*"])
(void)	*B*.fmtid_set_right_border(*real scalar fmtid*, "*style*" [, "*color*"])
(void)	*B*.fmtid_set_top_border(*real scalar fmtid*, "*style*" [, "*color*"])
(void)	*B*.fmtid_set_bottom_border(*real scalar fmtid*, "*style*" [, "*color*"])
(void)	*B*.fmtid_set_diagonal_border(*real scalar fmtid*, "*direction*", "*style*" [, "*color*"])
(void)	*B*.fmtid_set_fill_pattern(*real scalar fmtid*, "*pattern*", "*fgcolor*" [, "*bgcolor*"])
(void)	*B*.fmtid_set_column_width(*real scalar fmtid*, *real scalar col1*, *real scalar col2*, *real scalar width*)
(void)	*B*.fmtid_set_row_height(*real scalar fmtid*, *real scalar row1*, *real scalar row2*, *real scalar height*)
(void)	*B*.fmtid_set_text_wrap(*real scalar fmtid*, {"on"|"off"})
(void)	*B*.fmtid_set_shrink_to_fit(*real scalar fmtid*, {"on"|"off"})
(void)	*B*.fmtid_set_text_rotate(*real scalar fmtid*, *real scalar rotation*)
(void)	*B*.fmtid_set_text_indent(*real scalar fmtid*, *real scalar indent*)
(void)	*B*.fmtid_set_format_lock(*real scalar fmtid*, {"on"|"off"})
(void)	*B*.fmtid_set_format_hidden(*real scalar fmtid*, {"on"|"off"})
real scalar	*B*.add_fontid()
(void)	*B*.fmtid_set_fontid(*real scalar fmtid*, *real scalar fontid*)
(void)	*B*.fontid_set_font(*real scalar fontid*, "*fontname*", *real scalar size* [, "*color*"])

(void)	*B*.fontid_set_font_bold(*real scalar fontid*, {"on"|"off"})
(void)	*B*.fontid_set_font_italic(*real scalar fontid*, {"on"|"off"})
(void)	*B*.fontid_set_font_strikeout(*real scalar fontid*, {"on"|"off"})
(void)	*B*.fontid_set_font_underline(*real scalar fontid*, {"on"|"off"})
(void)	*B*.fontid_set_font_script(*real scalar fontid*, {"sub"|"super"|"normal"})

Utility functions for use in all steps

(varies)	*B*.query(["*item*"])
real vector	*B*.get_colnum(*string vector*)
string vector	*B*.get_colletter(*real vector*)
(void)	*B*.set_keep_cell_format("on"|"off")
(void)	*B*.set_error_mode("on"|"off")
real scalar	*B*.get_last_error()
string scalar	*B*.get_last_error_message()

where *item* can be

```
filename
mode
filetype
sheetname
missing
```

Remarks and examples

stata.com

Remarks are presented under the following headings:

Definition of B

A variable of type `xl` is called an instance of the `xl()` class. *B* is an instance of `xl()`. You can use the class interactively:

```
b = xl()
b.create_book("results", "Sheet1")
...
```

In a function, you would declare one instance of the `xl()` class *B* as a `scalar`.

```
void myfunc()
{
    class xl scalar   b
    b = xl()
    b.create_book("results", "Sheet1")
    ...
}
```

When using the class inside other functions, you do not need to create the instance explicitly as long as you declare the member-instance variable to be a `scalar`:

```
void myfunc()
{
    class xl scalar   b
     b.create_book("results", "Sheet1")
    ...
}
```

Specifying the Excel workbook

To read from or write to an existing Excel workbook, you need to tell the `xl()` class about that workbook. To create a new workbook to write to, you need to tell the `xl()` class what to name that workbook and what type of Excel file that workbook should be. Excel 1997/2003 (`.xls`) files and Excel 2007/2010 (`.xlsx`) files can be created. You must either load or create a workbook before you can use any sheet or read or write *member functions* of the `xl()` class.

B`.create_book("`*filename*`", "`*sheetname*`" [ , { "xls" | "xlsx" }, "`*locale*`" ])`
creates an Excel workbook named *filename* with the sheet *sheetname*. By default, an `.xlsx` file is created. If you use the optional `xls` argument, then an `.xls` file is created. *locale* specifies the locale used by the workbook. You might need this option when working with extended ASCII character sets. This option has no effect on Microsoft Windows. The default locale is UTF-8.

B`.load_book("`*filename*`" [ , "`*locale*`" ])`
loads an existing Excel workbook. Once it is loaded, you can read from or write to the workbook. *locale* specifies the locale used by the workbook. You might need this option when working with extended ASCII character sets. This option has no effect on Microsoft Windows. The default locale is UTF-8.

