Title

stata.com

lmbuild — Easily create function library

Description Syntax Options Remarks and examples Also see

Description

`lmbuild` builds Mata function libraries just as `mata mlib` does. Even though `lmbuild` virtually requires the creation of a do-file, it is easier to use and is therefore a better alternative than `mata mlib`.

Why two commands to do the same thing? `mata mlib` existed first to create Mata libraries, but it was complicated to use. `lmbuild` was added later and makes it easier to create those libraries.

You type `lmbuild` after Stata's dot prompt, not Mata's colon prompt.

Syntax

. `lmbuild` *libname* [, [`new` | `replace` | `add`] `dir(`*dirname*`)` `size(#)`]

libname is the name of a Mata library, such as `lexample.mlib`. Library names must start with `l` and end in `.mlib`. Library names may be specified with or without the extension.

`lmbuild` is a Stata command that you use from Stata's dot prompt, not from Mata's colon prompt.

Options

`new`, `replace`, and `add` are alternatives and indicate whether the library file is new, should be replaced, or should be added to.

`new` is the default. It specifies that *libname* does not already exist and is to be created.

`replace` specifies that *libname* might already exist, and if it does, the library is to be replaced.

`add` specifies that *libname* already exists and functions are to be added to the existing library. We advise you not to use this option except in carefully constructed do-files that create the library from start to finish. If you choose to use `add` interactively, you run the risk of creating irreproducible libraries.

`dir(`*dirname*`)` specifies the directory in which *libname* exists or is to be created. *dirname* may be one of the following:

`dir(PERSONAL)` is the default. Libraries will be created or updated in your personal directory. You can type the `sysdir` command to find out where your personal directory is. Libraries in your personal directory will be automatically found by Mata. If you do not already have a personal directory, `lmbuild` will create one for you.

`dir(SITE)` specifies that the library be created or updated in the site directory. This directory is shared across Stata users at your location. You can type the `sysdir` command to find out where your site directory is. You will probably need administrator privileges to write to this directory.

`dir(.)` specifies that the library exists or is to be created in the current directory. The only reasons to specify `dir(.)` are that you intend to copy the library to another directory later, to send the library to someone, or to include the library in a package for distribution to other users.

`dir(`*directory*`)` specifies that the library exists or is to be created in *directory*. Specifying this option is not recommended because Mata will not find such libraries unless they are added to Mata's search path.

`size(#)` specifies the maximum number of functions to be allowed in the library. Libraries allow up to 1,024 functions by default. # may be a number from 2 to 2,048. `size()` may be specified only with new libraries or libraries that are being replaced.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Background

Mata functions that you write and then store in libraries are placed on the same footing as Mata's built-in functions. They can be used in code that you write without being preloaded, whether that code is in do-files, ado-files, or Mata.

You can have as many Mata libraries as you wish. Each library may contain up to 1,024 functions, or up to 2,048 if you specify `lmbuild`'s `size()` option.

Libraries store the compiled version of functions, not the source code. We recommend that you place your source code in do-files that look like the following:

begin `hello.mata`

```
version #
mata:
void hello()
{
        printf("hello world\n")
}
end
```

end `hello.mata`

You can load the function into Mata by typing `do hello.mata` at the Stata prompt. You can test the function. When you want to place the function in a library, you type

```
. clear all
. do hello.mata
. lmbuild lmylib
```

`lmbuild lmylib` creates a Mata library named `lmylib.mlib` containing all the Mata functions loaded into memory since the last `clear all`. Thus, this library will contain just one function, namely, `hello()`.

If you had other functions stored in other .mata files, you could load each of them and then create the library:

```
. clear all
. do hello.mata
. do havelunch.mata
. do goodbye.mata
. lmbuild lmylib
```

The Mata library would contain three functions, assuming the three .mata files defined three Mata functions. The three functions defined are probably named hello(), havelunch(), and goodbye(), but it is not required that the function name match the filename. Each .mata file, in fact, can define as many functions as you wish. If the file hello.mata contained

begin hello.mata

```
version #
mata:
void hello()
{
        printf("hello world\n")
}
void goodbye()
{
        printf("good-bye world\n")
}
end
```

end hello.mata

and you typed

```
. clear all
. do hello.mata
. lmbuild lmylib
```

then there would be two functions in the library: hello() and goodbye().

