InheritOptions package

Mark A. Caprio, Department of Physics, University of Notre Dame

Version 2.0 (June 22, 2011)

Introduction

Often it is necessary to define several related functions which accept similar sets of options. (For instance, many of the *Mathematica* plotting functions accept similar sets of options.) It is useful for such related functions to be able to share the default values for their options, so that a default value set for one can affect all the others as well. The InheritOptions package provides a framework for this, as well as for much more sophisticated possibilities.

The InheritOptions package applies the concept of "inheritance", one of the foundations of object oriented programming, to *Mathematica*'s option system. Inheritance is the process by which one object, known as the "child" object, can be automatically assigned some of the properties of another type of object, known as the "parent" object. (For instance, in object oriented programming, the objects are usually data structures, and the child data structure automatically contains some or all of the data fields defined for the parent.)

The InheritOptions package implements a form of inheritance in which a child symbol inherits the default values of options from a parent symbol. Recall that, under the basic *Mathematica* option system, the value for an option for a symbol can be explcitly specified as an argument (*e.g.*, `Plot[`*args*`,`**`Frame→True`**`]`), but that if no value is specified the value is taken from the list of rules `Options[`*symbol*`]` which gives the default values. Under the inheritance system set up by the InheritOptions package, there is another possibility. If the value of an option for symbol *child* given as `Inherited`, then the value used for it can be obtained from the list of defaults `Options[`*parent*`]` for a parent symbol *parent*.

Under the InheritOptions package, one child symbol can inherit different options from different parent symbols ("multiple inheritance"), and an arbitrary number of generations of inheritance are possible (a child inheriting an option from a parent which in turn inherits the option from a grandparent, *etc.*). The code implementing inheritance provides validation of the options, including checking for unknown option names in the list of options given to a symbol.

The package also provides several general option processing utilities, not directly related to inheritance, including utilities to sort and combine lists of options.

How the default options for functions are defined

First one or more "parent" symbols (functions) must be defined which do not inherit any options. Then, other "child" symbols can be defined, inheriting their options from these parents.

`DefineOptions[` *symbol* `,` `{` *option1* → *value1* `,` ... `}]`	Defines the options for *symbol*, with no inherited options; but all necessary setup is carried out for *symbol* to later be used as a parent symbol

Option definition for a function with no parents.

The options for the parent must be defined using `DefineOptions`, as shown above. To define the options for *symbol*, this `DefineOptions` must be given a list of rules for the default values of *symbol*'s options. This list serves two slightly distinct purposes: it enumerates the names of the valid options for *symbol*, and it defines their default values. `DefineOptions` defines `Options[`*symbol*`]` to be the given list of rules, as usual under the *Mathematica* option

system. But is also defines a another set of rules, `OptionRealizationRules[`*symbol*`]`, used later in the option inheritance process as discussed below.

A classic example from object oriented programming is to define an object representing a geometric point, from which other geometric objects are derived as children. A point is characterized by its coordinates, which we choose here so that the point is by default at the origin.

```
DefineOptions[GeomPoint,{Coordinates→{0.,0.}}]
```

This results in the definition of `Options[GeomPoint]` as

```
{Coordinates→{0.,0.}}
```

`DefineOptions[` *symbol* `,` `{` *parent* `, All}, ... ,` `{` *newoption1* → *newvalue1* `, ... }]`	Defines the options for *symbol*, inheriting all the options of *parent*
`DefineOptions[` *symbol* `,` `{` *parent* `, {` *option1* `, ... }}, ... ,` `{` *newoption1* → *newvalue1* `, ... }]`	Defines the options for *symbol*, inheriting only some of the options of *parent*
`DefineOptions[` *symbol* `,` `{` *parent* `, {` *option1* → *value1* `, ... }}, ... ,` `{` *newoption1* → *newvalue1* `, ... }]`	Defines the options for *symbol*, inheriting only some of the options of *parent*, and overriding the default value for *option1* to something other than `Inherited`
`DefineOptions[` *symbol* `,` `{` *parent* `, {` *option1* → *value1* `, ... }, All}, ... ,` `{` *newoption1* → *newvalue1* `, ... }]`	Defines the options for *symbol*, inheriting all of the options of *parent*, but overriding the default value for *option1* to something other than `Inherited`

Option definition for child function.

Now we consider defining the options for a child symbol. The simplest possible case is that the child symbol inherits all the options which are defined for the parent, as in the first case in the box above. For our illustration we define a circle as having coordinates (inherited from the point) and also a radius, with unit radius as the default. The code

```
DefineOptions[GeomCircle,{GeomPoint,All},{Radius→1}]
```

results in the definition of `Options[GeomCircle]` as

```
{Coordinates→Inherited,Radius→1}
```

It also produces a definition for `OptionRealizationRules[GeomCircle]`, which contains the information necessary for the option `Coordinates` to be inherited from `GeomPoint` when its value is given as `Inherited`.

