NAME

Glib::Object – Bindings for GObject

DESCRIPTION

GObject is the base object class provided by the gobject library. It provides object properties with a notification system, and emittable signals.

Glib::Object is the corresponding Perl object class. Glib::Objects are represented by blessed hash references, with a magical connection to the underlying C object.

get and set

Some subclasses of `Glib::Object` override `get` and `set` with methods more useful to the subclass, for example `Gtk2::TreeModel` getting and setting row contents.

This is usually done when the subclass has no object properties. Any object properties it or a further subclass does have can always be accessed with `get_property` and `set_property` (together with `find_property` and `list_properties` to enquire about them).

Generic code for any object subclass can use the names `get_property` and `set_property` to be sure of getting the object properties as such.

HIERARCHY

`Glib::Object`

METHODS

object = `$class`**->new (...)**

- ... (list) key/value pairs, property values to set on creation

Instantiate a Glib::Object of type *`$class`*. Any key/value pairs in ... are used to set properties on the new object; see `set`. This is designed to be inherited by Perl-derived subclasses (see Glib::Object::Subclass), but you can actually use it to create any GObject-derived type.

scalar = Glib::Object->new_from_pointer ($pointer, `$noinc`**=FALSE)**

- `$pointer` (unsigned) a C pointer value as an integer.
- `$noinc` (boolean) if true, do not increase the GObject's reference count when creating the Perl wrapper. this typically means that when the Perl wrapper will own the object. in general you don't want to do that, so the default is false.

Create a Perl Glib::Object reference for the C object pointed to by *`$pointer`*. You should need this *very* rarely; it's intended to support foreign objects.

NOTE: the cast from arbitrary integer to GObject may result in a core dump without warning, because the type-checking macro **G_OBJECT()** attempts to dereference the pointer to find a GTypeClass structure, and there is no portable way to validate the pointer.

unsigned = `$object`**->get_data ($key)**

- `$key` (string)

Fetch the integer stored under the object data key *`$key`*. These values do not have types; type conversions must be done manually. See `set_data`.

`$object`**->set_data ($key,** `$data`**)**

- `$key` (string)
- `$data` (scalar)

GObject provides an arbitrary data mechanism that assigns unsigned integers to key names. Functionality overlaps with the hash used as the Perl object instance, so we strongly recommend you use hash keys for your data storage. The GObject data values cannot store type information, so they are not safe to use for anything but integer values, and you really should use this method only if you know what you are doing.

pspec or undef = `$object_or_class_name`**->find_property ($name)**

- `$name` (string)

Find the definition of object property *`$name`* for *`$object_or_class_name`*. Return `undef` if no

such property. For the returned data see Glib::Object::list_properties.

`$object`**->freeze_notify**
Stops emission of ''notify'' signals on *`$object`*. The signals are queued until `thaw_notify` is called on *`$object`*.

`$object`**->get (...)**
- ... (list) list of property names

Alias for `get_property` (see ''get and set'' above).

`$object`**->set (key =>** `$value`**, ...)**
- ... (list) key/value pairs

Alias for `set_property` (see ''get and set'' above).

list = `$object_or_class_name`**->list_properties**
List all the object properties for *`$object_or_class_name`*; returns them as a list of hashes, containing these keys:

name
The name of the property

type
The type of the property

owner_type
The type that owns the property

descr
The description of the property

flags
The Glib::ParamFlags of the property

`$object`**->notify ($property_name)**
- `$property_name` (string)

Emits a ''notify'' signal for the property *`$property`* on *`$object`*.

gpointer = `$object`**->get_pointer**
Complement of `new_from_pointer`.

`$object`**->get_property (...)**
Fetch and return the values for the object properties named in ....

`$object`**->set_property (key =>** `$value`**, ...)**
Set object properties.

unsigned = `$object_or_class_name`**->signal_add_emission_hook ($detailed_signal,** `$hook_func`**,** `$hook_data`**=undef)**
- `$detailed_signal` (string) of the form ''signal−name::detail''
- `$hook_func` (subroutine)
- `$hook_data` (scalar)

Add an emission hook for a signal. The hook will be called for any emission of that signal, independent of the instance. This is possible only for signals which don't have the `G_SIGNAL_NO_HOOKS` flag set.

The *`$hook_func`* should be reference to a subroutine that looks something like this:

```
sub emission_hook {
    my ($invocation_hint, $parameters, $hook_data) = @_;
    # $parameters is a reference to the @_ to be passed to
    # signal handlers, including the instance as $parameters->[0].
    return $stay_connected;  # boolean
}
```

This function returns an id that can be used with `remove_emission_hook`.

Since 1.100.

list = `$instance`**->signal_chain_from_overridden (...)**

- ... (list)

Chain up to an overridden class closure; it is only valid to call this from a class closure override.

