The General Toolkit

Version 1.0
April 1997

by Peter Mattis

The General Toolkit

1 Copying

GTK is *free*; this means that everyone is free to use it and free to redestribute it on a free basis. GTK is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GTK that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of GTK, that you receive source code or else can get it if you want it, that you can change GTK or use pieces of it in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of GTK, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for my own protection, we must make certain that everyone finds out that there is no warranty for GTK. If GTK is modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will no reflect on our reputation.

The precise conditions of the licenses for GTK are found in the General Public Licenses that accompanies it.

2 What is GTK?

GTK is a library for creating graphical user interfaces similar to the Motif "look and feel". It is designed to be small and efficient, but still flexible enough to allow the programmer freedom in the interfaces created. GTK allows the programmer to use a variety of standard user interface widgets (see ⟨undefined⟩ [Widgets], page ⟨undefined⟩) such as push, radio and check buttons, menus, lists and frames. It also provides several "container" widgets which can be used to control the layout of the user interface elements.

GTK provides some unique features. (At least, I know of no other widget library which provides them). For example, a button does not contain a label, it contains a child widget, which in most instances will be a label. However, the child widget can also be a pixmap, image or any combination possible the programmer desires. This flexibility is adhered to throughout the library.

3 Object Overview

GTK implements a semi-simple class mechanism and an associated class hierarchy for widgets and several other useful objects. The GtkObject type is the root of the class hierarchy. It provides a few items needed by all classes, the foundation for the signal (see ⟨undefined⟩ [Signals], page ⟨undefined⟩) mechanism and the "destroy" method.

The class hierarchy is defined by a type hierarchy. This hierarchy allows queries to be made in regards to a type. The basic query that can be performed is asking whether a given type has an "is a" relation with another type. For instance, it is common to ask whether a general widget pointer is a type of specific widget so that runtime sanity checks can be made.

3.1 Type utility functions

The `GtkTypeInfo` structure is used to communicate information to `gtk_type_unique` as opposed to passing in large numbers of parameters.

```
typedef struct _GtkTypeInfo GtkTypeInfo;

struct _GtkTypeInfo
{
  gchar *type_name;
  guint object_size;
  guint class_size;
  GtkClassInitFunc class_init_func;
  GtkObjectInitFunc object_init_func;
  GtkValueInitFunc value_init_func;
}
```

- The `type_name` field refers to the name of the type. It is convention for the type name to be the same as the C structure type. For example, the type name of the `GtkObject` structure is "GtkObject".
- The `object_size` field refers to the size in bytes of the C structure. The easiest (and portable) means of computing this size is by using the C `sizeof` operator. For instance, the sizeof of the `GtkObject` structure is computed by doing `sizeof (GtkObject)`.
- The `class_size` field refers to the size in bytes of the C structure for the class. Again, the `sizeof` operator should be used to compute this value.
- The `class_init_func` field is a callback which is used by the type mechanism to initialize class specific fields. The single argument this function takes is a pointer to a class structure.
- The `object_init_func` field is a callback which is used by the type mechanism to initialize object specific fields. The single argument this functions takes is a pointer to an object structure.
- The `value_init_func` field is a callback which is used by the type mechanism to initialize object stack value types. (FIXME: unfinished).

`guint` **gtk_type_unique** (`guint` *parent_type*, `GtkTypeInfo` `*`*type_info*) Function

The *parent_type* is simply the value of the new types parent type. If *parent_type* is 0, then the new type is the root of the type hierarchy. *type_info* is a pointer to a structure which contains necessary information for construction of the new type. Specifically, the `type_name`, `object_size` and `class_size` fields are required. The `class_init_func`, `object_init_func` and `value_init_func` fields may be NULL.

`gchar*` **gtk_type_name** (`guint` *type*) Function

The returned string is the name of *type* as specified to `gtk_type_unique`.

`guint` **gtk_type_from_name** (`guchar` `*`*name*) Function

Return the type associated with *name*. If there is no type associated with *name*, then 0 will be returned.

`guint` **gtk_type_parent** (`guint` *type*) Function

Returns the parent type of *type* or 0 if *type* is the root of the type hierarchy.

`gpointer` **gtk_type_class** (`guint` *type*) Function

Returns the initialized class structure for *type*. The class structure is actually created and initialized the first time it is needed. If creation and initialization occurs, the `class_size` field of the `GtkTypeInfo` structure used to initialize this type is used to determine how large the class structure is. The `class_init_func` field from the `GtkTypeInfo` structure is called for all the members in the types ancestry, including the type. The order of this invocation proceeds from the root on down. For example, the `GtkWidgetClass` is first initialized as an `GtkObjectClass` by the object class initialization routine and then by the widget class initialization routine. This allows the widget class initialization routine to override values set by the object class initialization routine. The returned structure is shared by all objects of *type* and, as such, should not be modified.

`gpointer` **gtk_type_new** (`guint` *type*) Function

Returns a new instance of an *type* object. The object structure is created and initialized similarly to the class structure (as described above). The `object_size` and `object_init_func` fields of the `GtkTypeInfo` structure are used to determine the objects allocated size and the object specific initialization routine. Similarly to the class initialization, all the object initialization routines from the root on down to the particular type being created are invoked.

`void` **gtk_type_describe_heritage** (`guint` *type*) Function

Prints the type heritage for *type*. The heritage for a type includes the type and all its parent types up the type tree.

`void` **gtk_type_describe_tree** (`guint` *type*, `gint` *show_size*) Function

Prints the type tree which starts at *type*. *show_size* is a boolean which determines whether type sizes are printed.

`gint` **gtk_type_is_a** (`guint` *type*, `guint` *is_a_type*) Function
A predicate function which determines whether the relation *type* is_a *is_a_type* is true.

3.2 Object functions

The GtkObject type is the root of the type hierarchy used by GTK. It provides a minimal set of fields used to implement the actual object, class and signal mechanisms, as well as several utility routines which make dealing with objects easier.

For the adventurous, see ⟨undefined⟩ [Object Implementation], page ⟨undefined⟩.

`guint` **gtk_object_get_type** (`void`) Function
Returns the `GtkObject` type identifier.

`void` **gtk_object_class_add_signals** (`GtkObjectClass` **class*, `gint` **signals*, `gint` *nsignals*) Function
Adds *signals* to the `signals` field in the GtkObjectClass structure *class*. See ⟨undefined⟩ [Signals], page ⟨undefined⟩.

`void` **gtk_object_destroy** (`GtkObject` **object*) Function
Performs checks to make sure it is alright to destroy *object* and then emits the `destroy` signal. The check which is performed is to make sure *object* is not already processing another signal. If this were the case then destroying the object immediately would undoubtedly cause problems as the other signal would not be able to tell the object was destroyed. The solution is that if *object* is processing another signal we mark *object* is needing to be destroyed. When we finish processing of the other signal we check whether the object needs to be destroyed.

The GtkObject type provides a mechanism for associating arbitrary amounts of data with an object. The data is associated with the object using a character string key. The functions `gtk_object_set_data`, `gtk_object_get_data`, and `gtk_object_remove_data` are the interface to this mechanism. Two other routines, `gtk_object_set_user_data` and `gtk_object_get_user_data`, exist as convenience functions which simply use the same mechanism.

`void` **gtk_object_set_data** (`GtkObject` **object*, `const char` **key*, `gpointer` *data*) Function
Associate *data* with *key* in the data list of *object*.

`gpointer` **gtk_object_get_data** (`GtkObject` **object*, `const char` **key*) Function
Retrieve the data associated with *key* in the data list of *object*.

`void` **gtk_object_remove_data** (`GtkObject` **object*, `const char` **key*) Function
Remove the data associated with *key* in the data list of *object*.

`void` **gtk_object_set_user_data** (`GtkObject` **object*, `gpointer` *data*) Function

Sets *data* into the `user_data` field of *object*.

`gpointer` **gtk_object_get_user_data** (`GtkObject` **object*) Function

Returns the `user_data` field of *object*.

The GtkObject type also provides a mechanism for specifying initialization values for fields. This general mechanism is called object value stacks. The reason for using value stacks is that they can simplify the life of the programmer. For instance, by default widgets are non-visible when created. However, the "visible" value for widgets may be specified so that widgets are made visible when created. (FIXME: unfinished).

`void` **gtk_object_value_stack_new** (`guint` *object_type*, `const gchar` **value_id*, `GtkParamType` *value_type*) Function

`void` **gtk_object_push_value** (`guint` *object_type*, `const gchar` **value_id*, ...) Function

Push a value on the value stack specified by *object_type* and *value_id*. The type of value is implicitly given in the context of *object_type* and *value_id*. (That is, it is not specified explicitly in the function call). Only a single extra argument is expected which is the data which is to be placed on the stack.

`void` **gtk_object_pop_value** (`guint` *object_type*, `const gchar` **value_id*) Function

Pop a value of the value stack specified by *object_type* and *value_id*.

`gint` **gtk_object_peek_value** (`guint` *object_type*, `const gchar` **value_id*, `gpointer` *data*) Function

Peek at the value on the top of the value stack specified by *object_type* and *value_id*. The *data* argument is interpreted as the location of where to place the "peeked" data. For instance, if the peeked data is of type `GTK_PARAM_POINTER`, then *data* will be a pointer to a pointer. If the value stack is empty or does not exist or an error occurs, `gtk_object_peek_value` will return `FALSE`. On success it will return `TRUE`.

4 Signals Overview

Signals are GTK's method for objects to perform callbacks. A signal is an event which occurs upon an object. The programmer can connect to a signal of an object which involves specifying a function to be called when that signal is emitted in the specified object.

When a signal is emitted, both the class function associated with the signal (when it was defined) and all signal handlers installed for that signal on the particular object emitting the signal are called. The widget programmer can specify whether the class function is to be called before after or both before and after the signal handlers installed by the widget user. The widget user can, however, specify that their signal handler is to be run after the class function (using the "_after" signal connection routines). Any signal handling function can emit the same signal on the same object while it is running causing that signal emittion to either restart or to run recursively. Additionally, signal emittion can be terminated prematurely. While both such abilities are rarely used, they do allow for greater flexibility in regards to signals. For instance, a programmer can attach to the key press event signal and intercept all tab key presses from a widget. This particular example is used in the file selection dialog to implement tab completion of filenames and prevent the entry widget from inserting the tab into its buffer.

