Title

stata.com

Pdf*() — Create a PDF file

Description Syntax Remarks and examples Also see

Description

The `Pdf*()` classes are used to programmatically create a PDF file. The `PdfDocument` class creates the overall file and is required. The other classes are `PdfParagraph`, which adds a paragraph to a `PdfDocument`; `PdfText`, which adds customized text; and `PdfTable`, which adds a table.

Syntax

The syntax diagrams describe a set of Mata classes for creating a PDF file. For help with class programming in Mata, see [M-2] **class**.

Syntax is presented under the following headings:

PdfDocument

The `PdfDocument` class is the foundation for creating a PDF file. Using the `PdfDocument` class is mandatory, but the remaining classes are not.

`PdfDocument()`

void `save(`*string scalar filename*`)`

void `close()`

void `setPageSize(`*string scalar sz*`)`

void `setLandscape(`*real scalar landscape*`)`

void `setMargins(`*real scalar left*`,` *real scalar top*`,`*real scalar right*`,` *real scalar bottom*`)`

void `setHAlignment(`*string scalar a*`)`

void `setLineSpace(`*real scalar sz*`)`

void `setBgColor(`*real scalar r*`,` *real scalar g*`,` *real scalar b*`)`

void `setColor(`*real scalar r*`,` *real scalar g*`,` *real scalar b*`)`

void `setFont(`*string scalar fontname* [`,` *string scalar style*]`)`

void `setFontSize(`*real scalar sz*`)`

void `addImage(`*string scalar filename* [`,` *real scalar cx*`,` *real scalar cy*]`)`

void addParagraph(*class* PdfParagraph *scalar p*)

void addTable(*class* PdfTable *scalar t*)

void addNewPage()

void addLineBreak()

PdfParagraph

The PdfParagraph class can be used to add a paragraph to a PdfDocument or PdfTable.

PdfParagraph()

void addString(*string scalar s*)

void addText(*class* PdfText *scalar t*)

void addLineBreak()

void clearContent()

void setFirstIndent(*real scalar sz*)

void setLeftIndent(*real scalar sz*)

void setRightIndent(*real scalar sz*)

void setTopSpacing(*real scalar sz*)

void setBottomSpacing(*real scalar sz*)

void setBgColor(*real scalar r*, *real scalar g*, *real scalar b*)

void setColor(*real scalar r*, *real scalar g*, *real scalar b*)

void setFont(*string scalar fontname* [, *string scalar style*])

void setFontSize(*real scalar sz*)

void setUnderline()

void setStrikethru()

void setHAlignment(*string scalar a*)

void setVAlignment(*string scalar a*)

void setLineSpace(*real scalar sz*)

PdfText

The PdfText class can be used to add customized text to a PdfParagraph.

PdfText()

void setBgColor(*real scalar r*, *real scalar g*, *real scalar b*)

void setColor(*real scalar r*, *real scalar g*, *real scalar b*)

void setFont(*string scalar fontname* [, *string scalar style*])

void setFontSize(*real scalar sz*)

void setUnderline()

void setStrikethru()

void setSuperscript()

void setSubscript()

void addString(*string scalar s*)

void clearContent()

PdfTable

The PdfTable class can be used to add a table to a PdfDocument.

PdfTable()

void init(*real scalar rows*, *real scalar cols*)

void setTotalWidth(*real scalar sz*)

void setColumnWidths(*real rowvector pct_v*)

void setWidthPercent(*real scalar pct*)

void setIndentation(*real scalar sz*)

void setHAlignment(*string scalar a*)

void setBorderWidth(*real scalar sz* [, *string scalar bordername*])

void setBorderColor(*real scalar r*, *real scalar g*, *real scalar b* [, *string scalar bordername*])

void setTopSpacing(*real scalar sz*)

void setBottomSpacing(*real scalar sz*)

void setCellContentString(*real scalar i*, *real scalar j*, *string scalar s*)

void setCellContentParagraph(*real scalar i*, *real scalar j*, *class* PdfParagraph *scalar p*)

void setCellContentImage(*real scalar i*, *real scalar j*, *string scalar filename*)

void setCellContentTable(*real scalar i*, *real scalar j*, *class* PdfTable *scalar tbl*)

