Aliases.m

Jason F. Harris (j.harris@phys.canterbury.ac.nz)
Department of Physics, University of Canterbury
Christchurch, New Zealand

Abstract

This is a simple package to enable the easy use of aliases in *Mathematica*. It defines the functions: `Alias`, `UnAlias` and `ListAliases`. These respectively add an alias, remove aliases and list all current aliases.

1 Introduction

Loosely speaking, one can define an *alias* as an abbreviation or short hand for an expression. For instance an alias for "`NonCommutitiveMultiply`" might be "`**`". Aliases are useful in *Mathematica* in that they are possibly a shorter and/or more intuitive input form than their Full-Forms are.

`Aliases.m` is a package designed to enable one to easily add aliases to *Mathematica*. The ideas behind this package are not new (see §3). I was motivated to write this package because there was some confusion on the internet about the feasibility and technicalities of being able to implement aliases in *Mathematica* [2,3]. David Withoff [1] addressed these queries and indicated that the parsing system will be restructured for *Mathematica* version 2.3. This might lead to the ability to implement aliases without special packages resulting in this package becoming outdated in the near future. Nevertheless this package would remain useful for version 2.2 down o 2.0. It should be said that this package does nothing spectacular but just provides a nice interface to `$PreRead`.

I would like to acknowledge Michael Ibrahim for the question he posed in [2] which I have used to illustrate parts of this package (my personal uses of aliases have been a bit more mundane and esoteric).

The following sections consist of some definitions, a short description of the functions in this package, some examples, the basics of the implementation, and finally some comments.

1.1 Definitions

This section introduces some notation and terminology associated with "aliases" and "aliasing". However if the reader has an intuitive idea of these topics than this section can probably be skipped or at least skimmed.

We will define an *alias-description* to be the rule taking an alias to its non abbreviated form. For example the alias-description of "`**`" is

$$\texttt{"**"} \rightarrow \texttt{"NonCommutitiveMultiply"}$$

When we "Apply" an alias-description, say $alias \rightarrow fullForm$, we replace $alias$ with $fullForm$ throughout the current input expression. Consider the alias `+/-` for the infix operation of a user defined function `~Uncertainty~`. If we apply the alias-description `"+/-"` → `"~Uncertainty~"`, on the input `2 +/- 0.01` it would be replaced by `Uncertainty[2,0.01]` which would be passed on as the input expression.

We describe "aliasing" as the automatic application of an alias-description on input expressions. We will also use the terminology: $alias$ is "aliased" to $fullForm$. We will sometimes refer to aliases and alias-descriptions interchangeably.

Aliases are useful in *Mathematica* in that they are possibly shorter or more intuitive for input than their Full-Forms are e.g. compare

`2 +/- 0.01` to **`Uncertainty[2,0.01]`**

2 The Package

The package `Alias.m` contains three functions.

- `Alias[`$\mathit{theAlias} \rightarrow \mathit{theFullForm}, \mathit{opts}$`]` will make the string *theAlias* an alias for the string *theFullForm* for all future input expressions. Aliases are applied in the order: last added first applied.
- `UnAlias[`$\mathit{aFullForm}$`]` will remove all aliases that have *aFullForm* as their Full-From.
- `ListAliases[]` will list all of the current aliases in the order in which they are applied to input expressions.

we can demonstrate these functions with the following examples

We load the package as usual

In[1]:=

```
<<Aliases.m
```

We can now add aliases with the command `Alias`

In[2]:=

```
Alias["+/-" -> "~Uncertainty~"]
```

This has aliased the infix string "+/-" to `Uncertainty`

In[3]:=

```
3 +/- 0.05
```

Out[3]=

```
Uncertainty[3, 0.05]
```

In[4]:=

```
2 +/- 0.01 + 4 +/- 0.02
```

Out[4]=

```
Uncertainty[2, 0.01] + Uncertainty[4, 0.02]
```

It would then be up to the user to create meaningful rules for the combination of `Uncertainties`.

We can add more aliases

In[5]:=

```
Alias["**"->"MyNonCommutitaveMultiply"]
```

In[6]:=

```
**[2,3,4] + 4 +/- 0.02
```

Out[6]=

```
MyNonCommutitaveMultiply[2, 3, 4] + Uncertainty[4, 0.02]
```

We can view the current aliases by

In[7]:=

```
ListAliases[]
```

Out[7]=

```
** -> MyNonCommutitaveMultiply
+/- -> "Uncertainty"
$PreRead -> $NewPreRead
```

Note that `$PreRead` is in fact aliased to `$NewPreRead` and this will be explained in the next section.