Signals are selected using either an integer identifier or a character string name. It is convention to name the signal the same as the class function which is associated with it. There are two versions of most of the signal functions, one which takes an integer identifier and one which takes a character string name for the signal.

`gint` **gtk_signal_new** (`gchar *`*name*`, GtkSignalRunType` *run_type*`, gint` *object_type*`, gint` *function_offset*`, GtkSignalMarsahller` *marshaller*`, GtkParamType` *return_val*`, gint` *nparams*`, ...`) Function

Create a new signal and give it the character string identifier *name*. *name* needs to be unique in the context of *object_type*'s branch of the class hierarchy. That is, *object_type* cannot create a signal type with the same name as a signal type created by one of its parent types.

run_type specifies whether the class function should be run before (`GTK_RUN_FIRST`), after (`GTK_RUN_LAST`) or both before and after normal signal handlers (`GTK_RUN_BOTH`). Additionally, the `GTK_RUN_NO_RECURSE` value can be or'ed with any of those values to specify that the signal should not be recursive. By default, emitting the same signal on the same widget will cause the signal to be emitted twice. However, if the `GTK_RUN_NO_RECURSE` flag is specified, emitting the same signal on the same widget will cause the current signal emittion to be restarted. This allows the widget programmer to specify the semantics of signal emittion on a per signal basis. (The `GTK_RUN_NO_RECURSE` flag is used by the GtkAdjustment widget).

The *function_offset* is the byte offset from the start of the class structure to the class function field within the class structure. The easiest means to compute this offset is by using the `GTK_SIGNAL_OFFSET` macro which takes the class structure type as the first argument and the field as the second argument. For example, `GTK_SIGNAL_OFFSET (GtkObjectClass, destroy)` will give the offset

of the `destroy` class function within the `GtkObjectClass`. Note: An offset is specified instead of an absolute location since there will be multiple instances of a class structure being referenced. (The `GtkWidgetClass` structure "is a" `GtkObjectClass` structure, etc.)

The *marshaller* function is used to invoke a signal handler. Since signal handlers may take different parameters and return values and a general mechanism for invoking them is not apparent, the approach of making the signal creator responsible for invoking the signal handler was taken. (FIXME: unfinished).

The *return_val* and *nparams* and the remaining arguments specify the return value and the arguments to the signal handler respectively. Note: There is an implicit first argument to every signal handler which is the widget the signal has been emitted from. The variable argument list (...) specifies the types of the arguments. These can be one of `GTK_PARAM_CHAR`, `GTK_PARAM_SHORT`, `GTK_PARAM_INT`, `GTK_PARAM_LONG`, `GTK_PARAM_POINTER` or `GTK_PARAM_FUNCTION`. It is undefined to specify `GTK_PARAM_NONE` as an argument type, however it is ok to use `GTK_PARAM_NONE` for *return_val*. (This corresponds to returning a `void`).

`gtk_signal_new` returns the integer identifier of the newly created signal. Signal identifiers start numbering at 1 and increase upwards. A value of -1 will be returned if an error occurs.

Note: `gtk_signal_new` is only needed by widget writers. A normal user of GTK will never needed to invoke this function.

`gint` **gtk_signal_lookup** (`gchar *`*name*`, gint` *object_type*) Function

Returns the integer identifier for the signal referenced by *name* and *object_type*. If *object_type* does not define the signal *name*, then the signal is looked for in *object_type*'s parent type recursively.

`gint` **gtk_signal_emit** (`GtkObject *`*object*`, gint` *signal_type*`,` ...) Function

Emit the signal specified by the integer identifier *signal_type* from *object*. If an error occurs, `gtk_signal_emit` will return `FALSE` and will return `TRUE` on success. The signal definition determines the parameters passed in the variable argument list (...). For example, if the signal is defined as:

```
gint (* event) (GtkWidget *widget, GdkEvent *event);
```

Then a call to emit the "event" signal would look like:

```
GdkEvent event;
gint return_val;
...
gtk_signal_emit (some_object,
                 gtk_signal_lookup ("event",
                   GTK_OBJECT_TYPE (some_object)),
                 &event, &return_val);
```

Notice that the `widget` argument is implicit in that the first argument to every signal is a type derived from `GtkObject`. The *return_val* argument is actually a pointer to the return value type since the signal mechanism needs to be

able to place the return value in an actual location. And lastly, the `gtk_signal_lookup` call is normally avoided by using the `gtk_signal_emit_by_name` function instead. `gtk_signal_emit` is normally used internally by widgets which know the signal identifier (since they defined the signal) and can therefore side-step the cost of calling `gtk_signal_lookup`.

`gint` **gtk_signal_emit_by_name** (`GtkObject` **object*, `gchar` **name*, ...) Function

Similar to `gtk_signal_emit` except that the signal is referenced by *name* instead of by its integer identifier.

`void` **gtk_signal_emit_stop** (`GtkObject` **object*, `gint` *signal_type*) Function

Stop the emission of the signal *signal_type* on *object*. *signal_type* is the integer identifier for the signal and can be determined using the function `gtk_signal_lookup`. Alternatively, the function `gtk_signal_emit_stop_by_name` can be used to refer to the signal by name. Attempting to stop the emission of a signal that isn't being emitted does nothing.

`void` **gtk_signal_emit_stop_by_name** (`GtkObject` **object*, `gchar` **name*) Function

Similar to `gtk_signal_emit_stop` except that the signal is referenced by *name* instead of by its integer identifier.

`gint` **gtk_signal_connect** (`GtkObject` **object*, `gchar` **name*, `GtkSignalFunc` *func*, `gpointer` *func_data*) Function

Connects a signal handling function to a signal emitting object. *func* is connected to the signal *name* emitted by *object*. The arguments and returns type of *func* should match the arguments and return type of the signal *name*. However, *func* may take the extra argument of *func_data*. Due to the C calling convention it is ok to ignore the extra argument. (It is ok to ignore all the arguments in fact).

`gtk_signal_connect` returns an integer identifier for the connection which can be used to refer to it in the future. Specifically it is useful for removing the connection and/or blocking it from being used.

`gint` **gtk_signal_connect_after** (`GtkObject` **object*, `gchar` **name*, `GtkSignalFunc` *func*, `gpointer` *func_data*) Function

Similar to `gtk_signal_connect` except the signal handler is connected in the "after" slot. This allows a signal handler to be guaranteed to run after other signal handlers connected to the same signal on the same object and after the class function associated with the signal.

Like `gtk_signal_connect`, `gtk_signal_connect_after` returns an integer identifier which can be used to refer to the connection.

`gint` **gtk_signal_connect_object** (`GtkObject` **object*, `gchar` **name*, `GtkSignalFunc` *func*, `GtkObject` **slot_object*) Function

Connects *func* to the signal *name* emitted by *object*. Similar to `gtk_signal_connect` with the difference that *slot_object* is passed as the first parameter to

func instead of the signal emitting object. This can be useful for connecting a signal emitted by one object to a signal in another object. A common usage is to connect the "destroy" signal of dialog to the "clicked" signal emitted by a "close" button in the dialog. That is, the "clicked" signal emitted by the button will caused the "destroy" signal to be emitted for the dialog. This is also the "right" way to handle closing of a dialog since the "destroy" signal will be sent if the dialog is deleted using a window manager function and this enables the two methods of closing the window to be handled by the same mechanism. Returns an integer identifier which can be used to refer to the connection.

`gint` **gtk_signal_connect_object_after** (`GtkObject *`*object*`,` `gchar *`*name*`,` `GtkSignalFunc` *func*`,` `GtkObject *`*slot_object*) Function

Similar to `gtk_signal_connect_object` except the signal handler is connected in the "after" slot. This allows a signal handler to be guaranteed to run after other signal handlers connected to the same signal on the same object and after the class function associated with the signal. Returns an integer identifier which can be used to refer to the connection.

`void` **gtk_signal_disconnect** (`GtkObject *`*object*`,` `gint` *id*) Function

Disconnects a signal handler from an object. The signal handler is identified by the integer *id* which is returned by the `gtk_signal_connect*` family of functions.

`void` **gtk_signal_disconnect_by_data** (`GtkObject *`*object*`,` `gpointer` *data*) Function

Disconnects a signal handler from an object. The signal handler is identified by the *data* argument specified as the *func_data* argument to the `gtk_signal_connect*` family of functions. For the `gtk_signal_connect_object*` functions, *data* refers to the *slot_object*.

Note: This will remove all signal handlers connected to *object* which were connected using *data* as their *func_data* argument. Multiple signal handlers may be disconnected with this call.

`void` **gtk_signal_handler_block** (`GtkObject *`*object*`,` `gint` *id*) Function

Blocks calling of a signal handler during signal emission. The signal handler is identified by the integer *id* which is returned by the `gtk_signal_connect*` family of functions. If the signal is already blocked no change is made.

`void` **gtk_signal_handler_block_by_data** (`GtkObject *`*object*`,` `gint` *data*) Function

Blocks calling of a signal handler during signal emission. The signal handler is identified by the *data* argument specified as the *func_data* argument to the `gtk_signal_connect*` family of functions. For the `gtk_signal_connect_object*` functions, *data* refers to the *slot_object*. If the signal is already blocked no change is made.

Note: This will block all signal handlers connected to *object* which were connected using *data* as their *func_data* argument. Multiple signal handlers may be blocked with this call.

`void` **gtk_signal_handler_unblock** (`GtkObject` **object*, `gint` *id*) Function

Unblocks calling of a signal handler during signal emission. The signal handler is identified by the integer *id* which is returned by the `gtk_signal_connect*` family of functions. If the signal is already unblocked no change is made.

`void` **gtk_signal_handler_unblock_by_data** (`GtkObject` **object*, `gint` *data*) Function

Unblocks calling of a signal handler during signal emission. The signal handler is identified by the *data* argument specified as the *func_data* argument to the `gtk_signal_connect*` family of functions. For the `gtk_signal_connect_object*` functions, *data* refers to the *slot_object*. If the signal is already unblocked no change is made.

