

# MarkDoc Wiki Manual (v. 4.0.0)

E. F. Haghish

Department of Mathematics and Computer Science  
University of Southern Denmark


This manual is created from GitHub Wiki documentations automatically. GitHub Wiki is a very convenient way to document a software. Using `markdoc`, you can convert the documentation to PDF, HTML, or even Stata help files (`sthlp`). Read more about `markdoc` on <https://github.com/haghish/markdoc/>


**MarkDoc** is a general-purpose literate programming package for Stata. **MarkDoc** is very simple and intuitive to use and it supports creating dynamic documents interactively. The software has a considerable focus on making literate programming easy and intuitive for newbies. Moreover, it greatly values the readability of the source code and thus provide several options to keep the source code as plain as possible. Therefore, **MarkDoc** can be taught to undergraduate students in introductory statistics courses to boost active learning, document code, and practice statistical reporting. Based on my personal experiences in teaching statistics students enjoy taking notes in their script files and writing dynamic documents.

Not only students, but also lecturers can get benefit from **MarkDoc** for creating dynamic presentation slides, directly from Stata, which makes their slides to be easily updatable, reusable, and easy to create. Finally, advanced Stata programmers, can get benefit from **MarkDoc** for creating Stata help files in (`sthlp`) or pdf package vignette from their source code.


## Resources


 MarkDoc package vignette (PDF)


 Journal Article


 Examples

 Torture tests

 Release notes

 MarkDoc engine structure

 Need help? Ask your questions on [statalist.org](http://statalist.org)

 **Need more help?** Contact the author to plan a workshop in your department or company

## Features

**MarkDoc** has several unique features which makes it an ideal package for practicing literate programming at any level, from a complete newbie to an advanced programmer. In brief, it can:

- highlight the syntax of Stata commands in HTML and PDF
- Produce dynamic documents in several formats by:
  - converting *smcl* log file to any format
  - actively reproducing a dynamic document from a *do-file*
- Produce presentation slides in PDF or HTML
- Produce Stata package help files (*sthlp*) or package vignette (*pdf*, *html*, *latex*, *docx*, ...) from the source code
- Render LaTeX mathematical notations in Microsoft Word *docx*, OpenOffice *odt*, *html*, and *pdf*
- Capture a graph automatically from Stata and include it in the dynamic document or slides
- Include a stored image in the dynamic document
- Create dynamic tables
- Write dynamic text for interpreting the results
- Specify what commands or results should be included in the dynamic document
- Include external text files (*markdown*, *LaTeX*, *html*, *smcl*, etc.) in the dynamic document
- Provide a "standard" template with descriptions for creating template help files

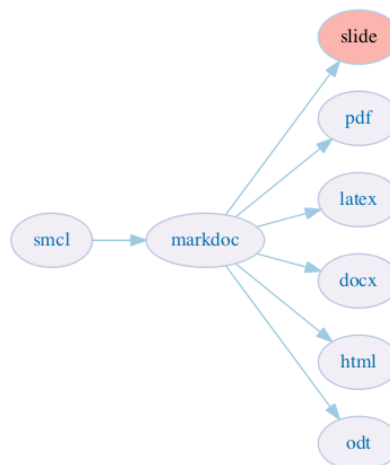
## Formats

The main idea of **MarkDoc** is that a single documentation format and a markup language should be able to produce a variety of document formats from the same source. For example, a graduate student should be able to produce an HTML output from a source code written with Markdown and also, use the same source to create a PDF analysis report, presentation slides, a LaTeX document, or a Microsoft Word document. This range of supported document formats makes "reusing" the documentation easy because a single format can be used for creating a variety of formats.

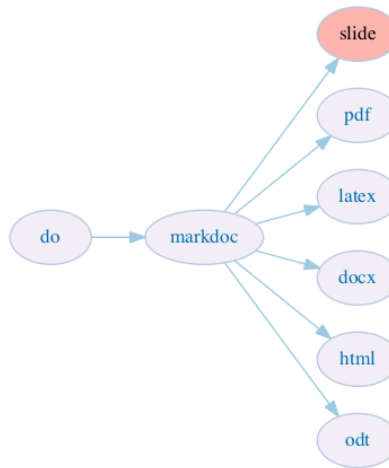
However, in addition to Markdown, **MarkDoc** recognizes 3 other markup languages for documentation which are LaTeX, HTML, and SMCL. **MarkDoc** applies the same format for documentation and can produce:

1. dynamic analysis document (*pdf*, *docx*, *tex*, *html*, *odt*, *epub*, *markdown*)
2. Stata package vignette (*pdf*, *docx*, *tex*, *html*, *odt*, *epub*, *markdown*)
3. dynamic presentation slides (*pdf*, *slidy*)
4. Stata help files(*sthlp*, *smcl*).

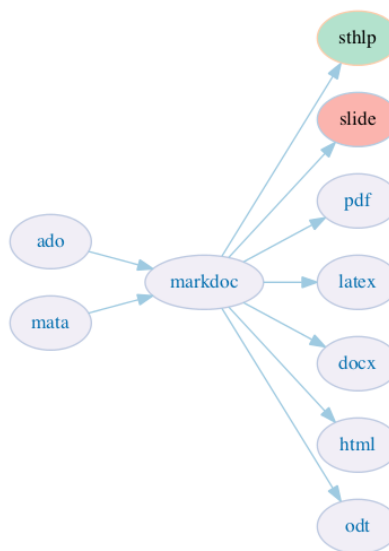
**MarkDoc** produces these document formats in several ways. Classically, **MarkDoc** takes a Stata *smcl* log file and converts it to any of the supported formats which are shown below.



In this case, **MarkDoc** processes the *smcl* file, which has a **.smcl** suffix and produces the document. The *smcl* file can be written in any of the supported markup languages which are Markdown (default), *html*, *latex*. A *smcl* log file that is written with Markdown can be converted to *pdf*, *docx*, *tex*, *html*, *odt*, *epub*, and *markdown*. If the *smcl* log file is written in *html* or *latex*, only a *pdf* and *html* or *latex* document can be exported respectively. Therefore, to get the maximum format compatibility from **MarkDoc**, Writing with Markdown is recommended compared to LaTeX or *html*.