B`.clear_book("`*filename*`")`
removes all worksheets from an existing Excel workbook.

To create an `.xlsx` workbook, code

```
b = xl()
b.create_book("results", "Sheet1", "xlsx")
```

To load an `.xls` workbook, code

```
b = xl()
b.load_book("Budgets.xls")
```

The `xl()` class will open and close the workbook for each member function you use that reads from or writes to the workbook. This is done by default, so you do not have to worry about opening and closing a file handle. This can be slow if you are reading or writing data at the cell level. In these cases, you should leave the workbook open, deal with your data, and then close the workbook. The following member functions allow you to control how the class handles file I/O.

B`.set_mode("open" | "closed")`
sets whether the workbook file is left open for reading or writing data. `set_mode("closed")`, the default, means that the workbook is opened and closed after every sheet or read or write member function.

B`.close_book()`
closes a workbook file if the file has been left open using `set_mode("open")`.

Below is an example of how to speed up file I/O when writing data.

```
b = xl()
b.create_book("results", "year1")
b.set_mode("open")
for(i=1;i<10000;i++) {
        b.put_number(i,1,i)
        ...
}
b.close_book()
```

Specifying the Excel worksheet

The following member functions are used to set the active worksheet the `xl()` class will use to read data from or write data to. By default, if you do not specify a worksheet, the `xl()` class will use the first worksheet in the workbook.

B`.add_sheet("`*sheetname*`")`
adds a new worksheet named *sheetname* to the workbook and sets the active worksheet to that sheet.

B`.set_sheet("`*sheetname*`")`
sets the active worksheet to *sheetname* in the `xl()` class.

The following member functions are sheet utilities:

B`.set_sheet_gridlines("`*sheetname*`", { "on" | "off" })`
sets whether gridlines are displayed for *sheetname*. The default is `on`.

B`.set_sheet_merge("`*sheetname*`",` *row*`,` *col*`)`
merges the cells in *sheetname* for each Excel cell in the Excel cell range specified in *row* and *col*. Both *row* and *col* can be a 1×2 `real vector`. The first value in the vectors must be the starting (upper-left) cell in the Excel worksheet to which you want to merge. The second value must be the ending (lower-right) cell in the Excel worksheet to which you want to merge.

B`.clear_sheet("`*sheetname*`")`
clears all cell values for *sheetname*.

B`.delete_sheet("`*sheetname*`")`
deletes *sheetname* from the workbook.

B`.delete_sheet_merge("`*sheetname*`",` *row*`,` *col*`)`
deletes the merged cells in *sheetname* for any Excel cells merged with the cell specified by *row* and *col*.

B`.get_sheets()` returns a `string colvector` of all the sheetnames in the current workbook.

You may need to make a change to all the sheets in a workbook. `get_sheets()` can help you do this.

```
void myfunc()
{
    class xl scalar   b
    string colvector  sheets
    real scalar   i
    b.load_book("results")
    sheets = b.get_sheets()
    for(i=1;i<=rows(sheets);i++) {
        b.set_sheet(sheets[i])
        b.clear_sheet(sheets[i])
        ...
    }
}
```

To create a new workbook with multiple new sheets, code

```
b.create_book("Budgets", "Budget 2009")
for(i=10;i<=13;i++) {
        sheet = "Budget 20" + strofreal(i)
        b.add_sheet(sheet)
}
```

Reading data from Excel

The following member functions of the `xl()` class are used to read data. Both *row* and *col* can be a `real scalar` or a 1×2 `real vector`.

B`.get_string(`*row*`,` *col*`)`
: returns a string matrix containing values in a cell range depending on the range specified in *row* and *col*.

B`.get_number(`*row*`,` *col* [`,` { `"asdate"` | `"asdatetime"` }]`)`
: returns a `real matrix` containing values in an Excel cell range depending on the range specified in *row* and *col*.

B`.get_cell_type(`*row*`,` *col*`)`
: returns a `string matrix` containing the string values `numeric`, `string`, `date`, `datetime`, or `blank` for each Excel cell in the Excel cell range specified in *row* and *col*.

To get the value in cell `A1` from Excel into a `string scalar`, code

```
string scalar val
val = b.get_string(1,1)
```

If `A1` contained the value `"Yes"`, then `val` would contain `"Yes"`. If `A1` contained the numeric value `1`, then `val` would contain `"1"`. `get_string()` will convert numeric values to strings.