Note: This will unblock all signal handlers connected to *object* which were connected using *data* as their *func_data* argument. Multiple signal handlers may be unblocked with this call.

`void` **gtk_signal_handlers_destroy** (`GtkObject` **object*) Function

Destroy all of the signal handlers connected to *object*. There should normally never be reason to call this function as it is called automatically when *object* is destroyed.

`void` **gtk_signal_default_marshaller** (`GtkObject` **object*, `GtkSignalFunc` *func*, `gpointer` *func_data*, `GtkSignalParam` **params*) Function

`gtk_signal_new` requires a callback in order to actually call a signal handler for a particular signal. The vast majority of signals are of the particular form:

```
(* std_signal) (gpointer std_arg);
```

`gtk_signal_default_marshaller` is a signal marshaller which marshals arguments for a signal of that form.

5 Widget Overview

Widgets are the general term used to describe user interface objects. A widget defines a class interface that all user interface objects conform to. This interface allows a uniform method for dealing with operations common to all objects such as hiding and showing, size requisition and allocation and events.

The common interface that widgets must adhere to is described by the GtkWidget and GtkWidgetClass structure. For the purposes of using GTK these structures can be considered read-only and, for the most part, opaque.

All widget creation routines in GTK return pointers to GtkWidget structures. In reality, all widget creation routines create structures that can be viewed as equivalent to the GtkWidget structure, but often have contain additional information. See ⟨undefined⟩ [Object Implementation], page ⟨undefined⟩

The widgets available for use are implemented in a hierarchy. Several widgets exist solely as common bases for more specific widgets. For example, it is not possible to create a ruler widget itself, but the ruler widget provides a base and functionality common to the horizontal and vertical rulers.

The available widgets (in alphabetical order):

5.1 The alignment widget

5.1.1 Description

The alignment widget is a container (see ⟨undefined⟩ [GtkContainer], page ⟨undefined⟩) derived from the bin widget (see ⟨undefined⟩ [GtkBin], page ⟨undefined⟩). Its entire purpose is to give the programmer flexibility in how the child it manages is positioned when a window is resized.

Normally, a widget is allocated at least as much size as it requests. (see ⟨undefined⟩ [GtkContainer], page ⟨undefined⟩ for a discussion of geometry management). When a widget is allocated more size than it requests there is a question of how the widget should expand. By convention, most GTK widgets expand to fill their allocated space. Sometimes this behavior is not desired. The alignment widget allows the programmer to specify how a widget should expand and position itself to fill the area it is allocated.

5.1.2 Options

xscale User Option
yscale User Option

The *xscale* and *yscale* options specify how to scale the child widget. If the scale value is 0.0, the child widget is allocated exactly the size it requested in that dimension. If the scale value is 1.0, the child widget is allocated all of the space in a dimension. A scale value of 1.0 for both x and y is equivalent to not using an alignment widget.

xalign User Option
yalign User Option

The *xalign* and *yalign* options specify how to position the child widget when it is not allocated all the space available to it (because the *xscale* and/or *yscale* options are less than 1.0). If an alignment value is 0.0 the widget is positioned to the left (or top) of its allocated space. An alignment value of 1.0 positions the widget to the right (or bottom) of its allocated space. A common usage is to specify *xalign* and *yalign* to be 0.5 which causes the widget to be centered within its allocated area.

5.1.3 Signals

5.1.4 Functions

`guint` **gtk_alignment_get_type** (`void`) Function

Returns the `GtkAlignment` type identifier.

`GtkWidget*` **gtk_alignment_new** (`gfloat` *xalign*, `gfloat` *yalign*, `gfloat` *xscale*, `gfloat` *yscale*) Function

Create a new `GtkAlignment` object and initialize it with the values *xalign*, *yalign*, *xscale* and *yscale*. The new widget is returned as a pointer to a `GtkWidget` object. `NULL` is returned on failure.

`void` **gtk_alignment_set** (`GtkAlignment *`*alignment*, `gfloat` *xalign*, `gfloat` *yalign*, `gfloat` *xscale*, `gfloat` *yscale*) Function

Set the *xalign*, *yalign*, *xscale* and *yscale* options of an alignment widget. It is important to not set the fields of the `GtkAlignment` structure directly (or, for that matter, any type derived from `GtkObject`).

Alignment, ALIGNMENT

5.2 The arrow widget

5.2.1 Description

The arrow widget is derived from the misc widget (see ⟨undefined⟩ [GtkMisc], page ⟨undefined⟩) and is intended for use where a directional arrow (in one of the four cardinal directions) is desired. As such, it has very limited functionality and basically only draws itself in a particular direction and with a particular shadow type. The arrow widget will expand to fill all the space it is allocated.

5.2.2 Options

arrow_type User Option

The *arrow_type* option specifies which direction the arrow will point. It can be one of `GTK_ARROW_UP`, `GTK_ARROW_DOWN`, `GTK_ARROW_LEFT` or `GTK_ARROW_RIGHT`.

shadow_type User Option

The *shadow_type* option specifies how to draw the shadow for the arrow. Currently, only the `GTK_SHADOW_IN` and `GTK_SHADOW_OUT` shadow types are supported for drawing arrows. Other shadow types will cause nothing to be drawn.

5.2.3 Signals

5.2.4 Functions

`guint` **gtk_arrow_get_type** (`void`) Function

Returns the `GtkArrow` type identifier.

`GtkWidget*` **gtk_arrow_new** (`GtkArrowType` *arrow_type*, `GtkShadowType` *shadow_type*) Function

Create a new `GtkArrow` object and initialize it with the values *arrow_type* and *shadow_type*. The new widget is returned as a pointer to a `GtkWidget` object. `NULL` is returned on failure.

`void` **gtk_arrow_set** (`GtkArrow *`*arrow*`, GtkArrowType` *arrow_type*`, GtkShadowType` *shadow_type*) Function

Set the *arrow_type* and *shadow_type* options of an arrow widget. It is important to not set the fields of the `GtkArrow` structure directly (or, for that matter, any type derived from `GtkObject`).

Arrow, ARROW

5.3 The bin widget

5.3.1 Description

The bin widget is a container (see ⟨undefined⟩ [GtkContainer], page ⟨undefined⟩) derived from the container widget. It is an abstract base class. That is, it is not possible to create an actual bin widget. It exists only to provide a base of functionality for other widgets. Specifically, the bin widget provides a base for several other widgets that contain only a single child. These widgets include alignments (see ⟨undefined⟩ [GtkAlignment], page ⟨undefined⟩), frames (see ⟨undefined⟩ [GtkFrame], page ⟨undefined⟩), items (see ⟨undefined⟩ [GtkItem], page ⟨undefined⟩), viewports (see ⟨undefined⟩ [GtkViewport], page ⟨undefined⟩) and windows (see ⟨undefined⟩ [GtkWindow], page ⟨undefined⟩)

5.3.2 Signals

5.3.3 Functions

`guint` **gtk_bin_get_type** (`void`) Function

Returns the `GtkBin` type identifier.

Bin, BIN

5.4 The box widget

5.4.1 Description

The box widget is a container (see ⟨undefined⟩ [GtkContainer], page ⟨undefined⟩) derived from the container widget. It is an abstract base class used by the horizontal box (see ⟨undefined⟩ [GtkHBox], page ⟨undefined⟩) and vertical box (see ⟨undefined⟩ [GtkVBox], page ⟨undefined⟩) widgets to provide a base of common functionality.

A box provides an abstraction for organizing the position and size of widgets. Widgets in a box are layed out horizontally or vertically. By using a box widget appropriately, a programmer can control how widgets are positioned and how they will be allocated space when a window gets resized.

The key attribute of boxes is that they position their children in a single row (horizontal boxes) or column (vertical boxes). In the case of horizontal boxes, all children are stretched vertically. The vertical size of the box is determined by the largest vertical requisition of all of its children. Similarly, a vertical box streches all of its children horizontally. The horizontal size (of the vertical box) is determined by the largest horizontal requisition of all of its children. An alignment widget (see ⟨undefined⟩ [GtkAlignment], page ⟨undefined⟩) can be used to control child allocation more precisely on a per child basis.

The second attribute of boxes is how they expand children. In the case of a horizontal box, the main control is over how children are expanded horizontally to fill the allocated area. (The rest of this discussion will focus on horizontal boxes but it applies to vertical boxes as well).

There are two flags which can be set controlling how a widget is expanded horizontally in a horizontal box. These are the `expand` and `fill`. There operation is fairly simple. If `expand` is set, the childs potentially allocated area will expand to fill available space. If `fill` is set, the childs actual allocated area will be its potentially allocated area. There is a difference between the potentially area (which is the area the box widget sets aside for the child) and the actual allocated area (which is the area the box widget actual allocates for the widget via `gtk_widget_size_allocate`).

The allocation of space to children occurs as follows (for horizontal boxes):

1. All children are allocated at least their requested size horizontally and the maximum requested child size vertically.
2. Any child with the `expand` flag set is allocated `extra_width / nexpand_children` extra pixels horizontally. If the `homogeneous` flag was set, all children are considered to have the `expand` flag set. That is, all children will be allocated the same area.The horizontal box is a fair widget and, as such, divides up any extra allocated space evenly among the "expand" children. (Those children which have the `expand` flag set). The exception occurs when `extra_width / nexpand_children` does not divide cleanly. The extra space is given to the last widget.
3. `spacing` number of pixels separate each child. Note: The separation is between the potentially allocated area for each child and not the actual allocated area. The `padding` value associated with each child causes that many pixels to be left empty to each side of the child.

4. If a child has the `fill` flag set it is allocated its potentially allocated area. If it does not, it is allocated its requested size horizontally and centered within its potentially allocated area. Its vertical allocation is still the maximum requested size of any child.
5. Children placed at the start of the box are placed in order of addition to the box from left to right in the boxes allocated area.. Children placed at the end of the box are placed in order of addition from right to left in the boxes allocated area.

See ⟨undefined⟩ [GtkHBox], page ⟨undefined⟩, and ⟨undefined⟩ [GtkVBox], page ⟨undefined⟩, for code examples of using horizontal and vertical boxes.