Usually, you will have multiple .mata files and define multiple functions in some of them. Each file will define a function and its subroutines. Sometimes, however, you will define related functions in the same file. Regardless of the situation, libraries should not be built interactively because someday code will change and you will need to rebuild the library. The right way to proceed is to make a do-file that will make it easy for you to create and re-create the library:

begin make_lmylib.do

```
* version number intentionally omitted
clear all
do hello.mata
do bigfcn.mata
do utilityfunctions.mata
.
.
lmbuild lmylib, replace
```

end make_lmylib.do

With this do-file written, all we have to type to create the library for the first time is

```
. do make_lmylib
```

All we have to type to re-create the library later is

```
. do make_lmylib
```

Why would we need to re-create the library? One reason would be that we need to re-create the library after fixing a bug in `bigfcn.mata`.

Notice the comment at the top of `make_lmylib.do`,

```
* version number intentionally omitted
```

and notice that we included version numbers in each of the `.mata` files. That is how you handle version control with libraries.

Version control

The version number appearing in each `.mata` file is the version number under which the code was written. If `hello.mata` was written back in the days of Stata 11, it would read

begin `hello.mata`

```
version 11
mata:
void hello()
{
        printf("hello world\n")
}
end
```

end `hello.mata`

The first line of the file sets the version of Mata in which the code is written. It is called the compile-time version number. Specifying the compile-time version number ensures that the code is backdated to retain its original behavior should the meaning or requirements of some aspect of Mata's programming language or some feature of Mata's compiler change.

In file `make_mylib.do`, there is nothing we want backdated. The entire purpose of `make_mylib.do` is to make a modern Mata library, even as new versions of Stata are released. Thus, the version number is intentionally omitted.

One more type of version number where Mata is concerned is called the run-time version number. It is not directly relevant for building libraries, but it is relevant when you want to change the way functions work for different versions of Stata just as we do at StataCorp with the functions we write. We do not preserve bugs, of course, but we do add features, and sometimes new features get in the way of old ones. If we did not write our code in a certain way, old do-files would not continue to work until the user had updated them to new syntax and calling sequences. We write code in such a way that users do not have to do that.

Let's consider a case where you wrote `bigfcn()` back in the days of Stata 13. In Stata 18, you rewrote the function, changed what the arguments did, and increased the number of arguments from one to two. Your original code looked like this:

begin `bigfcn.mata`

```
version 13
mata:
real matrix bigfcn(real matrix A)
{
        ...
}
end
```

end `bigfcn.mata`

Here is how your updated code might look if you wanted to preserve old behavior and allow new features:

begin `bigfcn.mata`

```
version 18
// ------------------------------------- version 18 starts here
mata:
real matrix bigfcn(real matrix A, |real scalar style)
{
        if (callersversion()>=18) return(bigfcn_new(A, style))
        else                      return(bigfcn_old(A))
}
real matrix bigfcn_new(real matrix A, real scalar style)
{
        ... new code ...
}
end
// ---------------------------------------------- and ends here
version 13
// ------------------------------------- version 13 starts here
mata:
real matrix bigfcn_old(real matrix A)
{
        ... old code ...
}
end
// ---------------------------------------------- and ends here
```

end `bigfcn.mata`

Notice that we specified `version 18` for part of the file and `version 13` for the other part. That is how we made sure that the old code compiled as intended, just in case the compiler changed.

In the new `bigfcn()` function, we make the second argument optional by specifying `|real scalar style`. Notice the vertical bar and see [M-2] **optargs**.

Finally, notice that we used Mata built-in function `callersversion()` to call the new or old code as appropriate.

With bigfcn() defined this way, old do-files continue to work, such as

begin oldfile.do

```
version 13
...
...   bigfcn(X) ...
...
```

end oldfile.do

Do-files specifying version 14 through 17 would continue to work, too.

A modern do-file set to version 18 or later, however, would use the improved bigfcn() and its two arguments:

begin modernfile.do

```
version 18
...
...   bigfcn(X, 1) ...
...
```

end modernfile.do

The version number specified in the do-file is known as its run-time setting.

Also see

[M-3] **mata mlib** — Create function library

[M-3] **Intro** — Commands for controlling Mata