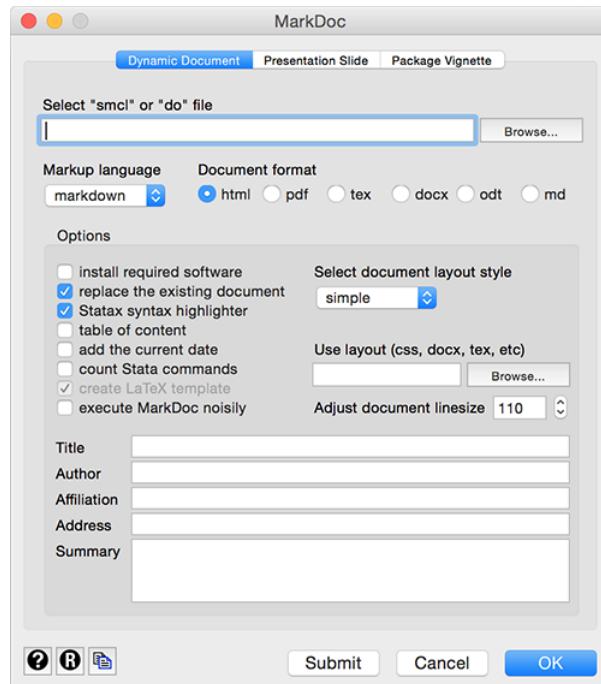


To ensure the analysis is **really reproducible**, converting a smcl file to a document is not enough. Instead a very restricted procedure is required to ensure the do-file loads the dataset that it uses for the analysis. In other words, the literate programming package should imagine there is no dataset loaded in Stata and then executes the do-file in a cleared workspace. When **MarkDoc** is given a Stata *do-file*, it executes the the script file in a new workspace and produces the dynamic document or presentation slides in any of the supported formats.



In addition, **MarkDoc** can also create a dynamic document, presentation slides, or package documentation from Stata script files which have the **.do**, **.ado**, and **.mata** suffix. Similarly, the documentations can be written in Markdown, html, or latex. However, for exporting Stata help files, only Markdown, smcl, or a combination of these two markup languages can be used to create **.sthlp** files. While html and latex can be used for creating package vignettes, it seems more plausible to write the documentation by combining smcl and Markdown, which on the one hand can greatly simplifies writing Stata help files and on the other, ensures that the help file can have the flexibility of the smcl language when it is needed.

## Dialog box



To further facilitate using **MarkDoc** in classrooms, a dialog box was written for Stata, which also shows the options and features of Stata. The dialog box is currently only supporting the *dynamic document* engine of **MarkDoc**.

To use the dialog box, type:

```
db markdoc
```

You can read more about the dialog box [here](#).

## Installation

MarkDoc package can **ONLY** be installed from Github and is not hosted on SSC any longer. The Stata github package can be used to install MarkDoc or any other Stata package hosted on Github. If you don't have the github command installed, type:

```
net install github, from("https://haghigh.github.io/github/")
```

Next, to install **MarkDoc** along with its dependencies from GitHub, type:

```
github install haghigh/markdoc
```

## Dependencies

The command above installs MarkDoc and its dependencies (other Stata packages), which are:

- **Weaver**
- **Statax**
- **md2smcl**

github installs the dependencies by executing dependency.do file in the Github repository after installing MarkDoc. This file includes the commands needed for installing the package dependencies.

## Required third-party software

MarkDoc also requires 3 third-party software, which are:

- **Pandoc**
- **pdfLaTeX**
- **wkhtmltopdf**

Having **Pandoc** installed on your system is rather necessary because many features of MarkDoc rely on **Pandoc**. The **pdfLaTeX** is optional, but required for generating PDF slides and typesetting documents written in LaTeX. If you write your document using Markdown and do not intend to generate dynamic PDF slides within Stata, you don't need to install **pdfLaTeX**. The **wkhtmltopdf** is only required for generating PDF documents from Markdown. Luckily, **Pandoc** and **wkhtmltopdf** can be installed automatically (see below).

## Automatic installation of third-party software

The `markdoc` command includes the `install` option, which downloads the **Pandoc** and **wkhtmltopdf** software automatically if they are not already installed or cannot be accessed by `markdoc`. The automatic installation was successfully tested on Mac OS X (10.9 and 10.10); 32-bit and 64-bit versions of Microsoft Windows (XP, 7, 8); Microsoft Windows 10 (64-bit); and Linux Ubuntu 14.04 (64-bit), Mint 17 (32-bit and 64-bit), and CentOS 7 (64-bit). However, manual installation is generally recommended because it ensures the installation of the latest version of the software.



`markdoc` installs the required software in a directory named *Weaver* inside the *plus directory*, where Stata expects to find user-written ado-files. The path to the `\ado\plus\` directory can be found using the `sysdir` command, which lists Stata's system directories. For Stata 13 and 14, the default Weaver directory paths are shown below based on the operating system.

```
Windows: C:\ado\plus\Weaver
Mac OS X: ~/Library/Application Support/Stata/ado/plus/Weaver
Linux: /home/username/ado/plus/Weaver
```

## Manual installation

Users can also download and install these software manually and define the paths to executable software in MarkDoc. For example, if you wish to create a dynamic document

The users can also permanently define the paths to these software using the `weave setup` command which memorizes the paths to the executable Pandoc and wkhtmltopdf permanently, even if MarkDoc package gets updated. The path information is stored in an *ado* file named *weaversetup.ado*, which can be found in `/PLUS/w/weaversetup.ado`.



Continue reading about `weave setup` in its separate page...

## `weave setup` command

The `weave setup` command is borrowed from `weaver` package and is used for defining the path to third-party software (e.g. Pandoc, wkhtmltopdf, pdfLaTeX) permanently. If you prefer to install the third-party software manually, this command comes very handy to avoid specifying the paths to these software anytime you call `markdoc`. To define permanent paths to these software, type:

```
weave setup
```

This command opens a file in your Stata where you can define file paths in it. The file has complete instructions written in it that you can follow. Inside the file, there are a few global macros that you can define the permanent file paths or change the behavior of `markdoc` package. To edit this file, insert the path to the executable file inside the quotations to define the global macro. Here is an example for defining the file path to pdfLaTeX on Windows (Note that the file path might be different on your machine):

```
global pathPdflatex "C:\Program Files\MiKTeX 2.9\miktex\bin\x64\pdflatex.exe"
```



If you wish to remove a path, make sure not to leave an empty space between the quotation marks. For example you can remove the example above by making the global macro NULL (i.e.

no string character is defined):

```
global pathPdflatex ""
```



The changes made in this file are permanent and do not get replaced with the next package update. But you can always access them using the `weave setup` command.