void setCellHAlignment(*real scalar i*, *real scalar j*, *string scalar a*)

void setCellVAlignment(*real scalar i*, *real scalar j*, *string scalar a*)

void setCellBgColor(*real scalar i*, *real scalar j*, *real scalar r*, *real scalar g*, *real scalar b*)

void setCellBorderWidth(*real scalar i*, *real scalar j*, *real scalar sz* [, *string scalar bordername*])

void setCellBorderColor(*real scalar i*, *real scalar j*, *real scalar r*, *real scalar g*, *real scalar b* [, *string scalar bordername*])

void setCellMargin(*real scalar i*, *real scalar j*, *real scalar sz* [, *string scalar marginname*])

void setCellFont(*real scalar i*, *real scalar j*, *string scalar fontname* [, *string scalar style*])

void setCellFontSize(*real scalar i*, *real scalar j*, *real scalar size*)

void setCellColor(*real scalar i*, *real scalar j*, *real scalar r*, *real scalar g*, *real scalar b*)

void setCellSpan(*real scalar i*, *real scalar j*, *real scalar rowcount*, *real scalar colcount*)

void setCellRowSpan(*real scalar i*, *real scalar j*, *real scalar count*)

void setCellColSpan(*real scalar i*, *real scalar j*, *real scalar count*)

void setRowSplit(*real scalar i*, *real scalar split*)

void addRow(*real scalar i*)

void delRow(*real scalar i*)

void addColumn(*real scalar j*)

void delColumn(*real scalar j*)

void fillStataMatrix(*string scalar name*, *real scalar colnames*, *real scalar rownames* [, *string scalar fmt*])

void fillMataMatrix(*real matrix name* [, *real scalar i*, *real scalar j*, *string scalar fmt*])

void fillData(*real matrix i*, *rowvector j*, *real scalar vnames*, *real scalar obsno* [, *scalar selectvar*])

Remarks and examples

stata.com

Remarks are presented under the following headings:

PdfDocument class details

The `PdfDocument` class is the foundation for creating a PDF file. Using the `PdfDocument` class is required to create a PDF file. Unless otherwise noted, all sizes are measured in points where 1 point = 1/72 inch.

`PdfDocument()` is the constructor for the `PdfDocument` class.

`save(`*filename*`)` performs the final rendering of the PDF file and saves it to disk.

`close()` closes the `PdfDocument()` and releases associated memory.

`setPageSize(`*sz*`)` sets the page size of the PDF file. The possible values for *size* are `Letter`, `Legal`, `A3`, `A4`, `A5`, `B4`, and `B5`.

`setLandscape(`*landscape*`)` sets the page orientation of the PDF file to `landscape` or `portrait`. By default, the orientation is `portrait`. If *landscape* is `1`, the orientation is set to `landscape`.

`setMargins(`*left*`,` *top*`,` *right*`,` *bottom*`)` sets the page margins *left*, *top*, *right*, and *bottom*. The margins are used to define the canvas where all text, images, tables, etc., are drawn using absolute coordinates from (*left*, *top*). The canvas height is equal to the {page height − (*top* + *bottom*)}, and the canvas width is equal to {page width − (*left* + *right*)}.

`setHAlignment(`*a*`)` sets the horizontal page alignment. The possible alignment values are `left`, `right`, `center`, `justified`, and `stretch`. The default is `left` alignment.

`setLineSpace(`*sz*`)` sets the distance between lines of text for the document.

`setBgColor(`*r*`,` *g*`,` *b*`)` sets the background color of the document with the specified RGB values in the range [0, 255]. Note that only integer values can be used.

`setColor(`*r*`,` *g*`,` *b*`)` sets the text color of the document using the specified RGB values in the range [0, 255]. Note that only integer values can be used.

`setFont(`*fontname* [`,` *style*]`)` sets the font for the document. Optionally, the font style may be specified. The possible values for the *style* are `Regular`, `Bold`, `Italic`, and `Bold Italic`. If *style* is not specified, `Regular` is the default.

`setFontSize(`*sz*`)` sets the text size in points for the document.

