New developments for gretl function packages

Allin Cottrell

Wake Forest University

Gretl Conference, Gdańsk, June 2023

Background

I'm returning to a topic I spoke about at Bilbao (2009) and Toruń (2011): gretl function packages.

Brief recap on the role of function packages...

Since November 2021: about 68000 downloads of 205 packages. 165 packages had more than 100 downloads; 69 of them had more than 400 downloads. (Greatest number for a single package: 1276 for lp-mfx.)

Background

I'm returning to a topic I spoke about at Bilbao (2009) and Toruń (2011): gretl function packages.

Brief recap on the role of function packages...

Since November 2021: about 68000 downloads of 205 packages. 165 packages had more than 100 downloads; 69 of them had more than 400 downloads. (Greatest number for a single package: 1276 for lp-mfx.)

Background

I'm returning to a topic I spoke about at Bilbao (2009) and Toruń (2011): gretl function packages.

Brief recap on the role of function packages...

Since November 2021: about 68000 downloads of 205 packages. 165 packages had more than 100 downloads; 69 of them had more than 400 downloads. (Greatest number for a single package: 1276 for `lp-mfx`.)

New developments at a glance

- Markdown for package documentation. (Prior forms: plain text or PDF).
- Facility for fine-tuning the dialog box shown for a package (relates to the "GUI hook" for packages).
- Support for relatively tight integration with R (provide gretl interfaces for selected R packages).

New developments at a glance

- Markdown for package documentation. (Prior forms: plain text or PDF).
- Facility for fine-tuning the dialog box shown for a package (relates to the "GUI hook" for packages).
- Support for relatively tight integration with R (provide gretl interfaces for selected R packages).

New developments at a glance

- Markdown for package documentation. (Prior forms: plain text or PDF).
- Facility for fine-tuning the dialog box shown for a package (relates to the "GUI hook" for packages).
- Support for relatively tight integration with R (provide gretl interfaces for selected R packages).

Markdown for documentation

Gretl's markdown conversion piggy-backs off Uwe Jugel's md2pango.

But uses GLib regular expressions rather than Javascript, and targets GtkTextTags rather than pango markup.

To use markdown, give your help file an `.md` suffix. In the spec file do:

```
help = myhelp.md
```

and/or

```
guihelp = myguihelp.md
```

Markdown for documentation

Gretl's markdown conversion piggy-backs off Uwe Jugel's md2pango.

But uses GLib regular expressions rather than Javascript, and targets GtkTextTags rather than pango markup.

To use markdown, give your help file an `.md` suffix. In the spec file do:

```
help = myhelp.md
```

and/or

```
guihelp = myguihelp.md
```

Markdown for documentation

Gretl's markdown conversion piggy-backs off Uwe Jugel's md2pango.

But uses GLib regular expressions rather than Javascript, and targets GtkTextTags rather than pango markup.

To use markdown, give your help file an `.md` suffix. In the spec file do:

```
help = myhelp.md
```

and/or

```
guihelp = myguihelp.md
```

Markdown for documentation

Gretl's markdown conversion piggy-backs off Uwe Jugel's md2pango.

But uses GLib regular expressions rather than Javascript, and targets GtkTextTags rather than pango markup.

To use markdown, give your help file an `.md` suffix. In the spec file do:

```
help = myhelp.md
```

and/or

```
guihelp = myguihelp.md
```

Markdown ‘flavour’

- First and second-level headings: start a (single) line with # or ##
- Boldface: `**text**`
- Italic: `*text*` or `_text_`
- Monospace: ‘`text`‘
- Itemized list: each item starts with “`- `” on a new line
- Enumerated list: each item starts with (e.g.) “`1. `” on a new line
- Code block: starts and ends with ‘ ‘ ‘ on its own line

More markdown details

- Itemized and enumerated lists cannot be nested.
- `http[s]` URLs turn into hyperlinks.
- ASCII straight double quotes replaced by left- and right-hand quotes.

In principle the range of conversions could be extended if necessary.

Take a look at an example. . .

More markdown details

- Itemized and enumerated lists cannot be nested.
- `http[s]` URLs turn into hyperlinks.
- ASCII straight double quotes replaced by left- and right-hand quotes.

In principle the range of conversions could be extended if necessary.

Take a look at an example. . .