Here are the list of globals you can define in `weave setup`:

```
// -----  
// Path to executable third-party software, required by Weaver and MarkDoc  
// =====  
  
// wkhtmltopdf  
// -----  
global pathWkhtmltopdf "" //example: "C:\wkhtmltopdf\bin\wkhtmltopdf.exe"  
  
// pdfLaTeX  
// -----  
global pathPdflatex "" //example: "C:\Program Files\MiKTeX 2.9\miktex\bin\x64\pdflatex.exe"  
  
// MathJax  
// -----  
global pathMathJax "" //example: "C:\Users\haghish\Downloads\MathJax-master"  
  
// Pandoc (for MarkDoc)  
// -----  
global pathPandoc "" //example Win: "C:\Pandoc\pandoc.exe" //example Mac: "/usr/local/bin/pandoc"  
  
  
// -----  
// Settings for Weaver and MarkDoc  
// =====  
  
// markup language for MarkDoc  
// -----  
global markdocDefault "" //example: "markdown" or "html" or "latex"  
  
// Default paper size  
// -----  
global doc_paper "" //example: "a4" or "letter"
```

The second part of the file is used for defining the default markup language and the paper size (for weaver only). For example, if you intend to use LaTeX as your primary markup language, you can change the default settings of markdoc.

## GUI

MarkDoc includes a dialog box (GUI), which not only makes working with the package easier, but also shows all of the potential of the package for writing dynamic documents, dynamic presentation slides, and Stata help files and package vignette. To begin using the GUI, type:

```
. db markdoc
```

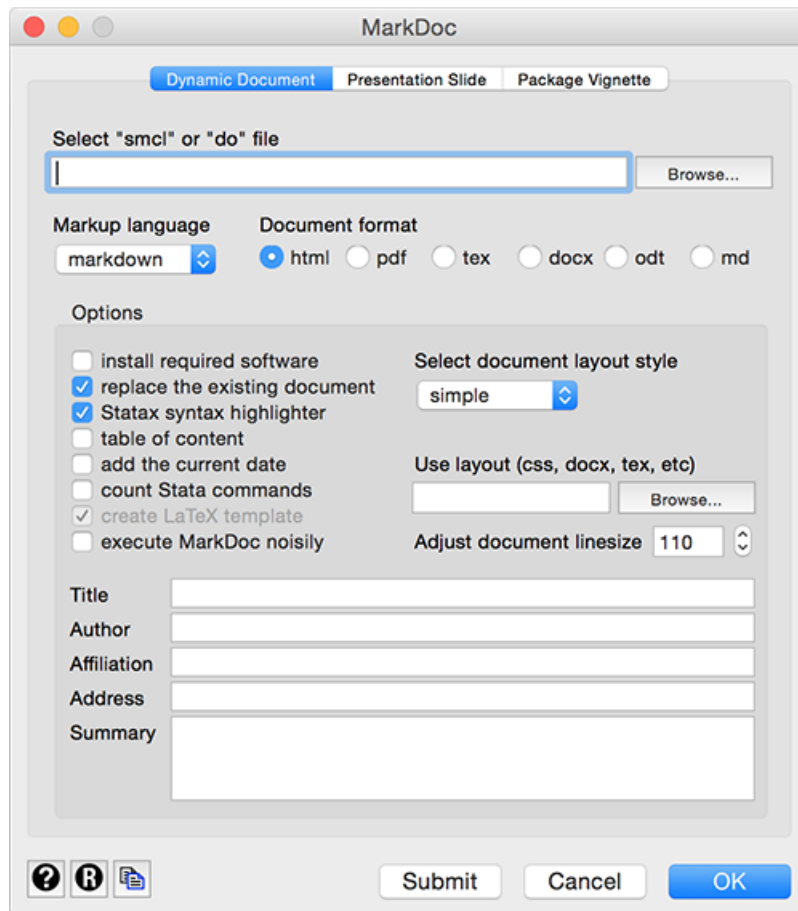
The command will open the GUI, which is divided into three main tabs, demonstrating the three separate engines that are included in MarkDoc package. The tabs are **Dynamic Document**, **Presentation S**lide, and **Package Vignette**.

In contrast to most of the Stata GUIs - which include a main tab and several tabs for options - the tabs in MarkDoc GUI are independent (because the engines are also independent). Therefore, if you wish to create a dynamic presentation slide or Stata help files, you should select the source file and options in the specified tab.

