Mathematica Tips, Tricks, and Techniques

$\boxed{\text{Packages}}$

Michael A. Morrison

(Version 2.4: February 2, 2000)

Contents

1 Package Basics (Everybody).

1.1 What is the safest, most efficient way to load a package?

Use `Needs`.

If, for example, you try to load the package `Graphics‘Graphics‘` using `Get["Graphics‘Graphics‘"]` and this package has already been loaded (e.g., by another package or by you earlier in your *Mathematica* session), then you'll get the dreaded *shadow error.* But if you type `Needs["Graphics‘Graphics‘"]`, then *Mathematica* will first *check to see whether the package has already been loaded.* If not, then *Mathematica* will loads it; if so, then *Mathematica* will do nothing.

1.2 When should I use `Get` instead of `Needs` to load a package?

Never.

The alternate syntax for `Get` is `<<`. So the command `<<Calculus‘LaplaceTransform‘` will cause *Mathematica* to load the package `Calculus‘LaplaceTransform‘` in the same way that `Get["Calculus‘LaplaceTransform‘"]`. But the safest way to load the package is to enter `Needs["Calculus‘LaplaceTransform‘"]`.[1]

1.3 Where should I put packages other than *Mathematica*'s standard packages?

All *Mathematica*'s built-in packages are in subdirectories which are in *Mathematica*'s built-in path . Put your packages (or those friends have given you) in your working directory, which you've set at the beginning of your notebook with `SetDirectory`. That way, you can load the package without having to remember and specify the name of the directory in which it's located.

Thus, to load a package named `monopole‘` which is contained in a file called `monopole.m`, you first copy `monopole.m` to your working directory, then type `Needs["monopole‘"]`. When you enter `Needs`, *Mathematica* first searches the directory it's in, then searches other directories on an internal path (called `ContextPath`). So long as you've told *Mathematica* to be in your temp directory (using `SetDirectory`) and you've copied the package into that directory, everything should proceed smoothly, as in the following example:

```
SetDirectory["\temp"];
Needs["monopole‘"]
```

1.4 What's the trickiest thing about packages?

Probably, the context name.

Here's what *The Mathematica Book* says about this topic:

> *In* Mathematica *the mechanism of "contexts" is used to keep package symbols separate from symbols in the main session. The basic idea is that the full name of any symbol is broken into two parts: a context and a short name. The full name is written as* `context‘short`*, where* ‘ *is the* **backquote** *or gravé accent character, called a* **context mark** *in* Mathematica*.*
>
> *Contexts in* Mathematica *work somewhat like file directories in many operating systems. You can always specify a particular file by giving its complete name, including its directory. But at any given point, there is usually a current working directory, analogous to the current* Mathematica *context. Variables that are in the current context can be specified just by giving their short names in the same way that files in the current working directory can be specified by just giving their short names. The global variable* `$Context` *gives the current* Mathematica *context.*
>
> *As is also the case with directories, contexts in* Mathematica *can be hierarchical. By convention, the symbols that are created by loading a standard* Mathematica *package have a context whose name is related to the name of the package. As an example, loading the package* `Calculus‘LaplaceTransform‘` *defines the symbol* `Calculus‘LaplaceTransform‘LaplaceTransform`*, which you can then use to compute the Laplace transform of a function.*

[1]If, unwisely, you use this shorthand `Get` syntax, remember that with this syntax *only*, the package name is *not* enclosed in double quotes. (This inconsistency is another reason I don't recommend using this form.)

1.5 What are the three most common errors made in using packages—and how can I avoid them?

1. **Context Marks!** The most common error is forgetting the context mark that appears at the end of the context name in any *Mathematica* package command. To avoid this error, start thinking of the context mark as *part* of the package name.

2. `Backquotes!` The second most common is using an apostrophe rather than a backquote for the context mark. The backquote ` lives on the bottom of the key with the tilde; this key usually appears in the upper left-hand corner of the keyboard. (Beware: The apostrophe ', or single quote, lurks at the bottom of the key with the double quote marks, which is usually next to the enter key.)