`addImage(`*file* [`,` *cx*`,` *cy*]`)` adds an image to the document. The *filename* of the image can be either an absolute path or a relative path from the current working directory. *cx* and *cy* specify the width and height of the image, which are measured in points.

If *cx* is not specified, the width of the image is determined by the image information and the canvas width of the document. If *cx* is larger than the canvas width, the canvas width is used.

If *cy* is not specified, the height of the image is determined by the width and the aspect ratio of the image.

The supported image types are `.jpeg` and `.png`.

`addParagraph(`*class* `PdfParagraph` *p*`)` adds a new paragraph to the document.

`addTable(`*class* `PdfTable` *t*`)` adds a new table to the document.

`addNewPage()` adds a page break to the document and moves the current position to the top left corner of the new page.

`addLineBreak()` adds a line break to the document and moves the current position to the next line.

PdfParagraph class details

The `PdfParagraph` class can be used to add a paragraph to a `PdfDocument`. `PdfParagraph` inherits basic attributes from `PdfDocument` such as the background color, text color, and font. Unless otherwise noted, all sizes are measured in points where 1 point $= 1/72$ inch.

`PdfParagraph()` is the constructor for a `PdfParagraph`.

`addString(`*s*`)` appends a string to the end of the paragraph.

`addText(`*class* `PdfText` *t*`)` appends a `PdfText` object to the end of the paragraph.

`addLineBreak()` adds a line break to the end of the paragraph. This moves the current position to the beginning of the next line.

`clearContent()` clears the content from the paragraph. This is useful if the paragraph object has already been added to the document and you intend to reuse its attributes with new content.

`setFirstIndent(`*sz*`)` sets the indentation for the first line of the paragraph.

`setLeftIndent(`*sz*`)` sets the left indentation of the paragraph, which applies to all lines.

`setRightIndent(`*sz*`)` sets the right indentation of the paragraph, which applies to all lines. If the sum of the left and right indentations is greater than the canvas width, the left indentation will dominate, and the right indentation will be overridden and set to 0.

`setTopSpacing(`*sz*`)` sets the space above the paragraph.

`setBottomSpacing(`*sz*`)` sets the space below the paragraph.

`setBgColor(`*r*`, `*g*`, `*b*`)` sets the background color of the paragraph with the specified RGB values in the range $[0, 255]$. Note that only integer values can be used.

`setColor(`*r*`, `*g*`, `*b*`)` sets the text color of the paragraph with the specified RGB values in the range $[0, 255]$. Note that only integer values can be used.

`setFont(`*fontname* [`, `*style*]`)` sets the font for the paragraph. Optionally, the font style may be specified. The possible values for the *style* are `Regular`, `Bold`, `Italic`, and `Bold Italic`.

`setFontSize(`*sz*`)` sets the text size in points for the paragraph.

`setUnderline()` underlines the text in the entire paragraph.

setStrikethru() strikes through the text in the entire paragraph.

setHAlignment(*a*) sets the horizontal alignment of the paragraph. The possible alignment values are left, right, center, justified, and stretch.

setVAlignment(*a*) sets the vertical alignment of the paragraph. The possible alignment values are top, center, baseline, and bottom. This setting has a visible effect only when the paragraph contains characters with varying fonts.

setLineSpace(*sz*) sets the distance between lines of text for the paragraph.

PdfText class details

The PdfText class can be used to add customized text to a paragraph. The PdfText class can set properties such as the background color, text color, font, etc. Unless otherwise noted, all sizes are measured in points where 1 point $= 1/72$ inch.

PdfText() is the constructor for a PdfText object.

setBgColor(*r*, *g*, *b*) sets the background color of the text with the specified RGB values in the range $[0, 255]$. Note that only integer values can be used.

setColor(*r*, *g*, *b*) sets the color of the text with the specified RGB values in the range $[0, 255]$. Note that only integer values can be used.

setFont(*fontname* [, *style*]) sets the font for the text. Optionally, the font style may be specified. The possible values for the *style* are Regular, Bold, Italic, and Bold Italic.

setFontSize(*sz*) sets the text size in points for the text.

setUnderline() underlines the text.

setStrikethru() strikes through the text.

setSuperscript() sets the text to be a superscript.

setSubscript() sets the text to be a subscript.

addString(*s*) adds the string to the text.

clearContent() clears the text content. This is useful if the text object has already been added to the paragraph and you intend to reuse its attributes with new content.