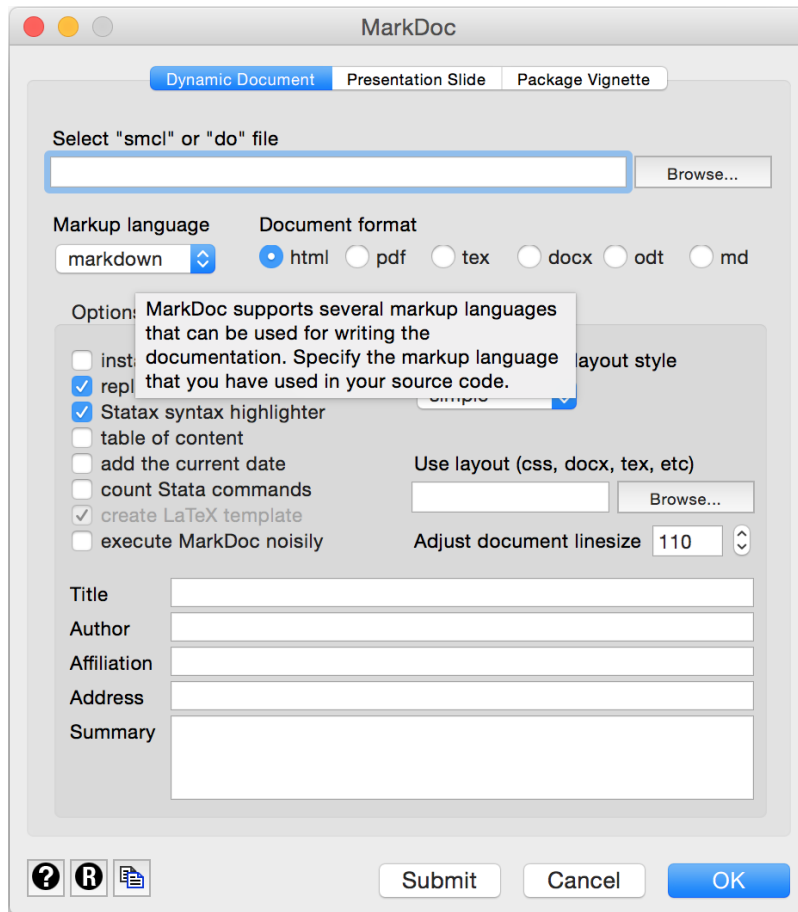
### Dynamic Document Tab

The first tab of the GUI is for creating dynamic documents from a source file, which can be a smcl log-file or a do-file. MarkDoc processes these two source files differently. A smcl log-file will simply

be converted to a dynamic document, whereas a do-file will be executed in a new workspace (i.e. the currently loaded data will not be available. You should include all of the code in the do-file) and the results are used to generate a dynamic document. When typing `db markdoc`, the following window appears:



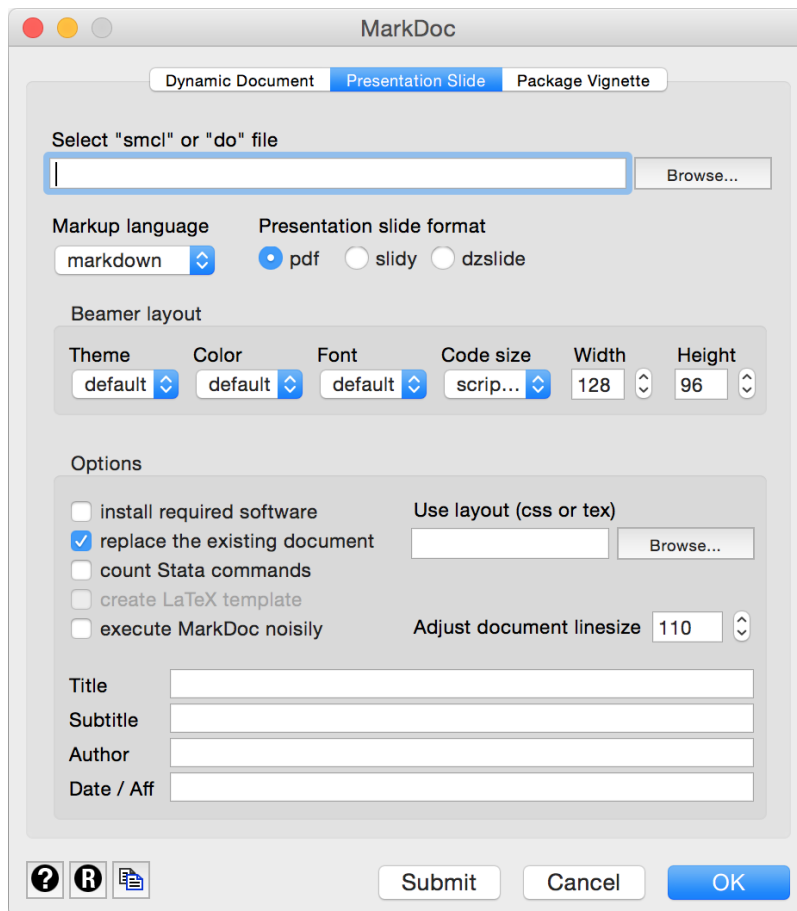
The default markup is **Markdown** but you can change that to HTML or LaTeX. The GUI also includes tips. By holding your mouse pointer on an object in the GUI, the tip will appear. For example, if you hold your pointer on the "Markup language", the following box will appear:



## Presentation Slide Tab

The second tab is for creating PDF or HTML-based presentation slides.





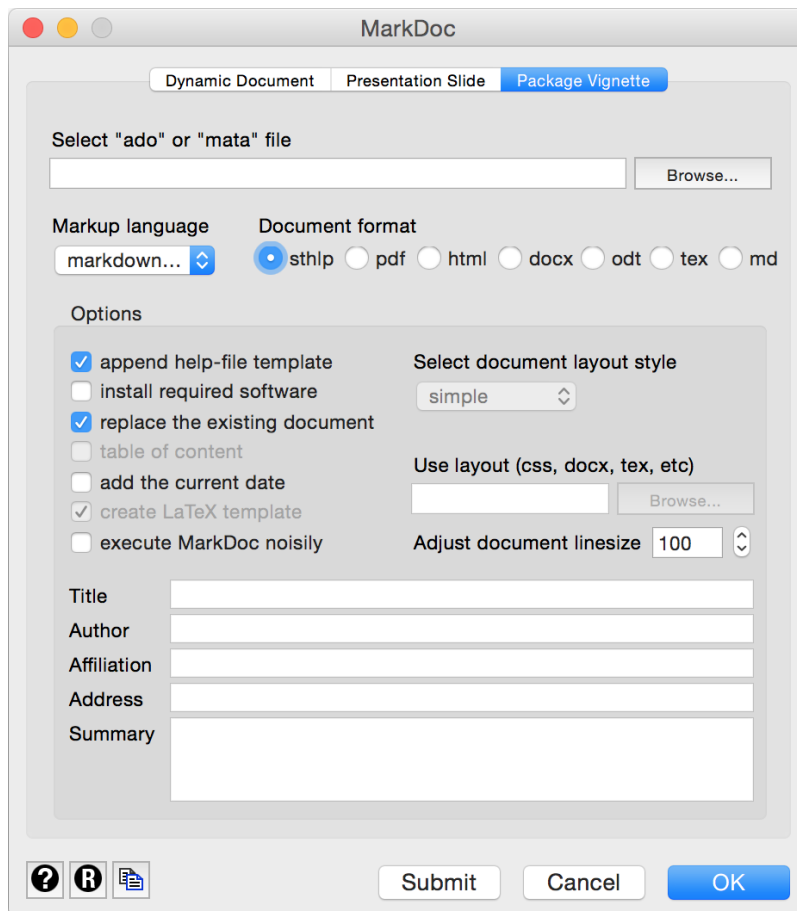
While generating HTML-based presentation slides can be attractive for some users, most users probably find the PDF presentation slides more useful and portable. The PDF presentation slides require LaTeX, however, they can be written in Markdown, which makes generating slides much easier. Moreover, MarkDoc includes several options for customizing the PDF slides. All of the Beamers' themes, colors, and fonts, as well as options for specifying the font size of Stata output and also the width and height of the slides can be customized.