We can remove aliases by `UnAlias`

In[8]:=

```
UnAlias["+/-"]
```

and now the alias for `"+/-"` is gone as can be seen by

In[9]:=

```
ListAliases[]
```

Out[9]=

```
** -> MyNonCommutitaveMultiply
$PreRead -> $NewPreRead
```

This completes the basic use of aliases in *Mathematica*. The following sections are only presented for those that will be using `$PreRead` and advanced substitutions.

`Alias.m` Also defines the variable `$NewPreRead` which is described in §4.

3 Implementation Basics

The implementation makes extensive use of the *Mathematica* global variable `$PreRead`, which is described in [5] as follows:

> ... you may want to modify this text string [the input string] before it is further processed by *Mathematica*. You can do this by assigning a function to the global variable `$PreRead`. This function will be applied to the text string, and the result will be used as the actual input string for the particular input line.

The basic idea behind `Aliases.m` is given in the *Mathematica* reference guide [5] §2.11.1 pgs 514-515. The idea is to set `$PreRead` to something like

```
$PreRead:= StringReplace[#,aliasDescriptions]&
```

This will perform a string replacement according to `aliasDescriptions` on every input expression. The variable `aliasDescriptions` is then changed to add or delete new aliases. e.g. to obtain *In[1]* we would set

```
aliasDescriptions = {"+/-" → "~Uncertantity~"}
```

This is the core idea behind Aliases.m

There would be nothing drastically wrong with directly setting up aliases in this way, but it is nice to have a friendlier interface. The functions `Alias` and `UnAlias` are just an easy way of adding and deleting aliases.

4 $NewPreRead

The package `Aliases.m` makes extensive use of `$PreRead` so it must introduce a new function, `$NewPreRead`, that fulfills `$PreRead`'s old role. `Aliases.m` aliases `$PreRead` to `$NewPreRead` in order to ensure that other packages are compatible with `Aliases.m`. The user can therefore be blissfully unaware that assigning a function to `$PreRead` actually sets `$NewPreRead`, so the user can use `$PreRead` just as before without any ill effects.

It is worth noting that an infix operator can be obtained by aliasing to ˜...˜, e.g. as was given above in `Alias["+/-"` → `"~Uncertainty~"]`.

5 The Option AddBeforeNewPreRead

The introduction of the variable `$NewPreRead` adds the extra complication of whether to apply the aliases before or after this function. This is addressed by the boolean option `AddBeforNewPreRead` of `Alias`. If this option is `True` the alias will be applied before evaluating `$NewPreRead`, otherwise the alias will be applied after evaluating `$NewPreRead`. The default option is True.

In[10]:=
```
Alias["bub"->"gub",AddBeforNewPreRead->False]
```

In[11]:=
```
ListAliases[]
```
Out[11]=
```
** -> MyNonCommutitaveMultiply
$PreRead -> $NewPreRead
bub -> gub
```

In[12]:=
```
$PreRead=If[StringMatchQ[#,"*d1*d2*"],
                HandleSpecial[#],#]&;
```

In[13]:=
```
??$NewPreRead
```
Out[13]=
```
$NewPreRead is the variable that the new $PreRead functions
   are stored in since the package Alias.m has to change that
   variable.

$NewPreRead =
  If[StringMatchQ[#1, "*d1*d2*"], HandleSpecial[#1], #1] &
```

In[14]:=
```
ListAliases[]
```
Out[14]=
```
** -> MyNonCommutitaveMultiply
$PreRead -> $NewPreRead
-- EXECUTION OF $NewPreRead --
bub -> gub
```

6 Compatibility with other packages

The only other package I have tested `Aliases.m` with that modifies the system is Todd Gayley's Package `CleanSlate.m` [6] . For compatibility `Aliases.m` must be loaded first. If `CleanSlate.m` is loaded first than it cannot remove `Aliases.m` or else`$PreRead` will get assigend `Removed[localPreRead][...]` which is garbage.

References

[1] David Withoff (withoff@wri.com) 14 January 1994 *Re: Special Input Forms* Appeared on the list-sever: mathgroup@yoda.physics.unc.edu.

[2] Michael Ibrahim (wasfy@gac.edu) 7 January 1994 *Special Input Forms* Appeared on the list-sever: mathgroup@yoda.physics.unc.edu.

[3] Roberto Sierra (bert@netcom.com) 13 January 1994 *Re: Special Input Forms* Appeared on the list-sever: mathgroup@yoda.physics.unc.edu.

[4] Roman Maeder 1990 *Programming in Mathematica* 2nd edn (Addison-Wesley, Redwood City, Calif).

[5] Stephen Wolfram 1991 *Mathematica: A System for Doing Mathematics by Computer* 2nd edn (Addison-Wesley).

[6] Todd Gayley 1993 *CleanSlate.m: A Package for Clearing Symbols and Freeing Memory* The *Mathematica* Journal **3,3** 46–51. Mathsource item number:0204-310.