PdfTable class details

The PdfTable class can be used to add a table to a PdfDocument. The PdfTable inherits basic attributes from the PdfDocument such as the background color, text color, and font. A PdfTable may contain up to 65,535 rows and up to 50 columns. Unless otherwise noted, all sizes are measured in points where 1 point $= 1/72$ inch.

PdfTable() is the constructor for a PdfTable.

init(*rows*, *cols*) must be invoked before editing the table or its attributes unless fillStataMatrix(), fillMataMatrix(), or fillData() will be used.

setTotalWidth(*sz*) sets the total width of the table. By default, the width of the table is equal to the canvas width.

`setColumnWidths(`*pct_v*`)` sets the width of each column for the table. By default, the width of each column will be equal. The column widths are specified as a fraction of the total table width. The length of the row vector must equal the number of columns in the table. The sum of the fractions must be 1.

`setWidthPercent(`*pct*`)` sets the width as a fraction of the canvas that the table will occupy. The value may be a number in the range (0, 1].

`setIndentation(`*sz*`)` sets the table indentation. Setting the indentation has no visible effect if the table width is equal to the canvas width.

`setHAlignment(`*a*`)` sets the horizontal alignment of the table. The possible alignment values are `left`, `right`, and `center`. `left` is the default. This setting has a visible effect only when the table width is less than the canvas width.

`setBorderWidth(`*sz* [`,` *bordername*]`)` sets the width of the table border. The possible values of *bordername* are `top`, `left`, `bottom`, `right`, `insideH`, `insideV`, and `all`, which identify, respectively, the top border, the left border, the bottom border, the right border, the inside horizontal edges border, the inside vertical edges border, and all borders. If *bordername* is not specified, `all` is used.

`setBorderColor(`*r*`,` *g*`,` *b* [`,` *bordername*]`)` sets the color of the table border using the specified RGB values in the range [0, 255]. Note that only integer values can be used. The possible values of *bordername* are `top`, `left`, `bottom`, `right`, `insideH`, `insideV`, and `all`, which identify, respectively, the top border, the left border, the bottom border, the right border, the inside horizontal edges border, the inside vertical edges border, and all borders. If *bordername* is not specified, `all` is used.

`setTopSpacing(`*sz*`)` sets the space above the table.

`setBottomSpacing(`*sz*`)` sets the space below the table.

`setCellContentString(`*i*`,` *j*`,` *s*`)` sets the cell content of the *i*th row and *j*th column to be text *s*.

`setCellContentParagraph(`*i*`,` *j*`,` *class* `PdfParagraph` *p*`)` sets the cell content of the *i*th row and *j*th column to be a `PdfParagraph`.

`setCellContentImage(`*i*`,` *j*`,` *filename*`)` sets the cell content of the *i*th row and *j*th column to be the image specified by *filename*. The width of the image fits the column width, while the height of the image is determined by the width of the image and its aspect ratio.

`setCellContentTable(`*i*`,` *j*`,` *class* `PdfTable` *tbl*`)` sets the cell content of the *i*th row and *j*th column to be a `PdfTable`.

`setCellHAlignment(`*i*`,` *j*`,` *a*`)` sets the horizontal alignment for the cell of the *i*th row and *j*th column. The possible values for *a* are `left`, `right`, `center`, `justified`, and `stretch`. `left` is the default.

`setCellVAlignment(`*i*`,` *j*`,` *a*`)` sets the vertical alignment for the cell of the *i*th row and *j*th column. The possible values for *a* are `top`, `middle`, and `bottom`. `top` is the default.

`setCellBgColor(`*i*`,` *j*`,` *r*`,` *g*`,` *b*`)` sets the background color of the cell of the *i*th row and *j*th column to be the specified RGB values in the range [0, 255]. Note that only integer values can be used.

`setCellBorderWidth(`*i*`, `*j*`, `*sz*` [ , `*bordername*` ])` sets the width of the specified border of the cell on the *i*th row and *j*th column. The possible values of *bordername* are `top`, `left`, `bottom`, `right`, and `all`, which identify, respectively, the top border, the left border, the bottom border, the right border, and all the borders. If *sz* equals 0, the border of the cell is not shown. If *bordername* is not specified, `all` is the default.