3. `Double Quotes!` The third most common error is forgetting the (double) quotes around the context name. These quotes *must* be present because a context name is a **string**—a sequence of text characters—and only the enclosing double quotes will make *Mathematica* treat it as a string. (Otherwise *Mathematica* will interpret the context name as a *symbol*, which it definitely is not!)

1.6 I've successfully loaded a package. How do I find out what new commands are available?

The same way you find out about other *Mathematica* commands: just ask.

If you want a list of all commands in the package `Quaternions`.m`, just enter `?Quaternions`*`. Note three familiar elements of this example:

1. We use `?` to inquire about *Mathematica* command(s).

2. We use the wildcard `*` to ask about *all Mathematica* commands in the context `Quaternions``.

3. We use the full package name—including the context mark—in our query.

1.7 Where do I find out about the *standard packages* which are supplied with *Mathematica*?

Go to the Help Browser and choose `AddOns`. In the AddOns menu, choose `StandardPackages`.

1.8 Where can I find lots of additional well-documented, carefully tested packages on the internet?

MathSource, on the internet.

1.9 The Help Browser claims not to know about a package. What's wrong?

Probably you have not chosen `Master Index` in the browser.

When the Help Browser first appears, it's preset to access `Built-in Function` only. You can get information about all commands in all standard packages by choosing `Master Index`.

2 Coping with package-related errors. (Everybody)

2.1 How can I avoid the dreaded *shadowing* error?

Always load packages you'll need in the Bookkeeping section at the start of your notebook, using `Needs`.

2.2 What does the error message `Get::noopen:` mean?

You tried to load a package *Mathematica* couldn't find. See the next question.

2.3 *Mathematica* claims it can't find a package! What do I do?

Check that you have correctly specified the folder (directory) that contains the package.

If you try to load a package called `physics` using `Needs["physics‘]` and *Mathematica* can't find the package, then it will respond (obliquely) with the error message

```
Get::noopen: Cannot open physics‘.
$Failed
```

All this means is that it couldn't find the package you asked for. If, for example, `physics.m` is in the directory `c:\temp` and this directory is *not* on the the *Mathematica* search path(see below), then you must tell *Mathematica* where it is, by entering `Needs["temp‘physics‘]`.

2.4 I tried to load a package and *Mathematica* gave me an error message `Needs::"nocont":`. Should I worry?

Nope.

This "error" just means that whoever wrote the package did not design it to establish a new context. If, for instance, you load a package called `ptstark.m`, then you'll get back this error message with the additional information

```
Context ptstark‘  was not created when Needs was evaluated.
```

(For once, a *Mathematica* error message that means just what it says.)

Most of the time this fact won't make any difference. All the commands in the package will be available to you.[2]

2.5 How can I ensure that *Mathematica* will never give me a `Needs::"nocont":` message?

In the `BookKeeping` section of your notebook, enter `Off[Needs::nocont]`.

3 How does *Mathematica*'s package naming scheme work? (Intermediate)

3.1 What is the context name of *Mathematica*'s built-in commands?

All built-in *Mathematica* objects are in the context `System‘`.

In spite of this fact, you never need refer to a command by its full name. That is, you enter `Integrate`, not `System‘Integrate`. This works because `System‘` is always on the list of contexts *Mathematica* searches when it tries to resolve an expression you've entered (this is called the `ContextPath` and is distinct from `$Path`, which is a list of directories (folders) on your hard disc.)

3.2 What context name is appended to the commands and expressions I define?

All user-defined *Mathematica* objects are assigned to the context `Global‘`.

Tip Although you don't really need to know about the `Symbol‘` context, knowing about `Global‘` is very handy. Usually you want to clean our *your* definitions but leave those in any packages you've defined. That is, you want to clean all defintions in the `Global‘` context. To do this, just enter

```
ClearAll["Global‘*"]
```

Note, as usual, the use of double quotes to demarcate the string, the context mark as the last character in the context name, and the wildcard to choose *all* commands in the requested context.

