How to make a package in Mathematica

Nasser M. Abbasi

July 16, 2010 Compiled on October 27, 2025 at 7:51pm

Contents

This note describes making a simple package in Mathematica. The first section shows how to make a simple package. The second section shows how to make a package with subpackages in them and where to save the files so that they are easily loaded.

1 Simple package

An example package that contains one function is made showing how to save it and load it into Mathematica and to update it again by adding a second function to it.

Each time a new function is added to the package, the package has to be reloaded in order to use the new function.

This note is not meant by any means to be comprehensive, but only to give someone who is new at Mathematica the minimal set of commands to make a basic package.

More details about packages are found in Mathematica help pages.

1. Open a new text file called `foo.m` (using any text editor) and write in it the following

```
BeginPackage["foo`"]
Unprotect @@ Names["foo`*"];
ClearAll @@ Names["foo`*"];

f::usage = "f[x]"
Begin["`Private`"]

f[x_] := Module[{}, x^2];

End[];
Protect @@ Names["foo`*"];
EndPackage[];
```

   In the above `foo[x]` is a public function which can be accessed from outside.

2. Now save the file `foo.m`. Assuming it is saved to `C:/mydata/foo.m`

3. To use the function `f[x]` defined in the package, open Mathematica notebook and load the package as follows (this is one easy way to load the package, there are other ways).

```
Get["c:/mydata/foo.m"]
```

Now the package is loaded and you can call the function defined inside the package as follows

```
f[2]
```

The function can also be called by explicitly pre-appending the package name to the function (which I prefer, as it makes it clear where this function came from) as follows

```
foo`f[2]
```

4. To add a second function, say `g[x]` to the package, open `foo.m` again using text editor, and change it to look as follows:

```
BeginPackage["foo`"]
Unprotect @@ Names["foo`*"];
ClearAll @@ Names["foo`*"];

f::usage = "f[x]"
g::usage = "g[x]"
Begin["`Private`"]

f[x_] := Module[{}, x^2];
g[x_] := Module[{}, x^4];

End[];
Protect @@ Names["foo`*"];
EndPackage[];
```

Now save the file `foo.m`.

5. To use the new function added to the package, the package needs to be reloaded again

```
Get["c:/mydata/foo.m"]
```

6. To see which functions in your package type

```
?foo`*
```

That is all. To add more functions, repeat the above steps.

7. The above functions $f(x), g(x)$ are public functions. This is because each of them had a `usage` statement. We can add private functions that are used internally in the package, which the user can not call or see as follow. Any function that has no usage statement is a private function.

```
BeginPackage["foo`"]
Unprotect @@ Names["foo`*"];
ClearAll @@ Names["foo`*"];

f::usage = "f[x]"
g::usage = "g[x]"
Begin["`Private`"]

(*these are public functions *)
f[x_] := Module[{}, h[x]];
g[x_] := Module[{}, x^4];
```

```
(*this is private function because it has no usage statement*)
h[x_]:= Module[{},x^2];

End[];
Protect @@ Names["foo`*"];
EndPackage[];
```

I'd like to thank Bill Rowe and Istvan Zachar on the Math newsgroup for helpful suggestions while working on this note.

2 More advanced package

This package was developed on windows and all the files will be stored in

```
FileNameJoin[{$UserBaseDirectory, "Applications"}]
```

which on windows is located at

`C:\Users\Owner\AppData\Roaming\Mathematica\Applications`

the `Owner` is replaced by your user name. On Linux the above gives

```
FileNameJoin[{$UserBaseDirectory, "Applications"}]
 (* /home/me/.Wolfram/Applications *)
```

Now the package can be loaded by just typing

```
<<nma`
```

Where `nma` is our application name that we will make below. This way the application is loaded automatically from anywhere since Mathematica will search that folder automatically.

The application name is the same as the top level package name which lives in the file `nma.m`.

The application name is the name of the folder where we will put the files in under `Applications/` folder.

The package name is `nma`. So I created a folder called

```
C:\Users\Owner\AppData\Roaming\Mathematica\Applications\nma
```

And put all the m files there. The structure of the folder `nma/` is the following

```
/home/me/.Wolfram/Applications
                   |
                  nma/
                   |
                  nma.m   <-- top level package
                  dsolver.m         <-- subpackage
                  firstOrderODE.m   <-- subpackage
                  secondOrderODE.m  <-- subpackage
                  Kernel/
                    |
                  init.m   <-- important file. loads all packages
                           This file is automatically read by
                           Mathematica when doing <<nma`
                           but is NOT read when doing Get["nma.m"]
```

Logically the application `nma` is structured as follows. There is one top level package called `nma` which is always the same name as the application name (i.e. the name of the folder under Application). This package contains 3 packages, these are `dsolver,firstOrderODE,secondOrderODE`. Each one is in its own file as shown above.