`setCellBorderColor(`*i*`, `*j*`, `*r*`, `*g*`, `*b* [, *bordername*])` sets the color of the specified border of the cell on the *i*th row and *j*th column to be the specified RGB values in the range [0, 255]. The possible values of *bordername* are `top`, `left`, `bottom`, `right`, and `all`, which identify, respectively, the top border, the left border, the bottom border, the right border, and all the borders. If *bordername* is not specified, `all` is the default.

`setCellMargin(`*i*`, `*j*`, `*sz* [, *marginname*])` sets the content margin of the cell of the *i*th row and *j*th column. The possible values of *marginname* are `top`, `left`, `bottom`, `right`, and `all`, which identify, respectively, the top margin, the left margin, the bottom margin, the right margin, and all the margins. If *marginname* is not specified, `all` is the default.

`setCellFont(`*i*`, `*j*`, `*fontname* [, *style*])` sets the font for the cell of the *i*th row and *j*th column. Optionally, the font style may be specified. The possible values for the *style* are `Regular`, `Bold`, `Italic`, and `Bold Italic`. If *style* is not specified, `Regular` is the default.

`setCellFontSize(`*i*`, `*j*`, `*size*`)` sets the text size in points for the cell of the *i*th row and *j*th column.

`setCellColor(`*i*`, `*j*`, `*r*`, `*g*`, `*b*`)` sets the text color for the cell of the *i*th row and *j*th column using the specified RGB values in the range [0, 255]. Note that only integer values can be used.

`setCellSpan(`*i*`, `*j*`, `*rowcount*`, `*colcount*`)` sets the cell of the *j*th column of the *i*th row to span *rowcount* cells vertically downward and span *colcount* cells horizontally to the right. This is equivalent to merging all the cells in the spanning range into the original cell (*i*, *j*). Any content in the spanning range other than the original cell (*i*, *j*) will be discarded.

`setCellRowSpan(`*i*`, `*j*`, `*count*`)` sets the cell of the *j*th column of the *i*th row to span *count* cells vertically downward. This is equivalent to merging all the cells in the vertical spanning range into the original cell (*i*, *j*). Any content in the spanning range other than the original cell (*i*, *j*) will be discarded. This function is equivalent to `setCellSpan(`*i*`, `*j*`, `*count*`, 1)`.

`setCellColSpan(`*i*`, `*j*`, `*count*`)` sets the cell of the *j*th column of the *i*th row to span *count* cells horizontally to the right. This is equivalent to merging all the cells in the horizontal spanning range into the original cell (*i*, *j*). Any content in the spanning range other than the original cell (*i*, *j*) will be discarded. This function is equivalent to `setCellSpan(`*i*`, `*j*`, 1, `*count*`)`.

`setRowSplit(`*i*`, `*split*`)` sets the *i*th row to split across multiple pages if *split* is not `0`. Otherwise, the *i*th row will be displayed starting from the top of the next page. This setting has no effect if the *i*th row can be displayed completely on one page.

`addRow(`*i*`)` adds a row to the table after the *i*th row. The range of *i* is from 0 to *rows*, where *rows* is the number of rows of the table. If *i* is specified as `0`, the row will be added before the first row. Otherwise, the row will be added after the *i*th row.

`delRow(`*i*`)` deletes the *i*th row from the table. The range of *i* is from 1 to *rows*, where *rows* is the number of rows of the table.

`addColumn(`*j*`)` adds a column to the table on the right of the *j*th column. The range of *j* is from 0 to *cols*, where *cols* is the number of columns of the table. If *j* is specified as `0`, the column will be added to the left of the first column, which is equivalent to adding a new first column. Otherwise, the column will be added to the right of the *j*th column.

`delColumn(`*j*`)` deletes the *j*th column from the table. The range of *j* is from 1 to *cols*, where *cols* is the number of columns of the table.