**NOTE** that generating dynamic slides is not yet documented in this manual, nor it is mentioned in the journal article. However the torture tests include a directory called Beamer that has several examples for generating dynamic slides with Markdown and LaTeX. Basically, creating slides is not really different for generating documents (despite the options and the engine), however, you should separate the slides from one another. When writing with Markdown, you can separate the slides by:

1. Writing heading 1
2. Using a line --- which breaks the slide
3. Creating a new documentation chunk using **"/" and "/"** signs

## Package Vignette

The package vignette tab uses ado-file or mata-file and extracts the documentation written in the file to generate a dynamic Stata help file (sthlp) or a package vignette, in any of the supported formats. The documentation can be written in smcl or a combination of smcl and Markdown. The package vignette tab is shown below:



It is recommended to write the documentation in combination of Markdown and smcl, which significantly simplifies writing and editing package documentation. This will ensure that your vignette documents will look as good as your Stata files.

## Dynaic Text

So far the only way for writing dynamic text - that is text that includes macros and scalars for interpreting the results has been using the `txt` command. The `txt` command is very enabling, it can be used inside loops, programs, etc, to display dynamic text in the dynamic document.

But I have been trying to make working with **MarkDoc** as simple as possible. Recently I developed a new marker, named `<!*!>` that can be used inside the documentation to view local and global macros, numeric and string scalars, as well as scalars from matrices and data sets. The syntax is fairly simple, instead of the `*`, put the object that you wish to evaluate. The object can be:

Object	Description
<code>&lt;!scalar!&gt;</code>	Numeric or String scalar
<code>&lt;!matrix[r,c]!&gt;</code>	Numeric scalar from a matrix
<code>&lt;!variable[n]!&gt;</code>	Nth observation of a variable
<code>&lt;!`local`!&gt;</code>	Numeric local macro
<code>&lt;!\$global!&gt;</code>	Numeric global macro
<code>&lt;!``local``!&gt;</code>	String local macro
<code>&lt;!"\$global"!&gt;</code>	String global macro

## Examples

1. Assume you have a local macro named `a` and you want to evaluate it inside your text. You'd write:

```
local a = 1
scalar b = 2
matrix define A = (20,30\40,50)
```

```
/**
This is heading <!`a`!>
=====
```

The values of a matrix can be displayed within the text. For example, you can write `<!A[1,1]!>` which shows the scalar of the first row and first column of the matrix in your documentation. This feature makes writing dynamic text much more convenient compared to the previous procedure.

```
This is heading <!b!>
-----
```

REMEMBER, that this procedure only works if you execute a do-file with `markdoc`, that is, using the ``markdoc filename.do, export(format)`` syntax.  
\*\*\*/

if you save this script in a Stata do-file -- say `example.do` -- and execute it with `markdoc`:

```
markdoc example.do, export(html) replace
```

you see that the local macro `a` is updated to 1 and scalar `b` is updated to 2. Note the difference between the way the `a` and `b` are called inside the `<!*!>` markers.

This procedure is much easier than using the `txt` command and more natural to the way `Markdoc` takes care of writing the documentation. However, it only works if you execute a do-file with `markdoc`.

## Additional Commands

The main command of the package is `markdoc` that produces the dynamic document from `smcl` log-file or `do` file or Stata package documentation from `ado` or `mata` files. However, the package includes other commands and it also borrows additional commands from `Weaver` package.

In general, you can create a dynamic document by just using a markup language (Markdown, LaTeX, HTML, `smcl`) and the `markdoc` command. However the additional commands can make working with **MarkDoc** package much more convenient.

### List of additional commands

- `img`
- `tbl`
- `txt`
- `pandoc`
- `wkhtmltopdf`

### `img` Commands

#### *Description*

All markup languages supported by **MarkDoc** can include images from the disk or internet in the dynamic document. For example, if you are writing with Markdown:

```
![image description](path/to/the/image)
```

if you are writing with LaTeX:

```
\includegraphics{path/to/the/image}
```

and if you are writing with HTML:

```

```

And to add an image dynamically, you can use the `txt` command to write the markup syntax:

```
txt ![image description](path/to/the/image)
```

Using a markup language for importing the an image requires two steps:

1. saving a graph from Stata to the disk
2. including the graph to the dynamic document

This procedure can be further simplified, using the `img` command which can automatically capture the current graph from Stata and include it in the dynamic document. This command is borrowed from the Weaver package.

## Features

To make **MarkDoc** a suitable literate programming package for teaching statistics, even in introductory courses, the `img` command was written to eliminate the need of learning a markup language for importing and styling images in the dynamic document. The command can:

1. Automatically capture the current graph from Stata and include it in the dynamic document
2. Include a figure from the disk/internet in the dynamic document
3. Resize the width and the height of the image in the dynamic document
4. Align the image to the left (default) or center of the document
5. Add a graph description

## Syntax

Import graphical files in the dynamic document

```
img [using filename] [, markup(str) title(str) width(int) height(int) left center ]
```

Automatically include the current graph from Stata in the dynamic document

```
img [, markup(str) title(str) width(int) height(int) left center ]
```

## Options

Options	Description
markup(str)	specify the markup language that should be added to the smcl log
title(str)	specify a header string (title) for the figure
width(int)	define the width of the figure
height(int)	define the height of the figure
left	aligns the figure to the left-side of the dynamic document (default)
center	aligns the figure to the center of the dynamic document