The `init.m` purpose is to load the whole application when one does

```
<<nma`
```

Inside `init.m` we will put the code to read all the files.

```
(*file init.m*)
getFile[file_]:=Internal`InheritedBlock[{$ContextPath},Get[file]]
getFile["nma`firstOrderODE`"];
getFile["nma`secondOrderODE`"];
getFile["nma`dsolver`"];
getFile["nma`nma`"];
```

The above code avoids the shadowing problem when same public symbol in subpackage is used as top package.

Important note. The above structure of the tree, does not have to be put in `FileNameJoin[{$UserBaseDirec` to work.

It can be put in your own local folder anywhere on system. But for `<<nma‘` to work, the tree has to be on the `$Path`. Note that `Get["nma.m"]` will not work. It has to be `<<nma‘`, the reason is that we need to have init.m in the Kernel folder read and this only happens if we do `<<nma‘` and the tree has the special folder Kernel there with init.m in it.

So to put the tree anywhere, say in your local folder instead of in the system Application folder, simply add your local folder to the path as follows (in your notebook)

```
SetDirectory[NotebookDirectory[]];
$Path = Prepend[$Path, NotebookDirectory[]]
```

And now you can do `<<nma‘` and it will load the package and read `Kernel/init.m` from the local tree instead of from the system Application folder.

But it is important to have your local tree have `Applications/` folder ! else it will not work. So my local tree is this

```
/home/me/my_mathematica_apps/Applications
                    |
                   nma/
                    |
                   nma.m   <-- top level package
                   dsolver.m         <-- subpackage
                   firstOrderODE.m   <-- subpackage
                   secondOrderODE.m  <-- subpackage
                   test_notebook.nb
                   Kernel/
                     |
                   init.m   <-- important file. loads all packages
                            This file is automatically read by
                            Mathematica when doing <<nma`
                            but is NOT read when doing Get["nma.m"]
```

And then the command is

```
$Path = Prepend[$Path, "/home/me/my_mathematica_apps/Applications"]
```

Now `<<nma‘` will read Kernel/init.m OK from the above instead from the system Application folder. If the name `Applications/` was not there, it will not work. So watch out for this if you plane to move your Applications/ folder.

Now we will talk about the content of the actual packages.

The top level package `nma.m` is

```
BeginPackage["nma`"] (*do not add ,{"nma`dsolver`"} *)

Unprotect @@ Names["nma`*"];
ClearAll @@ Names["nma`*"];

dsolve::usage = "dsolve[ode,y[x],x] or dsolve[{ode,ic},y[x],x]"

Begin["`Private`"]
    dsolve[ode_,y_[x_],x_] := Module[{},
        nma`dsolver`dsolve[ode,y[x],x] (*need to have nma` there *)
    ];

End[];
Protect @@ Names["nma`*"];
EndPackage[]
```

Notice it has one entry point, function called `dsolve[ode_,y_[x_],x_]` which calls one of its internal packages. Also note that we do not explicitly load any subpackages in the code. This is because this is all done in init.m.

The subpackage is `dsolver.m` which is

```
BeginPackage["nma`dsolver`"] (*subpackage of nma *)

Unprotect @@ Names["nma`dsolver`*"];
ClearAll @@ Names["nma`dsolver`*"];

dsolve::usage = "dsolve[ode,y[x],x]";

Begin["`Private`"]
    dsolve[ode_,y_[x_],x_] := Module[{},
        parseODE[ode,y[x],x]
    ];

    parseODE[ode_,y_[x_],x_] := Module[{},
        nma`firstOrderODE`step[ode,y[x],x];
        nma`secondOrderODE`step[ode,y[x],x]
    ];

End[];
Protect @@ Names["nma`dsolver`*"];
EndPackage[]
```

Notice that the same symbol `dsolve` shows in the subpackage and in the main package. This is OK, since the init.m file had code to avoid shadowing problem.

The above package calls functions in two other packages. Here they are `firstOrderODE.m`

```
BeginPackage["nma`firstOrderODE`"] (*subpackage of nma *)
```

```
Unprotect @@ Names["nma`firstOrderODE`*"];
ClearAll @@ Names["nma`firstOrderODE`*"];

step::usage = "step[ode,y[x],x]";

Begin["`Private`"]

    step[ode_,y_[x_],x_] := Module[{},
        Print["in nma`firstOrderODE`step[]"]
    ];

End[];
Protect @@ Names["nma`firstOrderODE`*"];
EndPackage[]
```

And finally `secondOrderODE.m`

```
BeginPackage["nma`secondOrderODE`"] (*subpackage of nma *)

Unprotect @@ Names["nma`secondOrderODE`*"];
ClearAll @@ Names["nma`secondOrderODE`*"];

step::usage = "step[ode,y[x],x]";

Begin["`Private`"]

    step[ode_,y_[x_],x_] := Module[{},
        Print["in nma`secondOrderODE`step[]"]
    ];

End[];
Protect @@ Names["nma`secondOrderODE`*"];
EndPackage[]
```

This completes the template package **nma**. Now we can add more functions and more packages if needed. Any time we make changes we just need to do

```
<<nma`
```

To reload it. The above command can be done from any notebook without having to set any path, and Mathematica will find the application automatically.