`fillStataMatrix(`*name*`,` *colnames*`,` *rownames* [`,` *fmt*]`)` fills the table with a Stata matrix. If *colnames* is not `0`, the first row of the table is filled with `matrix colnames`. If *rownames* is not `0`, the first column of the table is filled with `matrix rownames`. The elements of the matrix are formatted using *fmt*, if specified. Otherwise, `%12.0g` is used. The matrix is identified by *name*. If the matrix does not exist, an error code will be returned.

`fillMataMatrix(`*name* [`,` *i*`,` *j*`,` *fmt*]`)` fills the table with a Mata matrix. The matrix is identified by *name*. If *i* is specified, the matrix fills the table starting from the *i*th row. If *j* is specified, the matrix fills the table starting from the *j*th column. The elements of the Mata matrix are formatted using *fmt*, if specified. Otherwise, `%12.0g` is used. If the matrix does not exist, an error code will be returned.

`fillData(`*i*`,` *j*`,` *vnames*`,` *obsno* [`,` *selectvar*]`)` fills the table with the current Stata dataset in memory. If a value label is attached to the variable, the data are displayed using the value label. Otherwise, the data are displayed based on the display format. *i*, *j*, and *selectvar* are specified in the same way as `st_data()`. Factor variables and time-series-operated variables are not allowed. If *vnames* is not `0`, the first row of the table will be filled with variable names. If *obsno* is not `0`, the first column of the table will be filled with observation numbers.

Error codes

Functions can abort only if one of the input parameters does not meet the specification; for example, a string scalar is used when a real scalar is required. Functions return a negative error code when there is an error:

Negative code	Meaning
−17100	an error occurred
−17101	PDF file not created
−17102	value is missing
−17103	attribute failed to set
−17104	file not saved
−17105	image not added
−17106	table not added
−17107	paragraph not added
−17108	failed to add new page
−17109	failed to add line break
−17110	PDF file not closed
−17120	table not created
−17121	table failed to set table dimensions
−17122	table not initialized
−17123	cell index out of range
−17124	table failed to set cell value
−17125	table failed to fill Stata matrix
−17126	table failed to fill Mata matrix
−17127	table failed to fill data
−17130	paragraph not created
−17131	paragraph failed to add text
−17132	paragraph failed to clear content

Examples

Examples are presented under the following headings:

Add paragraph

```
mata:
pdf = PdfDocument()
p = PdfParagraph()
p.addString("This is our first example of a paragraph. ")
p.addString("Let's add another sentence. ")
p.addString("Now we will conclude our first paragraph.")
pdf.addParagraph(p)
p = PdfParagraph()
p.setFirstIndent(36)
p.setFontSize(14)
p.setTopSpacing(10)
p.addString("This is our second paragraph. ")
p.addString("The first line of this paragraph has an indentation of ")
p.addString("36 points or 1/2 inch. The font size of this paragraph ")
p.addString("is 14.")
pdf.addParagraph(p)
p = PdfParagraph()
p.setTopSpacing(10)
p.setFontSize(14)
p.setHAlignment("justified")
p.addString("This is our third paragraph. ")
p.addString("Notice that we have switched back to block mode, which ")
p.addString("means that there is not any indentation. ")
p.addString("You should also notice that this paragraph sets the ")
p.addString("alignment to justified.")
pdf.addParagraph(p)
pdf.save("paragraph1.pdf")
pdf.close()
// clean up
mata drop p
mata drop pdf
end
```

Add paragraph with customized text

```
mata:
pdf = PdfDocument()
t1 = PdfText()
t1.setSuperscript()
t1.addString("This is superscript text.")
t2 = PdfText()
t2.setSubscript()
t2.addString("This is subscript text.")
p = PdfParagraph()
p.addString("Hello, this is our paragraph's normal text. ")
p.addText(t1)
p.addText(t2)
p.addString("Here is some more text using the attributes from ")
p.addString("the paragraph. ")
t = PdfText()
t.setFont("Courier New")
t.setFontSize(18)
t.setColor(255, 0, 0)
t.addString("Here is an example of text that uses a large ")
t.addString("Courier New font.  Its text color is red. ")
p.addText(t)
```

```
p.addString("We could insert more text here using the paragraph's ")
p.addString("normal attributes. ")
t = PdfText()
t.setFontSize(30)
t.setColor(0, 0, 255)
t.setStrikethru()
t.addString("Here is one more example of adding modified text ")
t.addString("to a paragraph. ")
p.addText(t)
p.addString("Now we will conclude this paragraph with normal text.")
pdf.addParagraph(p)
pdf.save("text1.pdf")
pdf.close()
// clean up
mata drop t
mata drop t1
mata drop t2
mata drop p
mata drop pdf
end
```