To get the value in cell `A1` from Excel into a `real scalar`, code

```
real scalar val
val = b.get_number(1,1)
```

If A1 contained the value "Yes", then val would contain a missing value. get_number will return a missing value for a string value. If A1 contained the numeric value 1, then val would contain the value 1.

To read a range of data into Mata, you must specify the cell range by using a 1×2 rowvector. To read row 1, columns B through F of a worksheet, code

```
string rowvector cells
real rowvector cols
cols = (2,6)
cells = b.get_string(1,cols)
```

To read rows 1 through 3 and columns B through D of a worksheet, code

```
real matrix cells
real rowvector rows, cols
rows = (1,3)
cols = (2,4)
cells = b.get_number(rows,cols)
```

Writing data to Excel

The following member functions of the xl() class are used to write data. *row* and *col* are real scalars. When you write a matrix or vector, *row* and *col* are the starting (upper-left) cell in the Excel worksheet to which you want to begin saving.

B.put_string(*row*, *col*, *s*)
writes a string scalar, vector, or matrix to an Excel worksheet.

B.put_number(*row*, *col*, *r*[, {"asdate" | "asdatetime" | "asdatenum" | asdatetimenum}])
writes a real scalar, vector, or matrix to an Excel worksheet.

B.put_formula(*row*, *col*, *s*)
writes a string scalar, vector, or matrix containing valid Excel formulas to an Excel worksheet.

B.put_picture(*row*, *col*, *filename*)
writes a portable network graphics (.png), JPEG (.jpg), Windows metafile (.wmf), device-independent bitmap (.dib), enhanced metafile (.emf), or tagged image file format (.tiff) file to an Excel worksheet.

To write the string "Auto Dataset" in cell A1 of a worksheet, code

```
b.put_string(1, 1, "Auto Dataset")
```

To write "mpg", "rep78", and "headroom" to cells B1 through D1 in a worksheet, code

```
names = ("mpg", "rep78", "headroom")
b.put_string(1, 2, names)
```

To write values 22, 17, 22, 20, and 15 to cells B2 through B6 in a worksheet, code

```
mpg_vals = (22\17\22\20\15)
b.put_number(2, 2, mpg_vals)
```

To sum the cells A1 through A4 in cell A6 in a worksheet, code

```
b.put_formula(1, 6, "SUM(A1:A4)")
```

To write the file mygraph.png to starting cell D15 in a worksheet, code

```
b.put_picture(4, 15, "mygraph.png")
```

Dealing with missing values

set_missing() sets how Mata missing values are to be treated when writing data to a worksheet. Here are the three syntaxes:

B.set_missing() specifies that missing values be written as blank cells. This is the default.

B.set_missing(*num*) specifies that missing values be written as the real scalar *num*.

B.set_missing(*val*) specifies that missing values be written as the string scalar *val*.

Let's look at an example.

```
my_mat = J(1,3,.)
b.load_book("results")
b.set_sheet("Budget 2012")
b.set_missing(-99)
b.put_number(1, 1, my_mat)
b.set_missing("no data")
b.put_number(2, 1, my_mat)
b.set_missing()
b.put_number(3, 1, my_mat)
```

This code would write the numeric value -99 in cells A1 through C1 and "no data" in cells A2 through C2; cells A3 through C3 would be blank.

Dealing with dates

Say that cell A1 contained the date value 1/1/1960. If you coded

```
mydate = b.get_number(1,1)
mydate
21916
```

the value displayed, 21916, is the number of days since 31dec1899. This is how Excel stores its dates. If we used the optional get_number() argument "asdate" or "asdatetime", mydate would contain 0 because the date 1/1/1960 is 0 for both *td* and *tc* dates. To store 1/1/1960 in Mata, code

```
mysdate = b.get_string(1,1)
mysdate
1/1/1960
```

To write dates to Excel, you must tell the xl() class how to convert the date to Excel's date or datetime format. To write the date 1/1/1960 12:00:00 to Excel, code

```
b.put_number(1,1,0, "asdatetime")
```

To write the dates 1/1/1960, 1/2/1960, and 1/3/1960 to Excel column A, rows 1 through 3, code

```
date_vals = (0\1\2)
b.put_number(1, 1, date_vals, "asdate")
```

"asdate" and "asdatetime" apply an Excel date format to the transformed date value when written. Use "asdatenum" or "asdatetimenum" to write the transformed Excel date number and preserve the cell's format.

Note: Excel has two different date systems; see the technical note Dates and times in [D] **import excel**.

Formatting functions

The following member functions of the xl() class are used to format cells of the active worksheet. Both *row* and *col* can be a real scalar or a 1×2 real vector. The first value in the vectors must be the starting (upper-left) cell in the Excel worksheet to which you want to format. The second value must be the ending (lower-right) cell in the Excel worksheet to which you want to format.