5.4.2 Options

5.4.3 Signals

5.4.4 Functions

`guint` **gtk_box_get_type** (`void`) Function

Returns the `GtkBox` type identifier.

`void` **gtk_box_pack_start** (`GtkBox` *_box_, `GtkWidget` *_child_, `gint` _expand_, `gint` _fill_, `gint` _padding_) Function

Add _child_ to the front of _box_. The flags _expand_ and _fill_ and the padding value of _padding_ are associated with _child_.

`void` **gtk_box_pack_end** (`GtkBox` *_box_, `GtkWidget` *_child_, `gint` _expand_, `gint` _fill_, `gint` _padding_) Function

Add _child_ to the end of _box_. The flags _expand_ and _fill_ and the padding value of _padding_ are associated with _child_.

`void` **gtk_box_pack_start_defaults** (`GtkBox` *_box_, `GtkWidget` *_widget_) Function

A convenience function which is equivalent to the following:

```
gtk_box_pack_start (box, widget, TRUE, TRUE, 0);
```

`void` **gtk_box_pack_end_defaults** (`GtkBox` *_box_, `GtkWidget` *_widget_) Function

A convenience function which is equivalent to the following:

```
gtk_box_pack_start (box, widget, TRUE, TRUE, 0);
```

Box, BOX

5.5 The button widget

5.5.1 Description

5.5.2 Signals

`void` **GtkButton::pressed** (`GtkButton *`*button*) Signal

`void` **GtkButton::released** (`GtkButton *`*button*) Signal

`void` **GtkButton::clicked** (`GtkButton *`*button*) Signal

`void` **GtkButton::enter** (`GtkButton *`*button*) Signal

`void` **GtkButton::leave** (`GtkButton *`*button*) Signal

5.5.3 Functions

`guint` **gtk_button_get_type** (`void`) Function

`GtkWidget*` **gtk_button_new** (`void`) Function

`GtkWidget*` **gtk_button_new_with_label** (`gchar *`*label*) Function

`void` **gtk_button_pressed** (`GtkButton *`*button*) Function

`void` **gtk_button_released** (`GtkButton *`*button*) Function

`void` **gtk_button_clicked** (`GtkButton *`*button*) Function

`void` **gtk_button_enter** (`GtkButton *`*button*) Function

`void` **gtk_button_leave** (`GtkButton *`*button*) Function

Button, BUTTON

5.6 The check button widget

5.6.1 Description

5.6.2 Signals

5.6.3 Functions

`guint` **gtk_check_button_get_type** (`void`) Function

`GtkWidget*` **gtk_check_button_new** (`void`) Function

`GtkWidget*` **gtk_check_button_new_with_label** (`gchar *`*label*) Function

`GtkCheckButton*` **GTK_CHECK_BUTTON** (`gpointer` *obj*) Function

`GtkCheckButtonClass*` **GTK_CHECK_BUTTON_CLASS** (`gpointer` *class*) Function

`gint` **GTK_IS_CHECK_BUTTON** (`gpointer` *obj*) Function

CheckButton, CHECK_BUTTON

5.7 The check menu item widget

5.7.1 Description

5.7.2 Signals

`void` **GtkCheckMenuItem::toggled** (`GtkCheckMenuItem` **check_menu_item*) Signal

5.7.3 Functions

`guint` **gtk_check_menu_item_get_type** (`void`) Function

`GtkWidget*` **gtk_check_menu_item_new** (`void`) Function

`GtkWidget*` **gtk_check_menu_item_new_with_label** (`gchar` **label*) Function

`void` **gtk_check_menu_item_set_state** (`GtkCheckMenuItem` **check_menu_item*, `gint` *state*) Function

`void` **gtk_check_menu_item_toggled** (`GtkCheckMenuItem` **check_menu_item*) Function

CheckMenuItem, CHECK_MENU_ITEM

5.8 The container widget

5.8.1 Description

5.8.2 Signals

`void` **GtkContainer::add** (`GtkContainer` **container*, `GtkWidget` **widget*) — Signal

`void` **GtkContainer::remove** (`GtkContainer` **container*, `GtkWidget` **widget*) — Signal

`void` **GtkContainer::need_resize** (`GtkContainer` **container*, `GtkWidget` **widget*) — Signal

`void` **GtkContainer::foreach** (`GtkContainer` **container*, `GtkCallback` *callback*, `gpointer` *callback_data*) — Signal

`gint` **GtkContainer::focus** (`GtkContainer` **container*, `GtkDirectionType` *direction*) — Signal

5.8.3 Functions

`guint` **gtk_container_get_type** (`void`) — Function

`void` **gtk_container_border_width** (`GtkContainer` **container*, `gint` *border_width*) — Function

`void` **gtk_container_add** (`GtkContainer` **container*, `GtkWidget` **widget*) — Function

`void` **gtk_container_remove** (`GtkContainer` **container*, `GtkWidget` **widget*) — Function

`void` **gtk_container_disable_resize** (`GtkContainer` **container*) — Function

`void` **gtk_container_enable_resize** (`GtkContainer` **container*) — Function

`void` **gtk_container_block_resize** (`GtkContainer` **container*) — Function

`void` **gtk_container_unblock_resize** (`GtkContainer` **container*) — Function

`gint` **gtk_container_need_resize** (`GtkContainer` **container*, `GtkWidget` **widget*) — Function

`void` **gtk_container_check_resize** (`GtkContainer` **container*, `GtkWidget` **widget*) — Function

`void` **gtk_container_foreach** (`GtkContainer` **container*, `GtkCallback` *callback*, `gpointer` *callback_data*) — Function

`void` **gtk_container_focus** (`GtkContainer` **container*, `GtkDirectionType` *direction*) Function

`GList*` **gtk_container_children** (`GtkContainer` *container*) Function

Container, CONTAINER

5.9 The dialog widget

5.9.1 Description

5.9.2 Signals

5.9.3 Functions

`guint` **gtk_dialog_get_type** (`void`) Function

`GtkWidget*` **gtk_dialog_new** (`void`) Function

Dialog, DIALOG

5.10 The drawing area widget

5.10.1 Description

5.10.2 Signals

5.10.3 Functions

`guint` **gtk_drawing_area_get_type** (`void`) Function

`GtkWidget*` **gtk_drawing_area_new** (`void`) Function

`void` **gtk_drawing_area_size** (`GtkDrawingArea` **darea*, `gint` *width*, `gint` *height*) Function

DrawingArea, DRAWING_AREA

5.11 The entry widget

5.11.1 Description

5.11.2 Signals

`void` **GtkEntry::insert** (`GtkEntry` **entry*, `gchar` **text*, `gint` *length*, `gint` **position*) Signal

`void` **GtkEntry::delete** (`GtkEntry` **entry*, `gint` *start_pos*, `gint` *end_pos*) Signal

`void` **GtkEntry::changed** (`GtkEntry` **entry*) Signal

5.11.3 Functions

`guint` **gtk_entry_get_type** (`void`) Function

`GtkWidget*` **gtk_entry_new** (`void`) Function

`void` **gtk_entry_set_text** (`GtkEntry` **entry*, `gchar` **text*) Function

`void` **gtk_entry_append_text** (`GtkEntry` **entry*, `gchar` **text*) Function

`void` **gtk_entry_prepend_text** (`GtkEntry` **entry*, `gchar` **text*) Function

`void` **gtk_entry_set_position** (`GtkEntry` **entry*, `gint` *position*) Function

`gchar*` **gtk_entry_get_text** (`GtkEntry` **entry*) Function

Entry, ENTRY

5.12 The file selection dialog widget

5.12.1 Description

5.12.2 Signals

5.12.3 Functions

`guint` **gtk_file_selection_get_Type** (`void`) Function

`GtkWidget*` **gtk_file_selection_new** (`gchar` **title*) Function

`void` **gtk_file_selection_set_filename** (`GtkFileSelection` **filesel*, `gchar` **filename*) Function

`gchar*` **gtk_file_selection_get_filename** (`GtkFileSelection` **filesel*) Function

FileSelection, FILE_SELECTION

5.13 The frame widget

5.13.1 Description

5.13.2 Signals

5.13.3 Functions

`guint` **gtk_frame_get_type** (`void`) Function

`GtkWidget*` **gtk_frame_new** (`gchar` **label*) Function

`void` **gtk_frame_set_label** (`GtkFrame` **frame*, `gchar` **label*) Function

`void` **gtk_frame_set_label_align** (`GtkFrame` **frame*, `gfloat` *xalign*, `gfloat` *yalign*) Function

`void` **gtk_frame_set_shadow_type** (`GtkFrame` **frame*, `GtkShadowType` *type*) Function