Translation: because of various details in how GObjects are implemented, the way to override a virtual method on a GObject is to provide a new ''class closure'', or default handler for a signal. This happens when a class is registered with the type system (see Glib::Type::register and Glib::Object::Subclass). When called from inside such an override, this method runs the overridden class closure. This is equivalent to calling `$self`->SUPER::$method (@_) in normal Perl objects.

unsigned = `$instance`**->signal_connect ($detailed_signal,** `$callback`**,** `$data`**=undef)**

- `$detailed_signal` (string)
- `$callback` (subroutine)
- `$data` (scalar) arbitrary data to be passed to each invocation of *callback*

Register *callback* to be called on each emission of *`$detailed_signal`*. Returns an identifier that may be used to remove this handler with `$object->signal_handler_disconnect`.

unsigned = `$instance`**->signal_connect_after ($detailed_signal,** `$callback`**,** `$data`**=undef)**

- `$detailed_signal` (string)
- `$callback` (scalar)
- `$data` (scalar)

Like `signal_connect`, except that *`$callback`* will be run after the default handler.

unsigned = `$instance`**->signal_connect_swapped ($detailed_signal,** `$callback`**,** `$data`**=undef)**

- `$detailed_signal` (string)
- `$callback` (scalar)
- `$data` (scalar)

Like `signal_connect`, except that *`$data`* and *`$object`* will be swapped on invocation of *`$callback`*.

retval = `$object`**->signal_emit ($name, ...)**

- `$name` (string) the name of the signal
- ... (list) any arguments to pass to handlers.

Emit the signal *name* on *`$object`*. The number and types of additional arguments in ... are determined by the signal; similarly, the presence and type of return value depends on the signal being emitted.

`$ihint` **=** `$instance`**->signal_get_invocation_hint**

Get a reference to a hash describing the innermost signal currently active on `$instance`. Returns undef if no signal emission is active. This invocation hint is the same object passed to signal emission hooks, and contains these keys:

signal_name

The name of the signal being emitted.

detail
: The detail passed on for this emission. For example, a `notify` signal will have the property name as the detail.

run_type
: The current stage of signal emission, one of ''run-first'', ''run-last'', or ''run-cleanup''.

`$object`**->signal_handler_block ($handler_id)**
- `$handler_id` (unsigned)

`$object`**->signal_handler_disconnect ($handler_id)**
- `$handler_id` (unsigned)

boolean = `$object`**->signal_handler_is_connected ($handler_id)**
- `$handler_id` (unsigned)

`$object`**->signal_handler_unblock ($handler_id)**
- `$handler_id` (unsigned)

integer = `$instance`**->signal_handlers_block_by_func ($func,** `$data`**=undef)**
- `$func` (subroutine) function to block
- `$data` (scalar) data to match, ignored if undef

integer = `$instance`**->signal_handlers_disconnect_by_func ($func,** `$data`**=undef)**
- `$func` (subroutine) function to block
- `$data` (scalar) data to match, ignored if undef

integer = `$instance`**->signal_handlers_unblock_by_func ($func,** `$data`**=undef)**
- `$func` (subroutine) function to block
- `$data` (scalar) data to match, ignored if undef

scalar = `$object_or_class_name`**->signal_query ($name)**
- `$name` (string)

Look up information about the signal *`$name`* on the instance type *`$object_or_class_name`*, which may be either a Glib::Object or a package name.

See also `Glib::Type::list_signals`, which returns the same kind of hash refs as this does.

Since 1.080.

`$object_or_class_name`**->signal_remove_emission_hook ($signal_name,** `$hook_id`**)**
- `$signal_name` (string)
- `$hook_id` (unsigned)

Remove a hook that was installed by `add_emission_hook`.

Since 1.100.

`$instance`**->signal_stop_emission_by_name ($detailed_signal)**
- `$detailed_signal` (string)

`$object`**->thaw_notify**
Reverts the effect of a previous call to `freeze_notify`. This causes all queued ''notify'' signals on *`$object`* to be emitted.

boolean = Glib::Object->set_threadsafe ($threadsafe)
- `$threadsafe` (boolean)

Enables/disables threadsafe gobject tracking. Returns whether or not tracking will be successful and thus whether using perl ithreads will be possible.

`$object`**->tie_properties ($all=FALSE)**
- `$all` (boolean) if FALSE (or omitted) tie only properties for this object's class, if TRUE tie the properties of this and all parent classes.

A special method available to Glib::Object derivatives, it uses perl's tie facilities to associate hash keys with the properties of the object. For example:

```
$button->tie_properties;
# equivilent to $button->set (label => 'Hello World');
$button->{label} = 'Hello World';
print "the label is: ".$button->{label}."\n";
```

Attempts to write to read-only properties will croak, reading a write-only property will return '[write−only]'.

Care must be taken when using tie_properties with objects of types created with Glib::Object::Subclass as there may be clashes with existing hash keys that could cause infinite loops. The solution is to use custom property get/set functions to alter the storage locations of the properties.

SIGNALS

notify (Glib::Object, Glib::ParamSpec)

SEE ALSO

Glib

COPYRIGHT