## Examples

The `img` command prints the required markup language in the smcl log to import a figure in the dynamic document. If the `filename` is not specified, `img` automatically captures the current graph from Stata and includes it in the dynamic document. For example:

```
. sysuse auto
. histogram price
. img
>
```

In this example, `img` has stored the current graph in a directory called **Weaver-figure** and then prints a markup syntax for importing the image in the dynamic document. If you are writing the

documentation using LaTeX, you have to change the markup language using the `markup()` option:

```
. img, markup(latex)
>\begin{figure}[h]
>\centering
>\includegraphics[width=350px, height=250px]{Weaver-figure/f
> igure_3.png}
>\end{figure}
```

So instead of writing this LaTeX code for including the image, you just have used a simple command. As you see by default, the image is imported with `width=350` and `height=250` pixels. You can also resize the figure using the `width` and `height` options. This time, I use the HTML markup as example:

```
. img, markup(html) width(400) height(300)
>
```

Unfortunately, markdown is a very simplified markup language and it does not allow you to resize the image. Therefore, if you want an image with a particular size you should use the second approach, namely, saving exporting your figure with a particular size and then importing it in the dynamic document. However, **this is only necessary in the html** format. So if you are intending to produce **pdf, slide, tex, docx, or odt** from a markdown file, you don't have to worry about an oversized file. **MarkDoc** takes care of that.

Finally, if you want to add a *description* to the graph or center the graph in the document, you can use the `title()` and `center` options respectively. These options will add the required markup to style your figure respectively. See the Examples for further demonstrations.

## Remarks

The `img` command adds the required markup syntax for importing a graph to the log. The default markup language is Markdown. Therefore, if you are using a different markup language such as HTML or LaTeX, you should specify that using the `markup()` option.

## tbl Commands

### Description

Similar to the `txt` and `img` commands, the `tbl` command is also borrowed from the Weaver package and was updated to support **MarkDoc** package. Therefore, you have to make sure the Weaver log is closed. To do so, type `weave query` to check the status of the Weaver log. The `tbl` command works similar to the

### Features

- `tbl` simplifies writing and styling dynamic tables
- It can also align the content of each column to the left, center, or right
- It creates a table somehow similar to the way a matrix is defined in Stata

### Syntax

The `tbl` command creates a dynamic table in the specified markup language. The default markup language is Markdown. The syntax of the command is:

```
tbl (*[,*...] [\ *[,*...] [\ [...]]]) [, markup(str) title(str) width(int) height(int) center left ]
```

where the `*` represents a display directive which is:

```
"double-quoted string"
`"compound double-quoted string"
[%fmt] [=]exp
,
{l}
{c}
{r}
```

## Options

The `tbl` command takes fairly simple options. When HTML or LaTeX markup languages are used for writing the documentation, the `markup(str)` option must be specified. Otherwise, the command will append Markdown syntax to the log.

As noted, Markdown is a very minimalistic markup language with limited styling possibilities. Therefore, there should be no surprise that the `width(int)`, `height(int)`, and `center` options are only available when writing in HTML or LaTeX.

Options	Description
<code>markup(str)</code>	specifies the markup language that is used for documentation
<code>title(str)</code>	displays the table description
<code>width(int)</code>	specifies the width of the table in HTML and LaTeX
<code>height(int)</code>	specifies the height of the table in HTML and LaTeX
<code>center</code>	aligns the table to the center of the document in HTML and LaTeX
<code>left</code>	aligns the table to the left of the document

## Display directives

Display Directive	Description
"double-quoted string"	displays the string without the quotes
<code>``compound double-quoted string``</code>	display the string without the outer quotes; allows embedded quotes
<code>[%fmt] [=] exp</code>	allows results to be formatted
<code>,</code>	separates the directives of each column of the table
<code>{l}</code>	creates a left-aligned column
<code>{c}</code>	creates a center-aligned column
<code>{r}</code>	creates a right-aligned column

## Examples

creating a simple 2x3 table with string and numbers

```
. tbl ("Column 1", "Column 2", "Column 3" \ 10, 100, 1000 )
```

Column 1	Column 2	Column 3
10	100	1000

creating a table that includes scalars and aligns the columns to left, center, and right respectively

```
. tbl ({l}"Left", {c}"Centered", {r}"Right" \ c(os), c(machine_type), c(username))
```

Left	Centered	Right
MacOSX	Macintosh (Intel 64-bit)	haghish

## Remarks

Note that the `tbl` command parses the rows using the backslash symbol. Therefore, to include LaTeX notations in a dynamic table that begin with a backslash such as `\beta` or `95%`, double backslash should be used to avoid conflict with the parsing syntax (e.g. `\\beta` and `95\\%`). Here are a couple of examples:

```
. tbl ("$\beta$", "$95\\%$ Confidence Interval" \ "values...", "values...")
```

$\beta$	95% Confidence Interval
values...	values..

```
. tbl ("$\beta$", "$\epsilon$" \ "$\sum$", "$\prod$")
```

$$\frac{\beta}{\Sigma} \quad \frac{\epsilon}{\Pi}$$

## txt Commands

**Note:** markdoc has a new method for writing dynamic text which is much simpler than using the `txt` command. You can read more about the new feature on this page.

### Description

**MarkDoc** has a very convenient way for writing text in the dynamic document, using a special comment signs that are distinguished from regular comment. However, using comments for documentation comes with a limit, namely, dynamic text - text that includes scalars and macros - cannot be displayed in the document. The `txt` command provides a solution to this problem by displaying values of scalar expressions or macros with text, allowing the users to write dynamic text. For example:

```
. sysuse auto
. summarize price
. txt "the mean of Price variable is " r(mean)
> the mean of Price variable is 6165.26
```

You might wonder what is the benefit of writing dynamic text? The main benefit is reducing human errors when returned values are meant to be used in the documentation. For example, you want to mention the mean of a variable in a text paragraph. The mean, however, can change if you drop an observation. Writing dynamic text will ensure that anytime there is a change in the data, the values in the text will be automatically updated. There is another advantage for using the `txt` command. Namely, you can produce dynamic text from a loop or a program. For example, imagine you are looping over many `varlists` and you wish to include the results in separate sections. You could:

```
foreach lname of varlist var1 var2 ... {
  txt "### Analyzing the `lname` Variable
  ...
}
```

The `txt` command belongs to Weaver package, but it was updated to support **MarkDoc**. The reason was to have a single command for writing dynamic text in both packages, instead of introducing another command. You can only use the `txt` command in **MarkDoc**, when Weaver is not in use, i.e. your "Weaver log" is off. To check the status of the Weaver log type:

```
. weave query
```

The `txt` command is to some extent similar to `display` command in Stata. For example, it can be used to carry out a mathematical calculation by typing:

```
. txt 1+1
> 2
```

### Features

The `txt` command can:

- Write dynamic text, i.e. text that can interpret scalar and macros.