Frame, FRAME

5.14 The horizontal box widget

5.14.1 Description

5.14.2 Signals

5.14.3 Functions

`guint` **gtk_hbox_get_type** (`void`) Function

`GtkWidget*` **gtk_hbox_new** (`gint` *homogeneous*, `gint` *spacing*) Function

HBox, HBOX

5.15 The horizontal ruler widget

5.15.1 Description

5.15.2 Signals

5.15.3 Functions

`guint` **gtk_hruler_get_type** (`void`) Function

`GtkWidget*` **gtk_hruler_new** (`void`) Function

HRuler, HRULER

5.16 The horizontal scale widget

5.16.1 Description

5.16.2 Signals

5.16.3 Functions

`guint` **gtk_hscale_get_type** (`void`) Function

`GtkWidget*` **gtk_hscale_new** (`GtkAdjustment *`*adjustment*) Function

HScale, HSCALE

5.17 The horizontal scrollbar widget

5.17.1 Description

5.17.2 Signals

5.17.3 Functions

`guint` **gtk_hscrollbar_get_type** (`void`) Function

`GtkWidget*` **gtk_hscrollbar_new** (`GtkAdjustment` **adjustment*) Function

HScrollbar, HSCROLLBAR

5.18 The horizontal separator widget

5.18.1 Description

5.18.2 Signals

5.18.3 Functions

`guint` **gtk_hseparator_get_type** (`void`) Function

`GtkWidget*` **gtk_hseparator_new** (`void`) Function

HSeparator, HSEPARATOR

5.19 The image widget

5.19.1 Description

5.19.2 Signals

5.19.3 Functions

`guint` **gtk_image_get_type** (`void`) Function

`GtkWidget*` **gtk_image_new** (`GdkImage` **val*) Function

`void` **gtk_image_set** (`GtkImage` **image*, `GdkImage` **val*) Function

`void` **gtk_image_get** (`GtkImage` **image*, `GdkImage` ***val*) Function

Image, IMAGE

5.20 The item widget

5.20.1 Description

5.20.2 Signals

`void` **GtkItem::select** (`GtkItem` **item*) Signal

`void` **GtkItem::deselect** (`GtkItem` **item*) Signal

`void` **GtkItem::toggle** (`GtkItem` **toggle*) Signal

5.20.3 Functions

`guint` **gtk_item_get_type** (`void`) Function

`void` **gtk_item_select** (`GtkItem` **item*) Function

`void` **gtk_item_deselect** (`GtkItem` **item*) Function

`void` **gtk_item_toggle** (`GtkItem` **item*) Function

Item, ITEM

5.21 The label widget

5.21.1 Description

5.21.2 Signals

5.21.3 Functions

`guint` **gtk_label_get_type** (`void`) Function

`GtkWidget*` **gtk_label_new** (`GtkLabel` **label*, `gchar` **str*) Function

`void` **gtk_label_set** (`GtkLabel` **label*, `gchar` **str*) Function

`void` **gtk_label_get** (`GtkLabel` **label*, `gchar` ***str*) Function

Label, LABEL

5.22 The list widget

5.22.1 Description

5.22.2 Signals

`void` **GtkList::selection_changed** (`GtkList *`*list*) Signal

`void` **GtkList::select_child** (`GtkList *`*list*, `GtkWidget *`*child*) Signal

`void` **GtkList::unselect_child** (`GtkList *`*list*, `GtkWidget *`*child*) Signal

5.22.3 Functions

`guint` **gtk_list_get_type** (`void`) Function

`GtkWidget*` **gtk_list_new** (`void`) Function

`void` **gtk_list_insert_items** (`GtkList *`*list*, `GList *`*items*, `gint` *position*) Function

`void` **gtk_list_append_items** (`GtkList *`*list*, `GList *`*items*) Function

`void` **gtk_list_prepend_items** (`GtkList *`*list*, `GList *`*items*) Function

`void` **gtk_list_remove_items** (`GtkList *`*list*, `GList *`*items*) Function

`void` **gtk_list_clear_items** (`GtkList *`*list*, `gint` *start*, `gint` *end*) Function

`void` **gtk_list_select_item** (`GtkList *`*list*, `gint` *item*) Function

`void` **gtk_list_unselect_item** (`GtkList *`*list*, `gint` *item*) Function

`void` **gtk_list_select_child** (`GtkList *`*list*, `GtkWidget *`*child*) Function

`void` **gtk_list_unselect_child** (`GtkList *`*list*, `GtkWidget *`*child*) Function

`gint` **gtk_list_child_position** (`GtkList *`*list*, `GtkWidget *`*child*) Function

`void` **gtk_list_set_selection_mode** (`GtkList *`*list*, `GtkSelectionMode` *mode*) Function

List, LIST

5.23 The list item widget

5.23.1 Description

5.23.2 Signals

5.23.3 Functions

`guint` **gtk_list_item_get_type** (`void`) Function

`GtkWidget*` **gtk_list_item_new** (`void`) Function

`GtkWidget*` **gtk_list_item_new_with_label** (`gchar` **label*) Function

`void` **gtk_list_item_select** (`GtkListItem` **list_item*) Function

`void` **gtk_list_item_deselect** (`GtkListItem` **list_item*) Function

ListItem, LIST_ITEM

5.24 The menu widget

5.24.1 Description

5.24.2 Signals

5.24.3 Functions

`guint` **gtk_menu_get_type** (`void`) Function

`GtkWidget*` **gtk_menu_new** (`void`) Function

`void` **gtk_menu_append** (`GtkMenu` **menu*, `GtkWidget` **child*) Function

`void` **gtk_menu_prepend** (`GtkMenu` **menu*, `GtkWidget` **child*) Function

`void` **gtk_menu_insert** (`GtkMenu` **menu*, `GtkWidget` **child*, `gint` *position*) Function

`void` **gtk_menu_popup** (`GtkMenu` **menu*, `GtkWidget` **parent_menu_shell*, `GtkWidget` **parent_menu_item*, `GtkMenuPositionFunc` *func*, `gpointer` *data*, `gint` *button*) Function

`void` **gtk_menu_popdown** (`GtkMenu` **menu*) Function

`GtkWidget*` **gtk_menu_get_active** (`GtkMenu` **menu*) Function

`void` **gtk_menu_set_active** (`GtkMenu` **menu*) Function

`void` **gtk_menu_set_accelerator_table** (`GtkMenu` **menu*, `GtkAcceleratorTable` **table*) Function

Menu, MENU

5.25 The menu bar widget

5.25.1 Description

5.25.2 Signals

5.25.3 Functions

`guint` **gtk_menu_bar_get_type** (`void`) Function

`GtkWidget*` **gtk_menu_bar_new** (`void`) Function

`void` **gtk_menu_bar_append** (`GtkMenuBar` **menu_bar*, `GtkWidget` **child*) Function

`void` **gtk_menu_bar_prepend** (`GtkMenuBar` **menu_bar*, `GtkWidget` **child*) Function

`void` **gtk_menu_bar_insert** (`GtkMenuBar` **menu_bar*, `GtkWidget` **child*, `gint` *position*) Function

MenuBar, MENU_BAR

5.26 The menu item widget

5.26.1 Description

5.26.2 Signals

`void` **GtkMenuItem::activate** (`GtkMenuItem` **menu_item*) Signal

5.26.3 Functions

`guint` **gtk_menu_item_get_type** (`void`) Function

`GtkWidget*` **gtk_menu_item_new** (`void`) Function

`GtkWidget*` **gtk_menu_item_new_with_label** (`gchar` **label*) Function

`void` **gtk_menu_item_set_submenu** (`GtkMenuItem` **menu_item*, `GtkWidget` **submenu*) Function

`void` **gtk_menu_item_set_placement** (`GtkMenuItem` **menu_item*, `GtkSubmenuPlacement` *placement*) Function

`void` **gtk_menu_item_accelerator_size** (`GtkMenuItem` **menu_item*) Function

`void` **gtk_menu_item_accelerator_text** (`GtkMenuItem` **menu_item*, `gchar` **buffer*) Function

`void` **gtk_menu_item_configure** (`GtkMenuItem` **menu_item*, `gint` *show_toggle_indicator*, `gint` *show_submenu_indicator*) Function

`void` **gtk_menu_item_select** (`GtkMenuItem` **menu_item*) Function

`void` **gtk_menu_item_deselect** (`GtkMenuItem` **menu_item*) Function

`void` **gtk_menu_item_activate** (`GtkMenuItem` **menu_item*) Function

MenuItem, MENU_ITEM

5.27 The menu shell widget

5.27.1 Description

5.27.2 Signals

`void` **GtkMenuShell::deactivate** (`GtkMenuShell *`*menu_shell*) Signal

5.27.3 Functions

`guint` **gtk_menu_shell_get_type** (`void`) Function

`void` **gtk_menu_shell_append** (`GtkMenuShell *`*menu_shell*, `GtkWidget *`*child*) Function

`void` **gtk_menu_shell_prepend** (`GtkMenuShell *`*menu_shell*, `GtkWidget *`*child*) Function

`void` **gtk_menu_shell_insert** (`GtkMenuShell *`*menu_shell*, `GtkWidget *`*child*, `gint` *position*) Function

`void` **gtk_menu_shell_deactivate** (`GtkMenuShell *`*menu_shell*) Function

MenuShell, MENU_SHELL

5.28 The misc widget

5.28.1 Description

5.28.2 Signals

5.28.3 Functions

`guint` **gtk_misc_get_type** (`void`) Function

`void` **gtk_misc_set_alignment** (`GtkMisc` **misc*, `gfloat` *xalign*, `gfloat` *yalign*) Function

`void` **gtk_misc_set_padding** (`GtkMisc` **misc*, `gint` *xpad*, `gint` *ypad*) Function

Misc, MISC

5.29 The notebook widget

5.29.1 Description

5.29.2 Signals

5.29.3 Functions

`guint` **gtk_notebook_get_type** (`void`) Function

`GtkWidget*` **gtk_notebook_new** (`void`) Function

`void` **gtk_notebook_append_page** (`GtkNotebook` **notebook*, `GtkWidget` **child*, `GtkWidget` **tab_label*) Function

`void` **gtk_notebook_prepend_page** (`GtkNotebook` **notebook*, `GtkWidget` **child*, `GtkWidget` **tab_label*) Function

`void` **gtk_notebook_insert_page** (`GtkNotebook` **notebook*, `GtkWidget` **child*, `GtkWidget` **tab_label*, `gint` *position*) Function

`void` **gtk_notebook_remove_page** (`GtkNotebook` **notebook*, `gint` *page_num*) Function

`void` **gtk_notebook_set_page** (`GtkNotebook` **notebook*, `gint` *page_num*) Function

`void` **gtk_notebook_next_page** (`GtkNotebook` **notebook*) Function

`void` **gtk_notebook_prev_page** (`GtkNotebook` **notebook*) Function

`void` **gtk_notebook_set_tab_pos** (`GtkNotebook` **notebook*, `GtkPositionType` *pos*) Function

`void` **gtk_notebook_set_show_tabs** (`GtkNotebook` **notebook*, `gint` *show_tabs*) Function

`void` **gtk_notebook_set_show_border** (`GtkNotebook` **notebook*, `gint` *show_border*) Function

Notebook, NOTEBOOK

5.30 The option menu widget

5.30.1 Description

5.30.2 Signals

5.30.3 Functions

`guint` **gtk_option_menu_get_type** (`void`) Function

`GtkWidget*` **gtk_option_menu_new** (`void`) Function

`GtkWidget*` **gtk_option_menu_get_menu** (`GtkOptionMenu` **option_menu*) Function