Numeric formatting

B.set_number_format(*row*, *col*, "*format*")
sets the numeric format for each Excel cell in the Excel cell range specified in *row* and *col*.

Text alignment

B.set_vertical_align(*row*, *col*, "*align*")
sets the text to vertical alignment for each Excel cell in the Excel cell range specified in *row* and *col*. *align* may be "top", "center", "bottom", "justify", or "distributed".

B.set_horizontal_align(*row*, *col*, "*align*")
sets the text to horizontal alignment for each Excel cell in the Excel cell range specified in *row* and *col*. *align* may be "left", "center", "right", "fill", "justify", "merge", or "distributed".

Cell borders

B.set_border(*row*, *col*, "*style*" [, "*color*"])
sets the top, left, right, and bottom border style and color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_left_border(*row*, *col*, "*style*" [, "*color*"])
sets the left border style and color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_right_border(*row*, *col*, "*style*" [, "*color*"])
sets the right border style and color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_top_border(*row*, *col*, "*style*" [, "*color*"])
sets the top border style and color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_bottom_border(*row*, *col*, "*style*" [, "*color*"])
sets the bottom border style and color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_diagonal_border(*row*, *col*, "*direction*", "*style*" [, "*color*"])
sets the diagonal border direction, style, and color for each Excel cell in the Excel cell range specified in *row* and *col*. *direction* may be "none", "down", "up", or "both".

B.set_fill_pattern(*row*, *col*, "*pattern*", "*fgcolor*" [, "*bgcolor*"])
sets the fill color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_column_width(*col1*, *col2*, *width*)
sets the column width for each Excel cell in the Excel cell column range specified in *col1* through *col2*. Column width is measured as the number of characters (0–255) rendered in Excel's default style's font.

B.set_row_height(*row1*, *row2*, *height*)
sets the row height for each Excel cell in the Excel cell row range specified in *row1* through *row2*. height is measured in point size.

Fonts

The following member functions of the xl() class are used to format text of a given cell in the active worksheet. Both *row* and *col* can be a real scalar or a 1×2 real vector. The first value in the vectors must be the starting (upper-left) cell in the Excel worksheet that you want to format. The second value must be the ending (lower-right) cell in the Excel worksheet that you want to format.

B.set_font(*row*, *col*, "*fontname*", *size* [, "*color*"])
sets the font, font size, and font color for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_font_bold(*row*, *col*, {"on"|"off"})
bolds or unbolds text for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_font_italic(*row*, *col*, {"on"|"off"})
italicizes or unitalicizes text for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_font_strikeout(*row*, *col*, {"on"|"off"})
strikesout or unstrikesout text for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_font_underline(*row*, *col*, {"on"|"off"})
underlines or ununderlines text for each Excel cell in the Excel cell range specified in *row* and *col*.

B.set_font_script(*row*, *col*, {"sub"|"super"|"normal"})
sets the script type for each Excel cell in the Excel cell range specified in *row* and *col*.

Other

The following member functions of the `xl()` class control other various cell formatting for a given cell in the active worksheet. Both *row* and *col* can be a `real scalar` or a 1×2 `real vector`. The first value in the vectors must be the starting (upper-left) cell in the Excel worksheet to which you want to format. The second value must be the ending (lower-right) cell in the Excel worksheet to which you want to format.

B`.set_text_wrap(`*row*`,` *col*`, {"on"|"off"})`
sets whether text is wrapped for each Excel cell in the Excel cell range specified in *row* and *col*.

B`.set_shrink_to_fit(`*row*`,` *col*`, {"on"|"off"})`
sets whether text is shrunk-to-fit the cell width for each Excel cell in the Excel cell range specified in *row* and *col*.

B`.set_text_rotate(`*row*`,` *col*`,` *rotation*`)`
sets the text rotation for each Excel cell in the Excel cell range specified in *row* and *col*.

B`.set_text_indent(`*row*`,` *col*`,` *indent*`)`
sets the text indention for each Excel cell in the Excel cell range specified in *row* and *col*. *indent* must be an integer less than or equal to 15.

B`.set_format_lock(`*row*`,` *col*`, {"on"|"off"})`
sets the locked protection property for each Excel cell in the Excel cell range specified in *row* and *col*.

B`.set_format_hidden(`*row*`,` *col*`, {"on"|"off"})`
sets the hidden protection property for each Excel cell in the Excel cell range specified in *row* and *col*.