More markdown details

- Itemized and enumerated lists cannot be nested.
- `http[s]` URLs turn into hyperlinks.
- ASCII straight double quotes replaced by left- and right-hand quotes.

In principle the range of conversions could be extended if necessary.

Take a look at an example. . .

More markdown details

- Itemized and enumerated lists cannot be nested.
- `http[s]` URLs turn into hyperlinks.
- ASCII straight double quotes replaced by left- and right-hand quotes.

In principle the range of conversions could be extended if necessary.

Take a look at an example. . .

More markdown details

- Itemized and enumerated lists cannot be nested.
- `http[s]` URLs turn into hyperlinks.
- ASCII straight double quotes replaced by left- and right-hand quotes.

In principle the range of conversions could be extended if necessary.

Take a look at an example. . .

Function package dialog box

Prior means of inflecting the widgets in a package's dialog box, via "decoration" of the parameters in a function's signature.

- Default values.
- Labels for parameters.
- Labels for values of discrete integer parameters.
- `bool` parameter type → check box shown.

Function package dialog box

Prior means of inflecting the widgets in a package's dialog box, via "decoration" of the parameters in a function's signature.

- Default values.
- Labels for parameters.
- Labels for values of discrete integer parameters.
- `bool` parameter type → check box shown.

Function package dialog box

Prior means of inflecting the widgets in a package's dialog box, via "decoration" of the parameters in a function's signature.

- Default values.
- Labels for parameters.
- Labels for values of discrete integer parameters.
- `bool` parameter type → check box shown.

Function package dialog box

Prior means of inflecting the widgets in a package's dialog box, via "decoration" of the parameters in a function's signature.

- Default values.
- Labels for parameters.
- Labels for values of discrete integer parameters.
- `bool` parameter type → check box shown.

Function package dialog box

Prior means of inflecting the widgets in a package's dialog box, via "decoration" of the parameters in a function's signature.

- Default values.
- Labels for parameters.
- Labels for values of discrete integer parameters.
- `bool` parameter type → check box shown.

GUI-oriented signature: example

```
function bundle GUI_rf (\
  series y "dependent variable",
  list X "independent variables",
  scalar pctrain[20:95:60:1] "training data, %",
  int mode[0:2:0] {"auto", "regress", "classify"},
  bool tune[1] "tune 'mtry'",
  int verbosity [1:2:2])
```

And let's see the GTK representation...

GUI-oriented signature: example

```
function bundle GUI_rf (\
  series y "dependent variable",
  list X "independent variables",
  scalar pctrain[20:95:60:1] "training data, %",
  int mode[0:2:0] {"auto", "regress", "classify"},
  bool tune[1] "tune 'mtry'",
  int verbosity [1:2:2])
```

And let's see the GTK representation...

New inflections

- A given parameter (of any type) can be marked as *dependent* on a specified boolean parameter.
- A parameter of type `int` can be given a data-dependent default, minimum and/or maximum value (to be fixed at run time).
- A parameter of type `list` can be inflected in up to three ways (more on this shortly).

We don't try to cram these things into the signature of the function in question. Instead we use an auxiliary bundle.

New inflections

- A given parameter (of any type) can be marked as *dependent* on a specified boolean parameter.
- A parameter of type `int` can be given a data-dependent default, minimum and/or maximum value (to be fixed at run time).
- A parameter of type `list` can be inflected in up to three ways (more on this shortly).

We don't try to cram these things into the signature of the function in question. Instead we use an auxiliary bundle.

New inflections

- A given parameter (of any type) can be marked as *dependent* on a specified boolean parameter.
- A parameter of type `int` can be given a data-dependent default, minimum and/or maximum value (to be fixed at run time).
- A parameter of type `list` can be inflected in up to three ways (more on this shortly).

We don't try to cram these things into the signature of the function in question. Instead we use an auxiliary bundle.

New inflections

- A given parameter (of any type) can be marked as *dependent* on a specified boolean parameter.
- A parameter of type `int` can be given a data-dependent default, minimum and/or maximum value (to be fixed at run time).
- A parameter of type `list` can be inflected in up to three ways (more on this shortly).

We don't try to cram these things into the signature of the function in question. Instead we use an auxiliary bundle.

