

## mata mlib — Create function library

[Description](#)[Syntax](#)[Options](#)[Remarks and examples](#)[Also see](#)

## Description

`mata mlib` creates, adds to, and causes Mata to index `.mlib` files, which are libraries containing the object-code functions.

`mata mlib create` creates a new, empty library.

`mata mlib add` adds new members to a library.

`mata mlib index` causes Mata to build a new list of libraries to be searched.

`mata mlib query` lists the libraries to be searched.

A library may contain up to 1,024 functions by default.

## Syntax

```
: mata mlib create libname [ , dir(path) replace size(#) ]
```

```
: mata mlib add libname fcnlist() [ , dir(path) complete ]
```

```
: mata mlib index
```

```
: mata mlib query
```

where *fcnlist()* is a *namelist* containing only function names, such as

*fcnlist()* examples

---

```
myfunc()
myfunc() myotherfunc() foo()
f*() g*()
*()
```

---

see [\[M-3\] namelists](#)

and where *libname* is the name of a library. You must start *libname* with the letter `l` and do not add the `.mlib` suffix as it will be added for you. Examples of *libnames* include

<i>libname</i>	Corresponding filename
<code>lmath</code>	<code>lmath.mlib</code>
<code>lmoremath</code>	<code>lmoremath.mlib</code>
<code>lnjc</code>	<code>lnjc.mlib</code>

Also *libnames* that begin with the letters `lmata`, such as `lmatabase`, are reserved for the names of official libraries.

This command is for use in Mata mode following Mata's colon prompt. To use this command from Stata's dot prompt, type

```
. mata: mata mlib ...
```

## Options

`dir(path)` specifies the directory (folder) into which the file should be written. `dir(.)` is the default, meaning that if `dir()` is not specified, the file is written into the current (working) directory. *path* may be a directory name or may be the `sysdir` shorthand `STATA`, `BASE`, `SITE`, `PLUS`, `PERSONAL`, or `OLDPLACE`; see [P] [sysdir](#). `dir(PERSONAL)` is recommended.

`complete` is for use when saving class definitions. It specifies that the definition be saved only if it is complete; otherwise, an error message is to be issued. See [M-2] [class](#).

`replace` specifies that the file may be replaced if it already exists.

`size(#)`, used with `mlib create`, specifies the maximum number of members the newly created library will be able to contain,  $2 \leq \# \leq 2048$ . The default is `size(1024)`.

## Remarks and examples

[stata.com](http://www.stata.com)

Remarks are presented under the following headings:

- [Background](#)
- [Outline of the procedure for dealing with libraries](#)
- [Creating a .mlib library](#)
- [Adding members to a .mlib library](#)
- [Listing the contents of a library](#)
- [Making it so Mata knows to search your libraries](#)
- [Advice on organizing your source code](#)

Also see [M-1] [how](#) for an explanation of object code.

## Background

`.mlib` files contain the object code for one or more functions. Functions which happen to be stored in libraries are called library functions, and Mata's library functions are also stored in `.mlib` libraries. You can create your own libraries, too.

Mata provides two ways to store object code:

1. In a `.mo` file, which contains the code for one function
2. In a `.mlib` library file, which may contain the code for up to 2,048 functions

`.mo` files are easier to use and work just as well as `.mlib` libraries; see [M-3] [mata mosave](#). `.mlib` libraries, however, are easier to distribute to others when you have many functions, because they are combined into one file.

## Outline of the procedure for dealing with libraries

Working with libraries is easy:

1. First, choose a name for your library. We will choose the name `lpersonal`.
2. Next, create an empty library by using the `mata mlb create` command.
3. After that, you can add new members to the library at any time, using `mata mlb add`.

`.mlib` libraries contain object code, not the original source code, so you need to keep track of the source code yourself. Also, if you want to update the object code in a function stored in a library, you must re-create the entire library; there is no way to replace or delete a member once it is added.

We begin by showing you the mechanical steps, and then we will tell you how we manage libraries and source code.

### Creating a `.mlib` library

If you have not read [M-3] [mata mosave](#), please do so.