Formatting examples

To change a cell's numeric format so that a number has commas and two decimal points and places all negative numbers in braces (`number_sep_d2_negbra`) for rows 2 through 7 and columns 2 through 4 for a worksheet, code

```
real rowvector rows, cols
b = xl()
...
rows = (2,7)
cols = (2,4)
b.set_number_format(rows, cols, "number_sep_d2_negbra")
```

To add a medium-thick border to all cell sides for the same cell range, code

```
b.set_border(rows, cols, "medium")
```

To change the font and font color for rows 1 through 7, column 1, code

```
rows = (1,7)
b.set_font(rows, 1, "Arial", 12, "white")
```

and to change the background fill color of the same cells, code

```
b.set_fill_pattern(rows, 1, "solid", "white", "lightblue")
```

To bold the text in cell B1 through C3, code

```
rows = (1,3)
cols = (2,3)
b.set_font_bold(rows, cols, "on")
```

Range formatting functions

By default, the `xl()` class creates a new format ID for each font and cell format change you make in a workbook using the standard `xl()` class formatting functions. Depending on how many format changes you make to a workbook, the number of format IDs can cause the Excel workbook to become so large that it will open slowly in Excel. To prevent this, you can create a format ID with specific font and format settings and apply that format ID across a cell range. Once a format ID has been attached to a cell range, any changes to the format ID are automatically applied to the cells.

Font formatting also has its own ID system, but you must attach a font ID to a format ID for the font ID to apply to the cell range. You can use one font ID with multiple format IDs. There is a limit of 512 font IDs per workbook.

Adding format IDs

B`.add_fmtid()`
: returns a new format ID and adds it to the current workbook.

Setting formats by ID

B`.set_fmtid(`*row*`, `*col*`, `*fmtid*`)`
: sets the format ID for each cell in the Excel cell range specified in *row* and *col*.

Cell formatting functions

The cell formatting functions below are used when formatting cells using a format ID.

B`.fmtid_set_number_format(`*fmtid*`, "`*format*`")`
: sets the numeric format for the specified format ID.

B`.fmtid_set_vertical_align(`*fmtid*`, "`*align*`")`
: sets the vertical alignment of the text for the specified format ID. *align* may be `"top"`, `"center"`, `"bottom"`, `"justify"`, or `"distributed"`.

B`.fmtid_set_horizontal_align(`*fmtid*`, "`*align*`")`
: sets the horizontal alignment of the text for the specified format ID. *align* may be `"left"`, `"center"`, `"right"`, `"fill"`, `"justify"`, `"merge"`, or `"distributed"`.

B`.fmtid_set_border(`*fmtid*`, "`*style*`" [, "`*color*`"])`
: sets the top, left, right, and bottom border style and color for the specified format ID.

B`.fmtid_set_left_border(`*fmtid*`, "`*style*`" [, "`*color*`"])`
: sets the left border style and color for the specified format ID.

B`.fmtid_set_right_border(`*fmtid*`, "`*style*`" [, "`*color*`"])`
: sets the right border style and color for the specified format ID.

B.fmtid_set_top_border(*fmtid*, "*style*" [, "*color*"])
sets the top border style and color for the specified format ID.

B.fmtid_set_bottom_border(*fmtid*, "*style*" [, "*color*"])
sets the bottom border style and color for the specified format ID.

B.fmtid_set_diagonal_border(*fmtid*, "*direction*", "*style*" [, "*color*"])
sets the diagonal border direction, style, and color for the specified format ID. *direction* may be "none", "down", "up", or "both".

B.fmtid_set_fill_pattern(*fmtid*, "*pattern*", "*fgcolor*" [, "*bgcolor*"])
sets the fill color for the specified format ID.

B.fmtid_set_column_width(*fmtid*, *col1*, *col2*, *width*)
sets the column width for the specified format ID in the Excel column range specified in *col1* through *col2*. Column width is measured as the number of characters (0–255) rendered in Excel's default style's font.

B.fmtid_set_row_height(*fmtid*, *row1*, *row2*, *height*)
sets the row height for the specified format ID in the Excel row range specified in *row1* through *row2*. *height* is measured in point size.

B.fmtid_set_text_wrap(*fmtid*, {"on"|"off"})
sets whether text is wrapped for the specified format ID.

B.fmtid_set_shrink_to_fit(*fmtid*, {"on"|"off"})
sets whether text is shrunk to fit the cell width for the specified format ID.

B.fmtid_set_text_rotate(*fmtid*, *rotation*)
sets the text rotation for the specified format ID.

B.fmtid_set_text_indent(*fmtid*, *indent*)
sets the text indention for the specified format ID. *indent* must be an integer less than or equal to 15.

B.fmtid_set_format_lock(*fmtid*, {"on"|"off"})
sets the locked protection property for the specified format ID.