`void` **gtk_option_menu_set_menu** (`GtkOptionMenu` **option_menu*, `GtkWidget` **menu*) Function

`void` **gtk_option_menu_remove_menu** (`GtkOptionMenu` **option_menu*) Function

`void` **gtk_option_menu_set_history** (`GtkOptionMenu` **option_menu*, `gint` *index*) Function

OptionMenu, OPTION_MENU

5.31 The pixmap widget

5.31.1 Description

5.31.2 Signals

5.31.3 Functions

`guint` **gtk_pixmap_get_type** (`void`) Function

`GtkWidget*` **gtk_pixmap_new** (`GdkPixmap` **normal*, `GdkPixmap` **active*, `GdkPixmap` **prelight*, `GdkPixmap` **selected*, `GdkPixmap` **insensitive*) Function

`void` **gtk_pixmap_set** (`GtkPixmap` **pixmap*, `GdkPixmap` **val*, `GtkStateType` *state*) Function

`void` **gtk_pixmap_get** (`GtkPixmap` **pixmap*, `GdkPixmap` ***val*, `GtkStateType` *state*) Function

Pixmap, PIXMAP

5.32 The preview widget

5.32.1 Description

5.32.2 Signals

5.32.3 Functions

`guint` **gtk_preview_get_type** (`void`) Function

`void` **gtk_preview_uninit** (`void`) Function

`GtkWidget*` **gtk_preview_new** (`GtkPreviewType` *type*) Function

`void` **gtk_preview_size** (`GtkPreview` **preview*, `gint` *width*, `gint` *height*) Function

`void` **gtk_preview_put** (`GtkPreview` **preview*, `GdkWindow` **window*, `GdkGC` **gc*, `gint` *srcx*, `gint` *srcy*, `gint` *destx*, `gint` *desty*, `gint` *width*, `gint` *height*) Function

`void` **gtk_preview_put_row** (`GtkPreview` **preview*, `guchar` **src*, `guchar` **dest*, `gint` *x*, `gint` *y*, `gint* w*) Function

`void` **gtk_preview_draw_row** (`GtkPreview` **preview*, `guchar` *data*, `gint` *x*, `gint` *y*, `gint* w*) Function

`void` **gtk_preview_set_expand** (`GtkPreview` **preview*, `gint` *expand*) Function

`void` **gtk_preview_set_gamma** (`double` *gamma*) Function

`void` **gtk_preview_set_color_cube** (`guint` *nred_shades*, `guint` *ngreen_shades*, `guint` *nblue_shades*, `guint` *ngray_shades*) Function

`void` **gtk_preview_set_install_cmap** (`gint` *install_cmap*) Function

`void` **gtk_preview_set_reserved** (`gint` *nreserved*) Function

`GdkVisual*` **gtk_preview_get_visual** (`void`) Function

`GdkColormap*` **gtk_preview_get_cmap** (`void`) Function

`GtkPreviewInfo*` **gtk_preview_get_info** (`void`) Function

Preview, PREVIEW

5.33 The progress bar widget

5.33.1 Description

5.33.2 Signals

5.33.3 Functions

`guint` **gtk_progress_bar_get_type** (`void`) Function

`GtkWidget*` **gtk_progress_bar_new** (`void`) Function

`void` **gtk_progress_bar_update** (`GtkProgressBar` **pbar*, `gfloat` *percentage*) Function

ProgressBar, PROGRESS_BAR

5.34 The radio button widget

5.34.1 Description

5.34.2 Signals

5.34.3 Functions

`guint` **gtk_radio_button_get_type** (`void`) Function

`GtkWidget*` **gtk_radio_button_new** (`GSList` **group*) Function

`GtkWidget*` **gtk_radio_button_new_with_label** (`GSList` **group*, `gchar` **label*) Function

`GSList*` **gtk_radio_button_group** (`GtkRadioButton` **radio_button*) Function

RadioButton, RADIO_BUTTON

5.35 The radio button widget

5.35.1 Description

5.35.2 Signals

5.35.3 Functions

`guint` **gtk_radio_menu_item_get_type** (`void`) Function

`GtkWidget*` **gtk_radio_menu_item_new** (`GSList` **group*) Function

`GtkWidget*` **gtk_radio_menu_item_new_with_label** (`GSList` **group*, `gchar` **label*) Function

`GSList*` **gtk_radio_menu_item_group** (`GtkRadioMenuItem` **radio_menu_item*) Function

RadioMenuItem, RADIO_MENU_ITEM

5.36 The range widget

5.36.1 Description

5.36.2 Signals

5.36.3 Functions

`guint` **gtk_range_get_type** (`void`) Function

`GtkAdjustment*` **gtk_range_get_adjustment** (`GtkRange` *range*) Function

`void` **gtk_range_set_update_policy** (`GtkRange` *range*, `GtkUpdatePolicy` *policy*) Function

`void` **gtk_range_set_adjustment** (`GtkRange` *range*, `GtkAdjustment` *adjustment*) Function

`void` **gtk_range_draw_background** (`GtkRange` *range*) Function

`void` **gtk_range_draw_trough** (`GtkRange` *range*) Function

`void` **gtk_range_draw_slider** (`GtkRange` *range*) Function

`void` **gtk_range_draw_step_forw** (`GtkRange` *range*) Function

`void` **gtk_range_draw_step_back** (`GtkRange` *range*) Function

`void` **gtk_range_slider_update** (`GtkRange` *range*) Function

`gint` **gtk_range_trough_click** (`GtkRange` *range*, `gint` *x*, `gint` *y*) Function

`void` **gtk_range_default_hslider_update** (`GtkRange` *range*) Function

`void` **gtk_range_default_vslider_update** (`GtkRange` *range*) Function

`gint` **gtk_range_default_htrough_click** (`GtkRange` *range*, `gint` *x*, `gint` *y*) Function

`gint` **gtk_range_default_vtrough_click** (`GtkRange` *range*, `gint` *x*, `gint` *y*) Function

`void` **gtk_range_default_hmotion** (`GtkRange` *range*, `gint` *xdelta*, `gint` *ydelta*) Function

`void` **gtk_range_default_vmotion** (`GtkRange` *range*, `gint` *xdelta*, `gint` *ydelta*) Function

`gfloat` **gtk_range_calc_value** (`GtkRange` *ragne*, `gint` *position*) Function

Range, RANGE

5.37 The ruler widget

5.37.1 Description

5.37.2 Signals

5.37.3 Functions

`guint` **gtk_ruler_get_type** (`void`) Function

`void` **gtk_ruler_set_metric** (`GtkRuler` **ruler*, `GtkMetricType` *metric*) Function

`void` **gtk_ruler_set_range** (`GtkRuler` **ruler*, `gfloat` *lower*, `gfloat` *upper*, `gfloat` *position*, `gfloat` *max_size*) Function

`void` **gtk_ruler_draw_ticks** (`GtkRuler` **ruler*) Function

`void` **gtk_ruler_draw_pos** (`GtkRuler` **ruler*) Function

Ruler, RULER

5.38 The scale widget

5.38.1 Description

5.38.2 Signals

5.38.3 Functions

`guint` **gtk_scale_get_type** (`void`) Function

`void` **gtk_scale_set_digits** (`GtkScale` *_scale_, `gint` _digits_) Function

`void` **gtk_scale_set_draw_value** (`GtkScale` *_scale_, `gint` _draw_value_) Function

`void` **gtk_scale_set_value_pos** (`GtkScale` *_scale_, `gint` _pos_) Function

`gint` **gtk_scale_value_width** (`GtkScale` *_scale_) Function

`void` **gtk_scale_draw_value** (`GtkScale` *_scale_) Function

Scale, SCALE

5.39 The scrollbar widget

5.39.1 Description

5.39.2 Signals

5.39.3 Functions

`guint` **gtk_scrollbar_get_type** (`void`) Function

Scrollbar, SCROLLBAR

5.40 The scrolled window widget

5.40.1 Description

5.40.2 Signals

5.40.3 Functions

`guint` **gtk_scrolled_window_get_type** (`void`) Function

`GtkWidget*` **gtk_scrolled_window_new** (`GtkAdjustment` **hadjustment*, `GtkAdjustment` **vadjustment*) Function

`GtkAdjustment*` **gtk_scrolled_window_get_hadjustment** (`GtkScrolledWindow` **scrolled_window*) Function

`GtkAdjustment*` **gtk_scrolled_window_get_vadjustment** (`GtkScrolledWindow` **scrolled_window*) Function

`void` **gtk_scrolled_window_set_policy** (`GtkScrolledWindow` **scrolled_window*, `GtkPolicyType` *hscrollbar_policy*, `GtkPolicyType` *vscrollbar_policy*) Function

ScrolledWindow, SCROLLED_WINDOW

5.41 The separator widget

5.41.1 Description

5.41.2 Signals

5.41.3 Functions

`guint` **gtk_separator_get_type** (`void`) Function

Separator, SEPARATOR

5.42 The table widget

5.42.1 Description

5.42.2 Signals

5.42.3 Functions

`guint` **gtk_table_get_type** (`void`) Function

`GtkWidget*` **gtk_table_new** (`gint` *rows*, `gint` *columns*, `gint` *homogeneous*) Function

`void` **gtk_table_attach** (`GtkTable` `*`*table*, `GtkWidget` `*`*child*, `gint` *left_attach*, `gint` *right_attach*, `gint` *top_attach*, `gint` *bottom_attach*, `gint` *xoptions*, `gint` *yoptions*, `gint` *xpadding*, `gint` *ypadding*) Function

`void` **gtk_table_attach_defaults** (`GtkTable` `*`*table*, `GtkWidget` `*`*widget*, `gint` *left_attach*, `gint` *right_attach*, `gint` *top_attach*, `gint` *bottom_attach*) Function

`void` **gtk_table_set_row_spacing** (`GtkTable` `*`*table*, `gint` *row*, `gint` *spacing*) Function

`void` **gtk_table_set_col_spacing** (`GtkTable` `*`*table*, `gint` *col*, `gint` *spacing*) Function

`void` **gtk_table_set_row_spacings** (`GtkTable` `*`*table*, `gint` *spacing*) Function