An example

We write a function which returns a bundle, containing a sub-bundle named for each parameter we want to inflect—here seed, n_train and X.

```
function bundle rf_ui_maker (void)
  maxstr = "ceil(0.9*$nobs)"
  defstr = "ceil(0.65*$nobs)"
  bundle b
  b["seed"] = _(depends="use_seed")
  b["n_train"] = _(maximum=maxstr, default=defstr)
  b["X"] = _(singleton=0, exclude="y", no_const=1)
  return b
end function
```

Hooking it up

The function we just saw is registered in the spec file for the package:

```
ui-maker = rf_ui_maker
```

It will then be called when the user selects the menu item associated with `GUI_rf()`, guiding the GUI treatment of the parameters.

We'll use an expanded signature in this case, adding these two parameters to `GUI_rf()`:

```
bool use_seed[0] "set random seed"
int seed[0:2147483647:1234567]
```

Hooking it up

The function we just saw is registered in the spec file for the package:

```
ui-maker = rf_ui_maker
```

It will then be called when the user selects the menu item associated with `GUI_rf()`, guiding the GUI treatment of the parameters.

We'll use an expanded signature in this case, adding these two parameters to `GUI_rf()`:

```
bool use_seed[0] "set random seed"
int seed[0:2147483647:1234567]
```

Hooking it up

The function we just saw is registered in the spec file for the package:

```
ui-maker = rf_ui_maker
```

It will then be called when the user selects the menu item associated with `GUI_rf()`, guiding the GUI treatment of the parameters.

We'll use an expanded signature in this case, adding these two parameters to `GUI_rf()`:

```
bool use_seed[0] "set random seed"
int seed[0:2147483647:1234567]
```

Recap of the content of the bundle b returned by the `ui-maker` function:

```
b["seed"] = _(depends="use_seed")
b["n_train"] = _(maximum=maxstr, default=defstr)
b["X"] = _(singleton=0, exclude="y", no_const=1)
```

`singleton=0`	require 2 or more series for list X
`exclude="y"`	exclude the series selected as y
`no_const=1`	and don't allow inclusion of `const`

Could add more keywords besides those above: suggestions?

R integration

First question: How much R integration do we actually *want*?

How to call R from gretl?

- Call the R executable, or
- Call into the R shared library.

The first option is simpler; the second is more efficient. Depends on loading certain R header files at build time. We do this for our Windows and macOS builds.

R integration

First question: How much R integration do we actually *want*?

How to call R from gretl?

- Call the R executable, or
- Call into the R shared library.

The first option is simpler; the second is more efficient. Depends on loading certain R header files at build time. We do this for our Windows and macOS builds.

R integration

First question: How much R integration do we actually *want*?

How to call R from gretl?

- Call the R executable, or
- Call into the R shared library.

The first option is simpler; the second is more efficient. Depends on loading certain R header files at build time. We do this for our Windows and macOS builds.

R_functions

When we're using the R shared library, we can define and "install" R functions, and can then call them as if they were hansl functions. Trivial example from the *Gretl User's Guide*:

```
set R_functions on
foreign language=R
   plus_one <- function(q) {
      z = q+1
      invisible(z)
   }
end foreign
scalar b=R.plus_one(2)
```

So what's actually new?

There's a new "special role" for a function within a package, namely `R-setup`. For example, in the spec file:

```
R-setup = rf_setup
```

This function should simply define one or more R functions, within a gretl `foreign` block.

When such a package is loaded, its `R-setup` is executed automatically. Subsequently, the R function(s) can be called as if they were native.

Let's look at an example...

So what's actually new?

There's a new "special role" for a function within a package, namely `R-setup`. For example, in the spec file:

```
R-setup = rf_setup
```

This function should simply define one or more R functions, within a gretl `foreign` block.

When such a package is loaded, its `R-setup` is executed automatically. Subsequently, the R function(s) can be called as if they were native.

Let's look at an example...

So what's actually new?

There's a new "special role" for a function within a package, namely `R-setup`. For example, in the spec file:

```
R-setup = rf_setup
```

This function should simply define one or more R functions, within a gretl `foreign` block.

When such a package is loaded, its `R-setup` is executed automatically. Subsequently, the R function(s) can be called as if they were native.

Let's look at an example...

So what's actually new?

There's a new "special role" for a function within a package, namely `R-setup`. For example, in the spec file:

```
R-setup = rf_setup
```

This function should simply define one or more R functions, within a gretl `foreign` block.

When such a package is loaded, its `R-setup` is executed automatically. Subsequently, the R function(s) can be called as if they were native.

Let's look at an example...