Title

stata.com

xl( ) — Excel file I/O class

Syntax Description Remarks and examples Also see

Syntax

If you are reading this entry for the first time, skip to *Description*. If you are trying to import or export an Excel file from or to 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" } ])`

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

(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*`.clear_sheet("`*sheetname*`")`

string colvector *B*`.get_sheets()`

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

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

Utility functions for use in all steps

(varies)	*B*.query(["*item*"])
real vector	*B*.get_colnum(*string 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
```

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.

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 `xl()` class about that workbook. To create a new workbook to write to, you need to tell `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 `xl()` class.

B`.create_book("`*filename*`", "`*sheetname*`", [ , { "xls" | "xlsx" } ])`
creates an Excel workbook named *filename* with the sheet *sheetname*. By default, an `.xls` file is created. If you use the optional `.xlsx` argument, then an `.xlsx` file is created.

B.load_book("*filename*")
loads an existing Excel workbook. Once it is loaded, you can read from or write to the workbook.

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()txt
```

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

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 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 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"}])
writes a real scalar, vector, or matrix 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)
```

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. 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 `xl()` class how to convert the date to Excel's date or datetime format. To write the date `1/1/1960 00:00:00` 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")
```

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

Utility functions

The following functions can be used whenever you have an instance of `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 setting 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

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

The following function is used for cell formats and styles.

B.set_keep_cell_format("on"|"off")
sets whether the put_number() class member functions preserve 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 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, "zzz.xls")
if (b.get_last_error()==16103) {
        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

Also see

[M-2] **class** — Object-oriented programming (classes)

[M-4] **io** — I/O functions

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

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

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