`void` **gtk_table_set_col_spacings** (`GtkTable` `*`*table*, `gint` *spacing*) Function

Table, TABLE

5.43 The text widget

5.43.1 Description

5.43.2 Signals

5.43.3 Functions

`guint` **gtk_text_get_type** (`void`) Function

Text, TEXT

5.44 The toggle button widget

5.44.1 Description

5.44.2 Signals

`void` **GtkToggleButton::toggled** (`GtkToggleButton *`*toggle_button*) Signal

5.44.3 Functions

`guint` **gtk_toggle_button_get_type** (`void`) Function

`GtkWidget*` **gtk_toggle_button_new** (`void`) Function

`GtkWidget*` **gtk_toggle_button_new_with_label** (`gchar *`*label*) Function

`void` **gtk_toggle_button_set_mode** (`GtkToggleButton *`*toggle_button*, `gint` *draw_indicator*) Function

`void` **gtk_toggle_button_set_state** (`GtkToggleButton *`*toggle_button*, `gint` *state*) Function

`void` **gtk_toggle_button_toggled** (`GtkToggleButotn *`*toggle_button*) Function

ToggleButton, TOGGLE_BUTTON

5.45 The tree widget

5.45.1 Description

5.45.2 Signals

5.45.3 Functions

`guint` **gtk_tree_get_type** (`void`) Function

Tree, TREE

5.46 The tree item widget

5.46.1 Description

5.46.2 Signals

5.46.3 Functions

`guint` **gtk_tree_item_get_type** (`void`) Function

TreeItem, TREE_ITEM

5.47 The vertical box widget

5.47.1 Description

5.47.2 Signals

5.47.3 Functions

`guint` **gtk_vbox_get_type** (`void`) Function

`GtkWidget*` **gtk_vbox_new** (`gint` *homogeneous*, `gint` *spacing*) Function

VBox, VBOX

5.48 The viewport widget

5.48.1 Description

5.48.2 Signals

5.48.3 Functions

`guint` **gtk_viewport_get_type** (`void`) Function

`GtkWidget*` **gtk_viewport_new** (`GtkAdjustment` *hadjustment*, `GtkAdjustment` *vadjustment*) Function

`GtkAdjustment*` **gtk_viewport_get_hadjustment** (`GtkViewport` *viewport*) Function

`GtkAdjustment*` **gtk_viewport_get_vadjustment** (`GtkViewport` *viewport*) Function

`void` **gtk_viewport_set_hadjustment** (`GtkViewport` *viewport*, `GtkAdjustment` *adjustment*) Function

`void` **gtk_viewport_set_vadjustment** (`GtkViewport` *viewport*, `GtkAdjustment` *adjustment*) Function

`void` **gtk_viewport_set_shadow_type** (`GtkViewport` *viewport*, `GtkShadowType` *type*) Function

Viewport, VIEWPORT

5.49 The vertical ruler widget

5.49.1 Description

5.49.2 Signals

5.49.3 Functions

`guint` **gtk_vruler_get_type** (`void`) Function

`GtkWidget*` **gtk_vruler_new** (`void`) Function

VRuler, VRULER

5.50 The vertical ruler widget

5.50.1 Description

5.50.2 Signals

5.50.3 Functions

`guint` **gtk_vscale_get_type** (`void`) Function

`GtkWidget*` **gtk_vscale_new** (`GtkAdjustment *`*adjustment*) Function

VScale, VSCALE

5.51 The vertical scrollbar widget

5.51.1 Description

5.51.2 Signals

5.51.3 Functions

`guint` **gtk_vscrollbar_get_type** (`void`) Function

`GtkWidget*` **gtk_vscrollbar_new** (`GtkAdjustment *`*adjustment*) Function

VScrollbar, VSCROLLBAR

5.52 The vertical separator widget

5.52.1 Description

5.52.2 Signals

5.52.3 Functions

`guint` **gtk_vseparator_get_type** (`void`) Function

`GtkWidget*` **gtk_vseparator_new** (`void`) Function

VSeparator, VSEPARATOR

5.53 The base widget

5.53.1 Description

5.53.2 Signals

`void` **GtkWidget::show** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::hide** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::map** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::unmap** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::realize** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::unrealize** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::draw** (`GtkWidget` **widget*, `GdkRectangle` **area*) Signal

`void` **GtkWidget::draw_focus** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::draw_default** (`GtkWidget` **widget*) Signal

`void` **GtkWidget::size_request** (`GtkWidget` **widget*, `GtkRequisition` **requisition*) Signal

`void` **GtkWidget::size_allocate** (`GtkWidget` **widget*, `GtkAllocation` **allocation*) Signal

`void` **GtkWidget::state_changed** (`GtkWidget` **widget*) Signal

`gint` **GtkWidget::install_accelerator** (`GtkWidget` **widget*, `gchar` **signal_name*, `gchar` *key*, `guint8` *modifiers*) Signal

`void` **GtkWidget::remove_accelerator** (`GtkWidget` **widget*, `gchar` **signal_name*) Signal

`gint` **GtkWidget::event** (`GtkWidget` **widget*, `GdkEvent` **event*) Signal

`gint` **GtkWidget::button_press_event** (`GtkWidget` **widget*, `GdkEventButton` **event*) Signal

`gint` **GtkWidget::button_release_event** (`GtkWidget` **widget*, `GdkEventButton` **event*) Signal

`gint` **GtkWidget::motion_notify_event** (`GtkWidget` **widget*, `GdkEventMotion` **event*) Signal

`gint` **GtkWidget::delete_event** (`GtkWidget` **widget*, `GdkEventAny` **event*) Signal

`gint` **GtkWidget::destroy_event** (`GtkWidget` **widget*, `GdkEventAny` **event*) Signal

`gint` **GtkWidget::expose_event** (`GtkWidget *`*widget*, `GdkEventExpose *`*event*) Signal

`gint` **GtkWidget::key_press_event** (`GtkWidget *`*widget*, `GdkEventKey *`*event*) Signal

`gint` **GtkWidget::key_release_event** (`GtkWidget *`*widget*, `GdkEventKey *`*event*) Signal

`gint` **GtkWidget::enter_notify_event** (`GtkWidget *`*widget*, `GdkEventCrossing *`*event*) Signal

`gint` **GtkWidget::leave_notify_event** (`GtkWidget *`*widget*, `GdkEventCrossing *`*event*) Signal

`gint` **GtkWidget::configure_event** (`GtkWidget *`*widget*, `GdkEventConfigure *`*event*) Signal

`gint` **GtkWidget::focus_in_event** (`GtkWidget *`*widget*, `GdkEventFocus *`*event*) Signal

`gint` **GtkWidget::focus_out_event** (`GtkWidget *`*widget*, `GdkEventFocus *`*event*) Signal

`gint` **GtkWidget::map_event** (`GtkWidget *`*widget*, `GdkEventAny *`*event*) Signal

`gint` **GtkWidget::unmap_event** (`GtkWidget *`*widget*, `GdkEventAny *`*event*) Signal

`gint` **GtkWidget::property_notify_event** (`GtkWidget *`*widget*, `GdkEventProperty *`*event*) Signal

`gint` **GtkWidget::selection_clear_event** (`GtkWidget *`*widget*, `GdkEventSelection *`*event*) Signal

`gint` **GtkWidget::selection_request_event** (`GtkWidget *`*widget*, `GdkEventSelection *`*event*) Signal

`gint` **GtkWidget::selection_notify_event** (`GtkWidget *`*widget*, `GdkEventSelection *`*event*) Signal

`gint` **GtkWidget::drop_event** (`GtkWidget *`*widget*, `GdkEventDrop *`*event*) Signal

`gint` **GtkWidget::drag_begin_event** (`GtkWidget *`*widget*, `GdkEventDragBegin *`*event*) Signal

`gint` **GtkWidget::other_event** (`GtkWidget *`*widget*, `GdkEventOther *`*event*) Signal

5.53.3 Functions

`guint` **gtk_widget_get_type** (`void`) Function

`void` **gtk_widget_class_init** (`GtkWidgetClass *`*class*) Function

`void` **gtk_widget_init** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_destroy** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_show** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_hide** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_map** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_unmap** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_realize** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_unrealize** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_draw** (`GtkWidget *`*widget*, `GdkRectangle *`*area*) Function

`void` **gtk_widget_draw_focus** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_draw_children** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_size_request** (`GtkWidget *`*widget*, `GtkRequisition *`*requisition*) Function

`void` **gtk_widget_size_allocate** (`GtkWidget *`*widget*, `GtkAllocation *`*allocation*) Function

`void` **gtk_widget_install_accelerator** (`GtkWidget *`*widget*, `GtkAcceleratorTable *`*table*, `gchar *`*signal_name*, `gchar` *key*, `guint8` *modifiers*) Function

`void` **gtk_widget_remove_accelerator** (`GtkWidget *`*widget*, `GtkAcceleratorTable *`*table*, `gchar *`*signal_name*) Function

`gint` **gtk_widget_event** (`GtkWidget *`*widget*, `GdkEvent *`*event*) Function

`void` **gtk_widget_reparent** (`GtkWidget *`*widget*, `GtkWidget *`*new_parent*) Function

`void` **gtk_widget_popup** (`GtkWidget *`*widget*, `gint` *x*, `gint` *y*) Function

`gint` **gtk_widget_intersect** (`GtkWidget *`*widget*, `GdkRectangle *`*area*, `GdkRectangle *`*intersection*) Function

`void` **gtk_widget_grab_focus** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_grab_default** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_restore_state** (`GtkWidget *`*widget*) Function

`void` **gtk_widget_set_name** (`GtkWidget *`*widget*, `gchar *`*name*) Function

`void` **gtk_widget_set_state** (`GtkWidget *`*widget*, `GtkStateType` *state*) Function

`void` **gtk_widget_set_sensitive** (`GtkWidget *`*widget*, `gint sensitive`) Function

`void` **gtk_widget_set_parent** (`GtkWidget *`*widget*, `GtkWidget *`*parent*) Function

`void` **gtk_widget_set_style** (`GtkWidget *`*widget*, `GtkStyle *`*style*) Function

