# class GTK::V3::Glib::GObject

GObject — The base object type

## Table of Contents

1 Synopsis
1.1 [g_object_] set_property
1.2 [g_object_] get_property
1.3 new
1.3.1 multi submethod BUILD ( :$widget! )
1.3.2 multi submethod BUILD ( Str :$build-id! )
1.4 debug
1.5 register-signal

## Synopsis

Top level class of almost all classes in the GTK, GDK and Glib libraries.

This object is almost never used directly. Most of the classes inherit from this class. The below example can be made much simpler by setting the label directly in the init of GtKLabel. The purpose of this example, however, is that there are other properties which can only be set this way. Also not all types are covered yet by GValue and GType.

```
use GTK::V3::Glib::GObject;
use GTK::V3::Glib::GValue;
use GTK::V3::Glib::GType;
use GTK::V3::Gtk::GtkLabel;

my GTK::V3::Glib::GType $gt .= new;
my GTK::V3::Glib::GValue $gv .= new(:init(G_TYPE_STRING));

my GTK::V3::Gtk::GtkLabel $label1 .= new(:label(''));
$gv.g-value-set-string('label string');
$label1.g-object-set-property( 'label', $gv);
```

### [g_object_] set_property

```
method g_object_set_property (
  Str $property_name, GTK::V3::Glib::GValue $value
)
```

Sets a property on an object.

### [g_object_] get_property

```
method g_object_get_property (
  Str $property_name, GTK::V3::Glib::GValue $value is rw
)
```

Gets a property of an object. value must have been initialized to the expected type of the property (or a type to which the expected type can be transformed) using g_value_init().

In general, a copy is made of the property contents and the caller is responsible for freeing the memory by calling g_value_unset().

### new

#### multi submethod BUILD ( :$widget! )

Create a Perl6 widget object using a native widget from elsewhere. $widget can be a N-GOBject or a Perl6 widget like GTK::V3::Gtk::GtkButton.

```
# some set of radio buttons grouped together
my GTK::V3::Gtk::GtkRadioButton $rb1 .= new(:label('Download everything'));
my GTK::V3::Gtk::GtkRadioButton $rb2 .= new(
  :group-from($rb1), :label('Download core only')
);

# get all radio buttons of group of button $rb2
my GTK::V3::Glib::GSList $rb-list .= new(:gslist($rb2.get-group));
loop ( Int $i = 0; $i < $rb-list.g_slist_length; $i++ ) {
  # get button from the list
  my GTK::V3::Gtk::GtkRadioButton $rb .= new(
    :widget($rb-list.nth-data-gobject($i))
  );

  if $rb.get-active == 1 {
    # execute task for this radio button

    last;
  }
}
```

Another example is a difficult way to get a button.

```
my GTK::V3::Gtk::GtkButton $start-button .= new(
  :widget(GTK::V3::Gtk::GtkButton.gtk_button_new_with_label('Start'))
);
```

### multi submethod BUILD ( Str :$build-id! )

Create a Perl6 widget object using a GtkBuilder. The GtkBuilder class will handover its object address to the GObject and can then be used to search for id's defined in the GUI glade design.

```
my GTK::V3::Gtk::GtkBuilder $builder .= new(:filename<my-gui.glade>);
my GTK::V3::Gtk::GtkButton $button .= new(:build-id<my-gui-button>);
```

## debug

```
method debug ( Bool :$on )
```

There are many situations when exceptions are retrown within code of a callback method, Perl6 is not able to display the error properly (yet). In those cases you need another way to display errors and show extra messages leading up to it.

## register-signal

```
method register-signal (
  $handler-object, Str:D $handler-name, Str:D $signal-name,
  Int :$connect-flags = 0, *%user-options
  --> Bool
)
```

Register a handler to process a signal or an event. When the handler has a **named argument** named **$event** it is assumed that the handler code is made is to handle events like key presses. Otherwise it is assumed that the code handles signals like button clicks. Information about signal names available to widgets can be found at the GTK developers site, e.g. for GtkButton click signals here and for a mouse button press event here. Notice that in the latter case you can see that one of the arguments has an argument type of **GdkEvent**.

- $handler-object is the object wherein the handler is defined.
- $handler-name is name of the method. Its signature is one of

```
handler ( object: :$widget, :$user-option1, ..., :$user-optionN )
```

or

```
handler ( object: :$widget, :$event, :$user-option1, ..., :$user-optionN )
```

The arguments are all optional but to register an event handler, the **:$event** argument must be present.

- $signal-name is the name of the event to be handled. Each gtk widget has its own series of signals, please look for it in the documentation of gtk.
- $connect-flags can be one of G_CONNECT_AFTER or G_CONNECT_SWAPPED. See documentation here.

- %user-options. Any other user data in whatever type. These arguments are provided to the user handler when an event for the handler is fired. There will always be one named argument :$widget which holds the class object on which the signal was registered. The name 'widget' is therefore reserved. An other reserved named argument is of course :$event.

```
# create a class holding a handler method to process a click event
# of a button.
class X {
  method click-handler ( :widget($button), Array :$user-data ) {
    say $user-data.join(' ');
  }
}

# create a button and some data to send with the signal
my GTK::V3::Gtk::GtkButton $button .= new(:label('xyz'));
my Array $data = [<Hello World>];

# register button signal
my X $x .= new(:empty);
$button.register-signal( $x, 'click-handler', 'clicked', :user-data($data));
```