- It can be used to use the values returned from Stata commands in the interpretation to minimize human errors and make them traceable
- It can also be used to write mono-space font in the document
- It can style text using the same markup language that the document is written with (Markdown, LaTeX, HTML)
- It supports several display directives, similar to the `display` command in Stata
- It can be included inside loops or programs to produce dynamic text

## Syntax

The `txt` command prints dynamic text on the smcl log

```
txt [code] [display_directive [display_directive [...]]]
```

where the `display_directive` can be:

```
"double-quoted string"
`"compound double-quoted string"
[%fmt] [=]exp
_skip(#)
_column(#)
_newline[#]
_dup(#)
,
,,
```

The code argument changes the behavior of the `txt` command to display the text as a code block, using a mono-space font (see below).

## Display directives

The supported `display_directives` are used in do-files and programs to produce formatted output. The directives are:

Display directive	Description
"double-quoted string"	displays the string without the quotes
`"compound double-quoted string"	display the string without the outer quotes; allows embedded quotes
<code>[%fmt] [=] exp</code>	allows results to be formatted
<code>_skip(#)</code>	skips # columns
<code>_column(#)</code>	skips to the #th column
<code>_newline</code>	goes to a new line
<code>_newline(#)</code>	skips # lines
<code>_dup(#)</code>	repeats the next directive # times
,	displays one blank between two directives
,,	places no blanks between two directives

## Styling dynamic text

By default, the `txt` command writes a text paragraph. However, the text can be displayed differently in the dynamic document using the same markup language that is used in the document. For example, if you are writing your document in Markdown, you can write a "Heading 3" dynamic text as follows:

```
. txt "### some text ... "
```

or if you are writing your documentation in LaTeX:

```
. txt "\subsubsection{some text ...} "
```

## Examples



You can use the `txt` command for interpreting your results. This works very similar to using the `display` command.

```
. sysuse auto
. summarize price
. txt "the mean of Price variable is " r(mean) " and SD is " %9.3f r(sd)
> the mean of Price variable is 6165.26 and SD is 2949.496
```

Using the `display_directives` reveals the power of the `txt` command for producing dynamic text with a particular structure. For example, you can use the `_newline` or simply `_n` to begin a new line. However, if you are writing in Markdown, HTML, and LaTeX, breaking the line is not enough to make sure the output also will be in multiple lines, although the `txt` command will break the lines in the `smcl` log anyway. In the example below, which assumes writing with markdown, I use double space characters at the end of each line to break the lines in the output. end of each line to ensure the output generated from

```
txt "this is the first line " _n ///
    "and this is the second line "
```

I can also add indents to the text using the `_column()` directive:

```
txt _column(10) "Hello World"
> Hello World
```

Or skip a number of characters:

```
. txt _skip(10) "Hello World"
> Hello World
```

## Remarks

In contrast to the `display` command that prints the scalar unformatted, the `txt` command uses the default `%10.2f` format for displaying the scalar. This feature helps the users avoid specifying the format for every scalar, due to popularity of this format.

```
. scalar num = 10.123
. txt "The value of the scalar is " num
> The value of the scalar is 10.12
```

However, specifying the format expression can overrule the default format. For example, to display the value of the scalar with only 1 decimal place I can change the default format of the `txt` command:

```
. txt "The value of the scalar is " %5.1f num
> The value of the scalar is 10.1
```

The example above will print the scalar with only 1 decimal number. This feature only supports scalar interpretation and does not affect the macro contents.

## pandoc Commands

**MarkDoc** relies on Pandoc software for converting the Markdown document to several formats. However, many of the functionalities of this software are not included in **MarkDoc**, simply because there is no obvious use for them for the majority of the users. However, since the software is needed, the `pandoc` command was added to allow the users use Pandoc interactively within Stata. The only advantage of this command is that it simplifies calling Pandoc and eliminates the need of memorizing Pandoc path. See Pandoc Examples to get an idea of what you can do with this command.

There are plenty of occasions that the `pandoc` command can be very helpful. For example, imagine you have used a user-written command to export a LaTeX or HTML table but you wish to include a Markdown table in your document. You can use the `pandoc` command to automatically convert the file

## Syntax

The syntax of the command is as follows:

```
pandoc command
```

## Examples

executing Pandoc command

```
. pandoc filename -o filename
```

adding more Pandoc arguments

```
. pandoc -s -S filename -o filename
```

## Remarks

Users can permanently define the path to Pandoc software using the **weave setup** command if they have not installed Pandoc automatically, using the **install** option.

## wkhtmltopdf Commands

### Description

Pandoc software which is the main file convertor engine of **MarkDoc** package requires a LaTeX distribution in order to create a PDF document. In other words, if users wish to write their documentation in Markdown and produce a PDF file using Pandoc, they must also have LaTeX on their machine. However, **MarkDoc** allows producing PDF documents from Markdown and HTML documents, using another third-party software called wkhtmltopdf, which can be automatically installed within Stata using the **install** option.

There are several user-written packages that produce HTML files from Stata. These packages can get benefit from wkhtmltopdf to convert their HTML files to PDF. Therefore, this command was written to simply allow the users to convert their HTML files to PDF.

### Syntax

wkhtmltopdf converts HTML files to PDF and the general syntax of the command is as follows:

```
wkhtmltopdf [arguments] filename.html filename.pdf
```

The command can take many arguments which are documented here. Using the correct arguments is crucial, especially if your HTML document includes several JavaScript programs.