[2]Note, however, that there may arise conflicts between commands you have already defined and commands (with the same name) that are defined in the package, because all the commands in the package will be defined in the `Global‘` context.

3.3 How is the context name of a package related to the name of the file on disc that contains the contents of the package?

In many respects, a *Mathematica* **context** is *analogous* to a Windows folder (or directory). In this analogy, the **context mark**—that ‘ I keep telling you not to forget—plays the role of the directory pathname separator `\`. The name of the *current context*, which precedes the context mark, plays the role of the current working directory in Windows.

Thus the package `Limit.m` resides in the subdirectory `Calculus` in the directory `StandardPackages` (which is itself in the directory `AddOns` of the directory `3.0` of *`Mathematica`* of `Wolfram Research` of `Program Files`—whew!). In the corresponding context name `Calculus‘Limit‘`, the context mark after `Calculus` separates this context name from `Limit‘`. Thus we load this package by entering `Needs["Calculus‘Limit‘"]`. Note the enclosing double quotes—something else to remember.

3.4 How can I find the *file name* that corresponds to a *Mathematica context name*?

Use the *Mathematica* command `ContextToFileName`.

For example, if you enter `ContextToFileName["Calculus‘Limit‘"]`, *Mathematica* will return `Calculus\Limit.m`. This information tells you that the file `Limit.m` is in the directory `Calculus`.

3.5 How can I find out which package contains a particular *Mathematica* command?

Use the package `Utilities‘Package‘`.

Mathematica cleverly provides a package in the `Utilities` subdirectory that you can use to find other packages (if you know about it). To load it, enter `Needs["Utilities‘Package‘"]`. Then use the command `FindPackages`. There are several ways to use this command.

1. **To find all packages in a given directory**. If, for example, you want to know the name of the package files in a directory `c:\temp`, you need only enter `FindPackages["c:\temp"]`.
2. **To find all packages whose context names match a pattern.** For example, if you want a list of all packages whose names begin with `Graphics`, you enter `FindPackages[$Path,"Graphics*"]`. Note the use of the wild card after the string `Graphics`.
3. **To list all available packages**. Although I don't advise doing so, *Mathematica* allows you to enter `FindPackages[]` (no arguments) and fill your screen with the most cluttered list of packages imaginable. (For a clean list, enter the variable `$Packages`.)

Tip `FindPackages` returns the *context name*, not the file name.

3.6 Where should I put my packages?

Either in a directory somewhere on your hard disc other than in the *Mathematica* installation directory *or* in the `ExtraPackages` folder.

If, like me, you prefer to keep all your packages in a directory outside the installation directory in `ProgramFiles\Wolfram Research\Math\3.0`, then you need to modify the variable `$Path` in the `BookKeeping` section of your notebook to ensure that *Mathematica* will look in your directory when it tries to find packages you invoke with `Needs` (see below for details).

There is a simpler way to handle your packages. In the `AddOns` folder you'll find a subfolder called `StandardPacakges`. In it are subfolders containing all the standard packages supplied with *Mathematica*. Simiarly, you'll find a subfolder called `ExtraPackages`. Any package you put in this subfolder *or in any subfolder of it* will be automatically accessible by *Mathematica*. If you put your packages in `ExtraPackages` or in, say, `\ExtraPackages\MyPackages`, then you need not worry about modifying the variable `$Path`.

3.7 How can I get a list of the directories *Mathematica* checks when it looks for my packages?

Enter the global variable `$Path`.

This so-called "global variable" is where *Mathematica* keeps its list of directories to search for packages.

Tip If you want to add to this list the name of a directory where you keep your packages, use `Prepend`.

If, for instance, you keep your packages in a directory called `MyPackages`, then *in the `BookKeeping` section of your notebook*, enter

```
Prepend[$Path, "c:\MyPackages"]
```