B.fmtid_set_format_hidden(*fmtid*, {"on"|"off"})
sets the hidden protection property for the specified format ID.

Adding font IDs

B.add_fontid()
returns a new font ID and adds it to the current workbook.

Setting font IDs for format IDs

B.fmtid_set_fontid(*fmtid*, *fontid*)
sets the font ID for the specified format ID.

Font formatting functions

The font formatting functions below are used when formatting fonts using a font ID.

B.fontid_set_font(*fontid*, "*fontname*", *size* [, "*color*"])
sets the font, font size, and font color for the specified font ID.

B.fontid_set_font_bold(*fontid*, {"on"|"off"})
bolds or unbolds text for the specified font ID.

B.fontid_set_font_italic(*fontid*, {"on"|"off"})
italicizes or unitalicizes text for the specified font ID.

B.fontid_set_font_strikeout(*fontid*, {"on"|"off"})
strikesout or unstrikesout text for the specified font ID.

B.fontid_set_font_underline(*fontid*, {"on"|"off"})
underlines or ununderlines text for the specified font ID.

B.fontid_set_font_script(*fontid*, {"sub"|"super"|"normal"})
sets the script type for the specified font ID.

Range formatting examples

To create a format ID with a numeric format that places all negative numbers in braces, uses commas for thousands separators, and specifies two digits after the decimal, (number_sep_d2_negbra), code

```
b = xl()
...
fmt_id1 = b.add_fmtid()
b.fmtid_set_number_format(fmt_id1, "number_sep_d2_negbra")
```

To also change the format ID to have a medium-thick border for all cell sides, code

```
b.fmtid_set_border(fmt_id1, "medium")
```

To apply these format changes for rows 2 through 7 and columns 2 through 4 for a worksheet, code

```
rows = (2,7)
cols = (2,4)
b.set_fmtid(rows, cols, fmt_id1)
```

To create a font ID with an Arial font and a font color of blue, code

```
font_id1 = b.add_fontid()
b.fontid_set_font(font_id1, "Arial", 12, "blue")
```

To apply these font changes to the format ID fmt_id1

```
b.fmtid_set_fontid(fmt_id1, font_id1)
```

To create a new format ID that sets the background fill color to lightblue, code

```
fmt_id2 = b.add_fmtid()
b.fmtid_set_fill_pattern(fmt_id2, "solid", "white", "lightblue")
```

To apply these format changes to cell A1 for a worksheet, code

```
b.set_fmtid(1, 1, fmt_id2)
```

To also apply the font_id1 font changes to row 1 column 1, type

```
b.fmtid_set_fontid(fmt_id2, font_id1)
```

By adding the font settings in font_id1 to fmt_id2, the font formatting is automatically applied to row 1 column 1.

Utility functions

The following functions can be used whenever you have an instance of the xl() class.

query() returns information about an xl() class. Here are the syntaxes for query():

void	*B*.query()
string scalar	*B*.query("filename")
real scalar	*B*.query("mode")
real scalar	*B*.query("filetype")
string scalar	*B*.query("sheetname")
transmorphic scalar	*B*.query("missing")

B.query()
lists the current values and settings of the class.

B.query("filename")
returns the filename of the current workbook.

B.query("mode")
returns 0 if the workbook is always closed by member functions or returns 1 if the current workbook is open.

B.query("filetype")
returns 0 if the workbook is of type .xls or returns 1 if the workbook is of type .xlsx.

B.query("sheetname")
returns the active sheetname in a string scalar.

B.query("missing")
returns J(1,0,.) (if set to blanks), a string scalar, or a real scalar depending on what was set with set_missing().

When working with different Excel file types, you need to know the type of Excel file you are using because the two file types have different column and row limits. You can use xl.query("filetype") to obtain that information.

```
...
if (xl.query("filetype")) {
        ...
}
else {
        ...
}
```

B.get_colnum()
returns a vector of column numbers based on the Excel column labels in the string vector argument.

To get the column number for Excel columns AA and AD, code

```
: mycol = ("AA","AD")
: col = b.get_colnum(mycol)
: col
        1    2
 1  |  27   30  |
```

B.get_colletter()
returns a vector of column letters based on the column numbers in the real vector argument.

To get the column letter for Excel columns 1 and 29, code

```
: mycol = (1, 29)
: col = b.get_colletter(mycol)
: col
      1    2
 1  |  A   AC  |
```

The following function is used for cell formats and styles.

B.set_keep_cell_format("on"|"off")
sets whether the put_number() class member function preserves a cell's style and format when writing a value. By default, preserving a cell's style and format is off.