`void` **gtk_widget_set_uposition** (`GtkWidget *`*widget*, `gint` *x*, `gint` *y*) Function

`void` **gtk_widget_set_usize** (`GtkWidget *`*widget*, `gint` *width*, `gint` *height*) Function

`GtkWidget*` **gtk_widget_get_toplevel** (`GtkWidget *`*widget*) Function

`GtkWidget*` **gtk_widget_get_ancestor** (`GtkWidget *`*widget*, `gint` *type*) Function

`GdkColormap*` **gtk_widget_get_colormap** (`GtkWidget *`*widget*) Function

`GdkVisual*` **gtk_widget_get_visual** (`GtkWidget *`*visual*) Function

`GtkStyle*` **gtk_widget_get_style** (`GtkWidget *`*style*) Function

Widget, WIDGET

5.54 The window widget

5.54.1 Description

5.54.2 Signals

`void` **GtkWindow::move_resize** (`GtkWindow` *_window_, `gint` *_x_, `gint` *_y_, `gint` _width_, `gint` _height_) — Signal

5.54.3 Functions

`guint` **gtk_window_get_type** (`void`) — Function

`GtkWidget*` **gtk_window_new** (`GtkWindowType` _type_) — Function

`void` **gtk_window_set_title** (`GtkWindow` *_window_, `gchar` *_title_) — Function

`void` **gtk_window_set_focus** (`GtkWindow` *_window_, `GtkWidget` *_focus_) — Function

`void` **gtk_window_set_default** (`GtkWindow` *_window_, `GtkWidget` *_defaultw_) — Function

`void` **gtk_window_set_policy** (`GtkWindow` *_window_, `gint` _allow_shrink_, `gint` _allow_grow_, `gint` _auto_shrink_) — Function

`void` **gtk_window_add_accelerator_table** (`GtkWindow` *_window_, `GtkAcceleratorTable` *_table_) — Function

`void` **gtk_window_remove_accelerator_table** (`GtkWindow` *_window_, `GtkAcceleratorTable` *_table_) — Function

`void` **gtk_window_position** (`GtkWindow` *_window_, `GtkWindowPosition` _position_) — Function

Window, WINDOW

6 Utility objects

6.1 The adjustment object

6.2 The data object

7 Initialization, exit and other features

7.1 Initializing and exiting GTK

7.2 Simplified menu creation

7.3 Simplified tree creation

7.4 Pop up help mechanism

7.5 Pop up help mechanism

7.6 Macros defined by all objects

There are three macros that are defined by all object types. The first two are used for performing casts and the last is for querying whether an object is of a particular type. These macros are both conveniences and debugging tools. If the GTK library was compiled with `NDEBUG` defined as a preprocessor symbol (via the -DNDEBUG to cc), then the macros check the object type and emit a warning if the cast is invalid. Doing such checking is fairly expensive since the cast macros are used everywhere in GTK and would normally be turned off in a public release of a product. Note: The functions below are indeed macros, but they may be considered functions for most purposes.

`Gtk<ObjectType>*` **GTK_<OBJECT_TYPE>** (`gpointer` *obj*) Function

Cast a generic pointer to `Gtk<ObjectType>*`. This function is provided in order to be able to provide checking during development stages of code development since it is possible to examine the actual type of object (using `gtk_type_is_a`) before performing the cast.

`Gtk<ObjectType>Class*` **GTK_<OBJECT_TYPE>_CLASS** (`gpointer` *class*) Function

Cast a generic pointer to `Gtk<ObjectType>Class*`. Like `GTK_<ObjectType>`, this function is, in reality, a macro.

`gint` **GTK_IS_<ObjectType>** (`gpointer` *obj*) Function

Determine if a generic pointer refers to a `Gtk<ObjectType>` object. This function is, in reality, a macro wrapper around the `gtk_type_is_a` function (see ⟨undefined⟩ [Objects], page ⟨undefined⟩).

8 Using GTK

8.1 The simplest GTK program

The 16 line GTK program shown below is just about the simplest possible program which uses GTK. (Well, technically, you don't have to create the window and it would still be a program which uses GTK). The program, when compiled and run, will create a single window 200x200 pixels in size. The program does not exit until its is explicitly killed using the shell or a window manager function.

```
#include <gtk/gtk.h>

int
main (int argc, char *argv[])
{
  GtkWidget *window;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_show (window);

  gtk_main ();

  return 0;
}
```

The first point of interest in this program is the standard initialization line.

```
gtk_init (&argc, &argv);
```

Almost every GTK program will contain such a line. GTK will initialize itself and GDK and remove any command line arguments it recognizes from *argc* and *argv*.

The next two lines of code create and display a window.

```
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_widget_show (window);
```

The `GTK_WINDOW_TOPLEVEL` argument specifies that we want the window to undergo window manager decoration and placement. One might be lead to think that the window, since it has no children, would be 0x0 pixels in size. But, this is not the case because a window that has no children defaults to 200x200 pixels in size. Mainly because 0x0 windows are annoying to manipulate or even see in some cases.

The last line enters the GTK main processing loop.

```
gtk_main ();
```

Normally, `gtk_main` is called once and the program should exit when it returns. See ⟨undefined⟩ [Initialization and exit], page ⟨undefined⟩.

8.2 Hello world in GTK

```
#include <gtk/gtk.h>

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *label;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_container_border_width (GTK_CONTAINER (window), 10);

  label = gtk_label_new ("Hello World");
  gtk_container_add (GTK_CONTAINER (window), label);
  gtk_widget_show (label);

  gtk_widget_show (window);

  gtk_main ();

  return 0;
}
```

8.3 An enhanced hello world

```
#include "gtk.h"

void
hello (void)
{
  g_print ("Hello World\n");
  gtk_exit (0);
}

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_container_border_width (GTK_CONTAINER (window), 10);

  button = gtk_button_new_with_label ("Hello World");
```

```
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
      GTK_SIGNAL_FUNC (hello), NULL);
  gtk_container_add (GTK_CONTAINER (window), button);
  gtk_widget_show (button);

  gtk_widget_show (window);

  gtk_main ();

  return 0;
}
```

8.4 Making Hello World II robust

```
#include "gtk.h"

void
hello (void)
{
  g_print ("Hello World\n");
  gtk_exit (0);
}

void
destroy (void)
{
  gtk_exit (0);
}

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_signal_connect (GTK_OBJECT (window), "destroy",
      GTK_SIGNAL_FUNC (destroy), NULL);
  gtk_container_border_width (GTK_CONTAINER (window), 10);

  button = gtk_button_new_with_label ("Hello World");
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
      GTK_SIGNAL_FUNC (hello), NULL);
  gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
     GTK_SIGNAL_FUNC (gtk_widget_destroy),
     GTK_OBJECT (window));
```

```
  gtk_container_add (GTK_CONTAINER (window), button);
  gtk_widget_show (button);

  gtk_widget_show (window);

  gtk_main ();

  return 0;
}
```

9 Object internals

Objects (or the `GtkObject` type) and the class hierarchy in general is implemented via a hierarchy of structs and type casting. Be aware that when classes are mentioned it is the conceptual idea of classes that is being referred to. GTK is written entirely in C which provides no direct support for classes.

The first part to the class mechanism is the object fields. These are fields that will be used on a per object basis. For example, the widget type contains a field for the widgets parent. Every derived type needs a reference to its parent type. A descendant class of `GtkObject` would define itself like:

```
struct Descendant
{
  GtkObject object;

  ...
};
```

It is important to note that the `GtkObject` field needs to appear first in the descendant type structure. This allows pointers to objects of type `Descendant` to be cast to pointers to `GtkObjects`'s and vice-versa.

The second part to the class mechanism is the class fields. These fields are defined on a per class basis. In the case of widgets, the class fields are all the "virtual" functions for widgets. The `GtkObject` class defines the `destroy` virtual function and the necessary fields for the signal mechanism as well as a field for determining the runtime type of an object. A virtual function is semantically the same as it is in C++. That is, the actual function that is called is determined based on the type of the object. Or, more specifically, the actual function call depends on the class structure that is pointed to by the `klass` field of the `GtkObject` structure.

To see how the class fields work it is necessary to see the object fields for a `GtkObject`. The `GtkObject` type is defined as follows:

```
typedef struct _GtkObject GtkObject;

struct _GtkObject
{
  guint32 flags;
  GtkObjectClass *klass;
  gpointer object_data;
};
```

The `class` field actually points to a class structure derived from `GtkObjectClass`. By convention, each new type defines its own class structure even if it is unnecessary. As an example, the hypothetical `Descendant` class would define its class structure as:

```
struct DescendantClass
{
  GtkObjectClass parent_class;

  ...
```

```
};
```

It is convention to name the parent class field (`GtkObjectClass` in this case), `parent_class`. For the same reason as stated above for the object structure, the parent class field must be the first field in the class structure.

Note: GTK assumes that the first field in a structure will be placed by the compiler at the start of the structure. This is certainly true for gcc, however, from my precursory reading of the C standard I was unable to come to a definite conclusion as to whether this was required or simply done for simplicity. I'm not too worried about this assumption, though, as every C compiler I've ever encountered would work with GTK.

The `flags` field of the `GtkObject` structure is used to keep track of a relatively few object flags and is also used by the `GtkWidget` type to store additional flags. At this time, the upper 16 bits of the flags field are reserved but unused.

The `object_data` field of the `GtkObject` structure is an opaque pointer used by the object data mechanism. In truth, it is a pointer to the beginning of the data list which is composed of the following structures.

```
typedef struct _GtkObjectData GtkObjectData;

struct _GtkObjectData
{
  guint id;
  gpointer data;
  GtkObjectData *next;
};
```

The data mechanism allows arbitrary data to be associated with a character string key in any object. A hash table is used to transform the character string key into the data id and then a search through the list is made to see if the data exists. The assumption being that the data list will usually be short and therefore a linear search is ok. Future work on the data mechanism might make use of a resizable array instead of a linked list. This would shrink the overhead of the `GtkObjectData` structure by 4 bytes on 32 bit architectures.

10 Signal internals

11 Widget internals

Function Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Short Contents

Table of Contents