Alternatively, the second syntax in the box above may be used. In this case, only some options, explicitly enumerated, are inherited from the parent. For our geometric example, we could have equivalently defined the options for the circle with

```
DefineOptions[GeomCircle,{GeomPoint,{Coordinates}},{Radius→1}]
```

Even more flexibility is provided by the third syntax. A default value other than `Inherited` can be specified for the inherited option. But the information necessary for inheritance is still stored in `OptionRealizationRules`. Thus, if the value `Inherited` ever is encountered for the option (perhaps if the user later sets the default value to `Inherited` with `SetOptions`, or if the user explicitly gives the value `Inherited` as an argument), the value will be inherited from the parent. For the `GeomCircle` example, we could have chosen circles to by default appear at some specific location

other than the origin but left open the possibility of inheriting the coordinates from `GeomPoint` through an explicit specification of `Inherited`. This would be accomplished by

```
DefineOptions[GeomCircle,{GeomPoint,{Coordinates→{1.,1.}}},{Radius→1}]
```

which defines `Options[GeomCircle]` as

```
{Coordinates→{1.,1.},Radius→1}
```

Note that a symbol may be defined to inherit some options from one parent and some from another, a situation referred to a "multiple inheritance". As an example, consider a function `BoxedText`, producing text framed in a box. It could be derived from two simpler functions, one for producing a colored box and one for drawing text, in which case it would require the options appropriate to both.

```
DefineOptions[ColoredBox,{FillColor→Red,BorderColor→Black}];
DefineOptions[PlainText,{FontFamily->"Times",FontColor→Black}];
DefineOptions[BoxedText,{ColoredBox,All},{PlainText,All},{}];
```

This defines `Options[BoxedText]` as

```
{BorderColor→Inherited,FillColor→Inherited,FontColor→Inherited,FontFamily→
Inherited}
```

There is no limit on the number of parents one child may have.

Review: How functions using options are conventionally defined in Mathematica

Before we consider how to implement functions which make use of inherited options, let us review how options are *conventionally* used within a function in *Mathematica*, *without* inheritance. Option values given explicitly to a function as arguments override the default values in `Options[`*symbol*`]`. Most *Mathematica* functions are very tolerant about how these option arguments are given: the options may be enclosed in lists or even nested sublists. An example of eccentric but acceptable nesting would be `Plot[`*args*`,PlotPoints→1000,{{Axes→False,{Frame→True},PlotPoints→33}}]`. If several rules for the same option are given, the leftmost rule applies.

The first step a function must carry out is to construct a complete list of rules giving the values to be used for all the options. For example, an ordinary *Mathematica* function analogous to the circle example above (but without inheritance) could be implemented as

```
Options[ConventionalCircle]={Coordinates→{0.,0.},Radius→1};
ConventionalCircle[Opts___?OptionQ]:=Module[
      {FullOpts=Flatten[{Opts,Options[ConventionalCircle]}]},
      Circle[Coordinates/.FullOpts,Radius/.FullOpts]
      ];
```

Here `Flatten` creates a flattened list containing all the rules given for the options: first those given explicitly as arguments (`Opts`), then the default rules (`Options[ConventionalCircle]`). Since the rules given in `Opts` appear first in the list, they take precedence over the defaults when `/.` is used to extract the values. Some example inputs and outputs are

```
ConventionalCircle[]
```

```
Circle[{0., 0.}, 1]
```

```
ConventionalCircle[Radius → 2]
```

```
Circle[{0., 0.}, 2]
```

How functions using inherited options are defined

`RealizeOptions[` *symbol*`,` (*option rules*)`]`	Produces a finalized list of values to be used for all the options for *symbol*, inherited from parents as appropriate

Option realization function for inherited options.

To write a function which makes use of inherited options, only one modification to the conventional approach is necessary. The function `RealizeOptions` must be used instead of `Flatten` to obtain the final list of rules for the options. `RealizeOptions` first combines any rules for options given explicitly as arguments with the list of default options, much as above. (In fact, it is "tidier", in that it also keeps only the first, highest-precedence rule for each option, yielding a slightly more compact list of options.) If the rule for any of the options in this list gives the option value as `Inherited`, the appropriate parent symbol's options are checked for the value. And if this value is `Inherited`, the appropriate grandparent is checked, *etc.*

`RealizeOptions` performs two validity checks on the option rules it is given. If a value `Inherited` is encountered for an option which is not inheritable, at any stage of looking at parent, grandparent, *etc.*, values for an option, an error message is generated. Also, if a rule is specified for anything which is not a known option (as defined with `DefineOptions`) for the symbol, an error message is generated. It is thus useful to use `RealizeOptions` in the implementation of *any* function defined using `DefineOptions`, even if that function does not inherit any options, to obtain the benefit of this option validation.