Add table (simple example)

```
mata:
pdf = PdfDocument()
t = PdfTable()
t.init(5, 4)
A = (0.2,0.5,0.15,0.15)
t.setColumnWidths(A)
for(i=1; i<=5; i++) {
        for(j=1; j<=4; j++) {
                result = sprintf("This is cell(%g, %g)", i, j) ;
                t.setCellContentString(i, j, result)
        }
}
pdf.addTable(t)
pdf.save("table1.pdf")
pdf.close()
// clean up
mata drop t
mata drop pdf
end
```

Add table (table with header and footer)

```
mata:
pdf = PdfDocument()
t = PdfTable()
t.init(5, 4)
A = (0.2,0.5,0.15,0.15)
t.setColumnWidths(A)
t.setCellContentString(1, 1, "This is header")
t.setCellHAlignment(1, 1, "center")
t.setCellColSpan(1, 1, 4)
```

```
for(i=2; i<=4; i++) {
        for(j=1; j<=4; j++) {
                result = sprintf("This is cell(%g, %g)", i, j) ;
                t.setCellContentString(i, j, result)
        }
}
t.setCellContentString(5, 1, "This is footer")
t.setCellHAlignment(5, 1, "center")
t.setCellColSpan(5, 1, 4)
pdf.addTable(t)
pdf.save("table2.pdf")
pdf.close()
// clean up
mata drop t
mata drop pdf
end
```

Add table (table with graph)

```
sysuse citytemp, clear
tabulate division, matcell(freq) matrow(vlabel)
local tabvar division
histogram `tabvar', discrete frequency addlabel scheme(sj)
graph export census.png, replace width(2000)
mata:
freq = st_matrix("freq")
vlabel = st_matrix("vlabel")
svlabel = st_vlmap(st_varvaluelabel(st_local("tabvar")), vlabel)
colheader = ("Census Division","Freq.","Percent","Cum.")
nrows = rows(freq)+2
ncols = cols(colheader)
percent = 0
cum = 0
pdf = PdfDocument()
t1 = PdfTable()
t1.init(nrows, ncols)
for(i=1;i<=ncols;i++) {
        t1.setCellContentString(1,i,colheader[1,i])
}
for(i=1;i<=rows(freq);i++) {
        t1.setCellContentString(i+1,1,svlabel[i,1])
         output = sprintf("%9.0g", freq[i,1])
         t1.setCellHAlignment(i+1,2,"right")
         t1.setCellContentString(i+1,2,output)
         percent = freq[i,1]/sum(freq)*100
         output = sprintf("%9.2f", percent)
         t1.setCellHAlignment(i+1,3,"right")
         t1.setCellContentString(i+1,3,output)
         cum = cum + percent
         output = sprintf("%9.2f", cum)
         t1.setCellHAlignment(i+1,4,"right")
         t1.setCellContentString(i+1,4,output)
}
t1.setCellContentString(nrows,1, "Total")
output = sprintf("%9.0g", sum(freq))
t1.setCellHAlignment(nrows,2,"right")
```

```
t1.setCellContentString(nrows,2,output)
t1.setCellHAlignment(nrows,3,"right")
t1.setCellContentString(nrows,3,"100.00")

t2 = PdfTable()
t2.init(1,2)

t2.setCellContentTable(1,1,t1)
t2.setCellContentImage(1,2,"census.png")
t2.setCellVAlignment(1,2,"bottom")
t2.setBorderWidth(0)

pdf.addTable(t2)

pdf.save("table3.pdf")
pdf.close()

// clean up
mata drop t1
mata drop t2
mata drop pdf
end
```

Also see

[M-4] **io** — I/O functions

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

[M-5] **xl( )** — Excel file I/O class

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

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

[P] **putpdf** — Create a PDF file