To create a new, empty library named `lpersonal.mlib` in the current directory, type

```
: mata mlb create lpersonal
(file lpersonal.mlib created)
```

If `lpersonal.mlib` already exists and you want to replace it, either erase the existing file first or type

```
: mata mlb create lpersonal, replace
(file lpersonal.mlib created)
```

To create a new, empty library named `lpersonal.mlib` in your `PERSONAL` (see [P] [sysdir](#)) directory, type

```
: mata mlb create lpersonal, dir(PERSONAL)
(file c:\ado\personal\lpersonal.mlib created)
```

or

```
: mata mlb create lpersonal, dir(PERSONAL) replace
(file c:\ado\personal\lpersonal.mlib created)
```

### Adding members to a `.mlib` library

Once a library exists, whether you have just created it and it is empty, or it already existed and contains some functions, you can add new functions to it by typing

```
: mata mlb add libname fcname()
```

So, if we wanted to add function `example()` to library `lpersonal.mlib`, we would type

```
: mata mlb add lpersonal example()
(1 function added)
```

In doing this, we do not have to say where `lpersonal.mlib` is stored; Mata searches for it along the `ado-path`.

Before you can add `example()` to the library, however, you must compile it:

```
: function example(...)
> {
>     ...
> }

: mata mlib add lpersonal example()
(1 function added)
```

You can add many functions to a library in one command:

```
: mata mlib add lpersonal example2() example3()
(2 functions added)
```

You can add all the functions currently in memory by typing

```
: mata mlib add lanother *()
(3 functions added)
```

In the above example, we added to `lanother.mlib` because we had already added `example()`, `example2()`, and `example3()` to `lpersonal.mlib` and trying to add them again would result in an error. (Before adding `*`, we could verify that we are adding what we want to add by typing `mata describe *`; see [\[M-3\] mata describe](#).)

## Listing the contents of a library

Once a library exists, you can list its contents (the names of the functions it contains) by typing

```
: mata describe using libname
```

Here we would type

```
: mata describe using lpersonal
(library contains 3 members)
```

# bytes	type	name and extent
32	auto transmorphic matrix	example()
32	auto transmorphic matrix	example2()
32	auto transmorphic matrix	example3()

`mata describe` usually lists the contents of memory, but `mata describe using` lists the contents of a library.

## Making it so Mata knows to search your libraries

Mata automatically finds the `.mlib` libraries on your `ado-path`. It does this when Mata is invoked for the first time during a session. Thus everything is automatic except that Mata will know nothing about any new libraries created during the Stata session, so after creating a new library, you must tell Mata about it. You do this by asking Mata to rebuild its library index:

```
: mata mlib index
```

You do not specify the name of your new library. That name does not matter because Mata rebuilds its entire library index.

You can issue the `mata mlib index` command right after creating the new library

```
: mata mlib create lpersonal, dir(PERSONAL)
: mata mlib index
```

or after you have created and added to the library:

```
: mata mlib create lpersonal, dir(PERSONAL)
: mata mlib add lpersonal *()
: mata mlib index
```

It does not matter. Mata does not need to rebuild its index after a known library is updated; Mata needs to be told to rebuild only when a new library is added during the session.

## Advice on organizing your source code

Say you wish to create and maintain `lpersonal.mlib`. Our preferred way is to use a do-file:

```
----- begin lpersonal.do -----
mata:
mata clear
function definitions appear here
mata mlib create lpersonal, dir(PERSONAL) replace
mata mlib add lpersonal *()
mata mlib index
end
----- end lpersonal.do -----
```

This way, all we have to do to create or re-create the library is enter Stata, change to the directory containing our source code, and type

```
. do lpersonal
```

For large libraries, we like to put the source code for different parts in different files:

```
----- begin lpersonal.do -----
mata: mata clear
do function1.mata
do function2.mata
...
mata:
mata mlib create lpersonal, dir(PERSONAL) replace
mata mlib add lpersonal *()
mata mlib index
end
----- end lpersonal.do -----
```

The function files contain the source code, which might include one function, or it might include more than one function if the primary function had subroutines:

---

begin `function1.mata`

```
mata:  
  function definitions appear here  
end
```

---

end `function1.mata`

---

We name our component files ending in `.mata`, but they are still just do-files.

### Also see

[M-3] `mata mosave` — Save function's compiled code in object file

[M-3] `intro` — Commands for controlling Mata