### Examples

convert html file to pdf

```
. wkhtmltopdf myfile.html myfile.pdf
```

### Remarks

Unless you have installed the required third-party software using the **install** option, you should use the **weave setup** command to define the permanent path to the wkhtmltopdf software.

## Tutorials

See:

- Examples
- Mathematical notations
- Writing with Markdown

- Writing with LaTeX

## Markdown tutorial

Although in addition to Markdown, **MarkDoc** supports HTML and LaTeX markup languages, using Markdown is in general recommended since it notably keeps the do-file easy to read. HTML and LaTeX syntax can reduce the readability of the script file, once it is added to a do-file that already include programming code. However, HTML and LaTeX provide much more control over the document whereas Markdown is a minimalistic language. Yet, the most notable benefit of writing the documentation with Markdown is that **MarkDoc** can typeset a document that is written with Markdown to several formats such as **pdf**, **html**, **latex**, **slide**, **docx**, **odt**, and **epub**.

In this section, I review the syntax of Markdown and provide several examples, demonstrating how Markdown can be used within Stata DO-file editor to generate a document.

## Headers

Without any syntax, the text written with Markdown appears as text paragraphs. However, writing a header is as simple as adding hash "#" sign before a text line, where a single hash represents header 1 and 6 hashes represent header 6:

```
# H1
## H2
### H3
#### H4
##### H5
##### H6
```

This is a text paragraph, which requires no syntax.

Markdown also provides an alternative syntax for creating header 1 and 2, which further improves the readability of the script file. The alternative syntax makes scanning through the documentation greatly simpler, since the headers will become more distinctive in the script file.


```
Header 1
=====
```

```
Header 2
-----
```

## Creating tables

In general, the tables that you include in the dynamic document can be divided in two groups:

1. static tables, where the content of the table does not include scalars or macros
2. dynamic tables, where the table includes scalar or macro values.

 MarkDoc has a specific command for generating tables called **tbl**. However, there are other possibilities for generating a table in MarkDoc which are covered in this document. Most of the users will find the **tbl** command much handier!

### Static tables

Any of the supported markup languages (Markdown, HTML, LaTeX) can be used to create a table. Here are simple examples of Markdown, HTML, and LaTeX tables that generate a 3 by 3 table.

**Table 1.** *preview of the example table written in Markdown*

<b>Animals</b>	<b>Sports</b>	<b>Fruits</b>
Cat	Soccer	Apple
Dog	Basketball	Orange

All of these examples will render a table in the dynamic document which looks like the table above. However, it would be a good idea to compare these markup languages to one another.

### Example 1. *Markdown table*

```
/**
| __Animals__ | __Sports__ | __Fruits__ |
|-----|-----|-----|
| Cat        | Soccer     | Apple      |
| Dog        | Basketball | Orange     |
***/
```

### Example 2. *LaTeX table*

```
/**
\begin{table}[]
\centering
\caption{My caption}
\label{my-label}
\begin{tabular}{|l|l|l|}
\hline
\textbf{Animals} & \textbf{Sports} & \textbf{Fruits} \\ \hline
Cat & Soccer & Apple \\ \hline
Dog & Basketball & Orange \\ \hline
\end{tabular}
\end{table}
***/
```

### Example 3. *HTML table*

```
/**
<table class="tg">
<tr>
<th class="tg-yw41"><b>Animals</b></th>
<th class="tg-yw41"><b>Sports</b></th>
<th class="tg-yw41"><b>Fruits</b></th>
</tr>
<tr>
<td class="tg-yw41">Cat</td>
<td class="tg-yw41">Soccer</td>
<td class="tg-yw41">Apple</td>
</tr>
<tr>
<td class="tg-yw41">Dog</td>
<td class="tg-yw41">Basketball</td>
<td class="tg-yw41">Orange</td>
</tr>
</table>
***/
```

As evident from the examples, the Markdown code is much more human readable compared to HTML and LaTeX. Using an easy-to-read markup language such as Markdown for writing documentation improves the readability of your script files.

## Dynamic tables

The problem with the examples above is that the content of the tables is static, i.e. it does not change. Often, when you are presenting the results of your data analysis and interpreting them, you need to create a table using the results returned from the analysis.

One solution to this problem is to use the `txt` command for adding the code to the `smcl` log file. For example, let's assume that you are writing your documentation using Markdown, and you want to create a table similar to the previous 3 by 3 table, however, you want to obtain the values from local macros.

### Example 4. Dynamic table problem

```
local 11 Cat
local 12 Dog
local 13 Soccer
local 14 Basketball
local 15 Apple
local 16 Orange
```

```
txt
" | __Animals__ | __Sports__ | __Fruits__ | " _n ///
" |-----|-----|-----| " _n ///
" | `11' | `13' | `15' | " _n ///
" | `12' | `14' | `16' | " _n
```

This example will be rendered similar to the **Table 1**. But creating such a table requires a lot of work. **MarkDoc** provides a simpler method for creating a dynamic table, which is using the `tbl` command. This command is documented in a separate section.

## Package structure

For those who wish to contribute to **MarkDoc**, this document can behave as a quick reference to different source files included in the package.

The main engine of MarkDoc is **markdoc.ado**. This program calls many other programs as well to keep the syntax coherent. Here, these commands are explained.

### Dynamic document and dynamic slides

The programs involved with making dynamic documents and slides are the same. **markdoc.ado** takes care of translating SMCL log files and generating these documents. However, if a do-file is specified as a source, markdoc will call rundoc program, defined in **rundoc.ado**. rundoc is called for executing Stata commands in a particular way to allow *passive dynamic text*.

### Dynamic Stata help files (sthlp)

For producing Stata help files and SMCL files, you should read **sthlp.ado**, **markup.ado**, and **md2smcl.ado**. Here is a description of other script files:

### Other files

- *markdocversion.ado* checks the new updates available for MarkDoc, if there is any
- *markdoccheck.ado* checks the required third-party software on the machine and if the **install** option is specified, it install the required software
- *markdocpandoc.ado* installs Pandoc automatically, if the **install** option is specified and the software is not found
- *markdocwkhtmltopdf.ado* installs wkhtmltopdf automatically, if the **install** option is specified and the software is not found