The following functions are used for error handling with an instance of class xl.

B.set_error_mode("on"|"off")
sets whether xl() class member functions issue errors. By default, errors are turned on.

B.get_last_error()
returns the last error code issued by the xl() class if set_error_mode() is set off.

B.get_last_error_message()
returns the last error message issued by the xl() class if set_error_mode() is set off.

Handling errors

Turning errors off for an instance of the xl() class is useful when using the class in an ado-file. You should issue a Stata error code in the ado-file instead of a Mata error code. For example, in Mata, when trying to load a file that does not exist within an instance, you will receive the error code r(16103):

```
: b = xl()
: b.load_book("zzz")
file zzz.xls could not be loaded
r(16103);
```

The correct Stata error code for this type of error is 603, not 16103. To issue the correct error, code

```
b = xl()
b.set_error_mode("off")
b.load_book("zzz")
if (b.get_last_error()==16103) {
        error(603)
}
```

You should also turn off errors if you set_mode("open") because you need to close your Excel file before exiting your ado-file. You should code

```
b = xl()
b.set_mode("open")
b.set_error_mode("off")
b.load_book("zzz")
...
b.put_string(1,300, "test")
if (b.get_last_error()==16116) {
        b.close_book()
        error(603)
}
```

If set_mode("closed") is used, you do not have to worry about closing the Excel file because it is done automatically.

Error codes

The error codes specific to the `xl()` class are the following:

Code	Meaning
16101	file not found
16102	file already exists
16103	file could not be opened
16104	file could not be closed
16105	file is too big
16106	file could not be saved
16111	worksheet not found
16112	worksheet already exists
16113	could not clear worksheet
16114	could not add worksheet
16115	could not read from worksheet
16116	could not write to worksheet
16121	invalid syntax
16122	invalid range
16130	could not read cell format
16131	could not write cell format
16132	invalid column format
16133	invalid column width
16134	invalid row format
16135	invalid row height
16136	invalid color
16140	invalid number format
16141	invalid alignment format
16142	invalid border style format
16143	invalid border direction format
16144	invalid fill pattern style format
16145	invalid font format
16146	invalid font size format
16147	invalid font name format
16148	invalid cell format

Appendix

Codes for numeric formats

format	Example
number	1000
number_d2	1000.00
number_sep	100,000
number_sep_d2	100,000.00
number_sep_negbra	(1,000)
number_sep_negbrared	(1,000)
number_d2_sep_negbra	(1,000.00)
number_d2_sep_negbrared	(1000.00)
currency_negbra	($4000)
currency_negbrared	($4000)
currency_d2_negbra	($4000.00)
currency_d2_negbrared	($4000.00)
account	5,000
accountcur	$ 5,000
account_d2	5,000.00
account_d2_cur	$ 5,000.00
percent	75%
percent_d2	75.00%
scientific_d2	10.00E+1
fraction_onedig	10 1/2
fraction_twodig	10 23/95
date	3/18/2007
date_d_mon_yy	18-Mar-07
date_d_mon	18-Mar
date_mon_yy	Mar-07
time_hmm_AM	8:30 AM
time_HMMSS_AM	8:30:00 AM
time_HMM	8:30
time_HMMSS	8:30:00
time_MMSS	30:55
time_HOMMSS	20:30:55
time_MMSS0	30:55.0
date_time	3/18/2007 8:30
text	this is text

Custom formatting

format also can be a custom code string formed by sections. 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. The following is a four section example:

```
#,###.00_);[Red](#,###.00);0.00;"sales "@
```

The following table describes the different symbols that are available for use in custom number formats:

Symbol	Description	Cell value	Fmt code	Cell displays
0	Digit placeholder (add zeros)	8.9	#.00	8.90
#	Digit placeholder (no zeros)	8.9	#.##	8.9
?	Digit placeholder (add space)	8.9	0.0?	8.9
.	Decimal point			
%	Percentage	.1	%	10%
,	Thousands separator			
E- E+ e- e+	Scientific format	12200000	0.00E+00	1.22E+07
$-+/():space	Display the symbol	12	(000)	(012)
\	Escape character	3	0\!	3!
*	Repeat character (fill in cell width)	3	3*	3xxxxx
_	Skip width of next character	−1.2	_0.0	1.2
"text"	Display text in quotes	1.23	0.00 "a"	1.23 a
@	Text placeholder	b	"a"@"c"	abc

The following table describes the different codes that are available for custom datetime formats:

Fmt code	Description	Cell displays
m	Months	1–12
mm	Months	01–12
mmm	Months	Jan–Dec
mmmm	Months	January–December
mmmmm	Months	J–D
d	Days	1–31
dd	Days	01–31
ddd	Days	Sun–Sat
dddd	Days	Sunday–Saturday
yy	Years	00–99
yyyy	Years	1909–9999
h	Hours	0–23
hh	Hours	00–23
m	Minutes	0–59
mm	Minutes	00–59
s	Seconds	0–59
ss	Seconds	00–59
h AM/PM	Time	5 AM
h:mm AM/PM	Time	5:36 PM
h:mm:ss A/P	Time	5:36:03 P
h:mm:ss.00	Time	5:34:03.75
[h]:mm	Elapsed time	1:22
[mm]:ss	Elapsed time	64:16
[ss].00	Elapsed time	3733.71

Custom formatting: Text color

To set the text color for a section of the format, type the name of one of the colors listed in the table under *Format colors* in square brackets in the section. The color must be the first item in the section.

Custom formatting: Conditional formatting

To set number formats that will be 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 the following:

Code	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

For example, the following format displays numbers that are less than or equal to 100 in a red font and numbers that are greater than 100 in 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.

Codes for border styles

style
none
thin
medium
dashed
dotted
thick
double
hair
medium_dashed
dash_dot
medium_dash_dot
dash_dot_dot
medium_dash_dot_dot
slant_dash_dot

Codes for fill pattern styles

pattern
none
solid
gray50
gray75
gray25
horstripe
verstripe
revdiagstripe
diagstripe
diagcrosshatch
thickdiagcrosshatch
thinhorstripe
thinverstripe
thinrevdiagstripe
thindiagstripe
thinhorcrosshatch
thindiagcrosshatch
gray12p5
gray6p25

Codes for text rotation

rotation	Meaning
0–90	text rotated counterclockwise 0 to 90 degrees
91–180	text rotated clockwise 1 to 90 degrees
255	vertical text

Format colors

color may be any of the color names listed below or an RGB (red, green, blue) value specified in double quotes ("255 255 255").

```
aliceblue                 deeppink
antiquewhite              deepskyblue
aqua                      dimgray
aquamarine                dodgerblue
azure                     firebrick
beige                     floralwhite
bisque                    forestgreen
black                     fuchsia
blanchedalmond            gainsboro
blue                      ghostwhite
blueviolet                gold
brown                     goldenrod
burlywood                 gray
cadetblue                 green
chartreuse                greenyellow
chocolate                 honeydew
coral                     hotpink
cornflowerblue            indianred
cornsilk                  indigo
crimson                   ivory
cyan                      khaki
darkblue                  lavender
darkcyan                  lavenderblush
darkgoldenrod             lawngreen
darkgray                  lemonchiffon
darkgreen                 lightblue
darkkhaki                 lightcoral
darkmagenta               lightcyan
darkolivegreen            lightgoldenrodyellow
darkorange                lightgray
darkorchid                lightgreen
darkred                   lightpink
darksalmon                lightsalmon
darkseagreen              lightseagreen
darkslateblue             lightskyblue
darkslategray             lightslategray
darkturquoise             lightsteelblue
darkviolet                lightyellow
```

```
lime                    peru
limegreen               pink
linen                   plum
magenta                 powerblue
maroon                  purple
mediumaquamarine        red
mediumblue              rosybrown
mediumorchid            royalblue
mediumpurple            saddlebrown
mediumseagreen          salmon
mediumslateblue         sandybrown
mediumspringgreen       seagreen
mediumturquoise         seashell
mediumvioletred         sienna
midnightblue            silver
mintcream               skyblue
mistyrose               slateblue
moccasin                snow
navajowhite             springgreen
navy                    steelblue
oldlace                 tan
olive                   teal
olivedrab               thistle
orange                  tomato
orangered               turquoise
orchid                  violet
palegoldenrod           wheat
palegreen               white
paleturquoise           whitesmoke
palevioletred           yellow
papayawhip              yellowgreen
peachpuff
```

Note: `.xls` files can only contain 56 unique colors.

fgcolor may be any color name specified in *color* or an RGB (red, green, blue) value specified in double quotes (`"255 255 255"`).

bgcolor may be any color name specified in *color* or an RGB (red, green, blue) value specified in double quotes (`"255 255 255"`).

Also see

[M-4] **io** — I/O functions

[M-5] **_docx*()** — Generate Office Open XML (.docx) file

[M-5] **Pdf*()** — Create a PDF file

[D] **import excel** — Import and export Excel files

[P] **putdocx** — Generate Office Open XML (.docx) file

[P] **putexcel** — Export results to an Excel file

[P] **putpdf** — Create a PDF file