As an example, consider the full implementation of the circle example discussed above.

```
DefineOptions[GeomPoint,{Coordinates→{0.,0.}}];
DefineOptions[GeomCircle,{GeomPoint,All},{Radius→1}];
GeomCircle[Opts___?OptionQ]:=Module[
      {FullOpts=RealizeOptions[GeomCircle,Opts]},
      Circle[Coordinates/.FullOpts,Radius/.FullOpts]
      ];
```

The following inputs and outputs show the list of option values constructed by `RealizeOptions`

```
Options[GeomCircle]
RealizeOptions[GeomCircle]
RealizeOptions[GeomCircle, Coordinates → {5.1, 3.4}]
```

```
{Coordinates → Inherited, Radius → 1}

{Coordinates → {0., 0.}, Radius → 1}

{Coordinates → {5.1, 3.4}, Radius → 1}
```

and the results for the function as a whole

```
GeomCircle[]
```

```
Circle[{0., 0.}, 1]
```

```
GeomCircle[Coordinates → {5.1, 3.4}]
```

```
Circle[{5.1, 3.4}, 1]
```

General option processing utilities

`KnownOptions[` *symbol*`]`	Returns a sorted list of the known option names for *symbol*, as defined in `Options[` *symbol*`]`
`OptionsSort[` *options*`]`	Flattens a set of option specifications and sorts them by option name; order is preserved for rules for the same option
`OptionsUnion[` *options*`]`	Flattens a set of option specifications and sorts them by option name; only the rule which was originally leftmost (highest precedence) for each option in preserved
`DuplicatedOptions[` *options*`]`	Returns a list of all options which are specified more than once in a set of option specifications
`FlatOptionListQ[` *x*`]`	Returns `True` if *x* is a flat list of option rules; this is a more stringent test than *Mathematica*'s `OptionQ`; the pattern accepted for an option rule is `(_Symbol → _) | (_Symbol :→ _)`

Option list manipulation utilities.

`KnownOption[` *symbol*`,` *option*`]`	Returns `True` if *option* is a known option for *symbol*, as defined in `Options[` *symbol*`]`

Utility function for testing definition of an individual option.

Technical notes

Notes on defining the options

The option default value rules given to `DefineOptions`, either for overriding the default values of inherited options or in the list of new options, may be either of the type `Rule` (→) or of the type `RuleDelayed` (:→).

An option cannot be defined more than once for the same symbol. Thus, an error will result if the same option is inherited from more than one parent, or listed more than once in the list of options to be inherited from a single parent, listed both as inherited and in the new option list, or listed more than once in the new option list.

The list of new options given to `DefineOptions` can have nested sublists (which will just be flattened out), but the lists of inherited options must be flat.

The list of option value rules stored in `Options[`*symbol*`]` is sorted "canonically" (*i.e.*, alphabetically) by option name.

`OptionRealizationRules[`*symbol*`]` is associated with *symbol* as an upvalue. It is stored either as a list of rules or as a dispatch table (`Dispatch`), whichever is more efficient.

Notes on realizing the options

The value `Inherited` for an option may be specified with `Rule` or `RuleDelayed`, interchangeably.

The steps carried out by `RealizeOptions` are as follows. First, to combine the explicit options with the default option list, `RealizeOptions` uses `OptionsUnion`, in the process removing duplicate values. It then applies the rules contained in `OptionRealizationRules[`*symbol*`]` to the unioned list of options. Applying `OptionRealizationRules[`*symbol*`]` replaces any option specification of the form *option*→`Inherited` or *option*:→`Inherited`, for a known and inheritable option, with the rule for *option* given in `Options[`*parent*`]`, where *parent* is the appropriate parent symbol. If that rule also gives the value `Inherited`, `OptionRealizationRules[`*parent*`]` is applied, *ad nauseum*.

When `RealizeOptions` applies `OptionRealizationRules[`*symbol*`]`, this has the effect of applying error trapping rules contained in `OptionRealizationRules[`*symbol*`]`. If a rule of the form *option*→`Inherited` or *option*:→ `Inherited` is encountered for an option which is known but *not* inheritable, the global error handler `$UninheritableOption` is called with the symbol name, option name, and full rule as arguments. If a rule is encountered for an unknown option, the global error handler `$UnknownOption` is called, with the same set of arguments. The default error handlers generate a warning message and return the rule unchanged, but the user can define alternate error handlers. The validity checks in `OptionRealizationRules[`*symbol*`]` can be completely omitted by specifying the options `TrapUninheritableOptions` and `TrapUnknownOptions` as `False` when `OptionRealizationRules[`*symbol*`]` is first created with `DefineOptions`.

Further resources

Examples of *Mathematica* code which rely heavily upon the InheritOptions inheritance scheme are the LevelScheme [M. A. Caprio, Comput. Phys. Commun. **171**, 107 (2005)] and SciDraw scientific figure preparation systems.

Mathematica version

This package requires *Mathematica* version 6 or above.

© Copyright 2011, Mark A. Caprio.