
GtkAda Documentation

Release 28.0w

AdaCore

Oct 08, 2026

CONTENTS

1	Introduction: What is GtkAda ?	1
2	Getting started with GtkAda	3
2.1	How to build and install GtkAda	3
2.2	How to distribute a GtkAda application	3
2.3	How to use GtkAda	4
2.4	Organization of the GtkAda package	4
2.5	How to compile an application with GtkAda	5
2.6	Architecture of the toolkit	5
2.7	Widgets Hierarchy	7
3	Transitioning from GtkAda 3 to GtkAda 4	9
4	Hierarchical composition of a window	11
5	Signal handling	13
5.1	Predefined signals	13
5.2	Connecting signals	13
5.2.1	Connecting via the <i>On_*</i> procedures	14
5.2.2	Connecting via the <i>Gtk.Handlers</i> package	14
5.3	Handling user data	16
5.3.1	First case: simple user data	17
5.3.2	Second case: using <i>Object_Connect</i> instead	17
5.3.3	Third case: manually disconnecting the callback	17
5.3.4	Fourth case: setting a watch on a specific widget	18
6	Starting an application with GtkAda	19
6.1	The main procedure	19
6.2	Creating the main window	20
6.3	Ending the application	21
6.4	Startup and shutdown	21
6.5	Command line and single instance	22
7	Resource files	23
8	Memory management	25
9	Tasking with GtkAda	27
10	Processing external events	29

11	Object-oriented features	31
11.1	General description of the tagged types	31
11.1.1	Why should I use object-oriented programming ?	31
11.1.2	Type conversions from C to Ada widgets	31
11.2	Using tagged types to extend GTK widgets	32
11.3	Creating new widgets in Ada	32
11.3.1	Creating composite widgets	33
11.3.2	Creating widgets from scratch	33
12	Binding new widgets	37
13	Debugging GtkAda applications	39
14	How to report bugs	41
15	System Package Dependencies	43
15.1	Red Hat Enterprise Linux 7 & 8	43
15.2	Red Hat Enterprise Linux 9	44
15.3	Ubuntu 18.04 & 20.04	44
16	Using GtkParasite to inspect and modify running GtkAda applications	47
17	Bibliography	49
18	Transitioning from GtkAda 2 to GtkAda 3	51
18.1	General	51
18.1.1	Interfaces	51
18.1.2	Ada 2012	52
18.2	Pango	52
18.2.1	Pango.Font	52
18.3	Glib	52
18.3.1	Glib.Object	52
18.3.2	Glib.G_Icon	52
18.4	Gdk	52
18.4.1	Gdk.Bitmap	52
18.4.2	Gdk.Color	53
18.4.3	Gdk.Cursor	53
18.4.4	Gdk.Dnd	53
18.4.5	Gdk.Drawable	53
18.4.6	Gdk.Event	53
18.4.7	Gdk.Font	54
18.4.8	Gdk.GC	54
18.4.9	Gdk.Image	54
18.4.10	Gdk.Main	54
18.4.11	Gdk.Pixbuf	54
18.4.12	Gdk.Pixmap	54
18.4.13	Gdk.Region	54
18.4.14	Gdk.RGB	55
18.4.15	Gdk.Window	55
18.4.16	Gdk.Window_Attr	55
18.5	Gtk	55
18.5.1	Gtk.Action	55
18.5.2	Gtk.Aspect_Frame	56
18.5.3	Gtk.Assistant	56
18.5.4	Gtk.Builder	56

18.5.5	Gtk.Button_Box	56
18.5.6	Gtk.Cell_Layout	56
18.5.7	Gtk.Cell_Renderer	56
18.5.8	Gtk.Cell_View	56
18.5.9	Gtk.Clist	56
18.5.10	Gtk.Container	56
18.5.11	Gtk.Color_Button	57
18.5.12	Gtk.Color_Selection	57
18.5.13	Gtk.Color_Selection_Dialog	57
18.5.14	Gtk.Combo	57
18.5.15	Gtk.Combo_Box	57
18.5.16	Gtk.Combo_Box_Entry	57
18.5.17	Gtk.Clipboard	57
18.5.18	Gtk.Ctree	57
18.5.19	Gtk.Curve	58
18.5.20	Gtk.Dialog	58
18.5.21	Gtk.Dnd	58
18.5.22	Gtk.Editable	58
18.5.23	Gtk.Entry_Completion	59
18.5.24	Gtk.Enums	59
18.5.25	Gtk.File_Chooser_Button	59
18.5.26	Gtk.File_Chooser_Dialog	59
18.5.27	Gtk.File_Chooser_Widget	59
18.5.28	Gtk.File_Selection	60
18.5.29	Gtk.Fixed	60
18.5.30	Gtk.Gamma_Curve	60
18.5.31	Gtk.GC	60
18.5.32	Gtk.GEntry	60
18.5.33	Gtk.GRange	60
18.5.34	Gtk.Handle_Box	60
18.5.35	Gtk.HRuler	60
18.5.36	Gtk.Icon_Factory	60
18.5.37	Gtk.Image	61
18.5.38	Gtk.Image_Menu_Item	61
18.5.39	Gtk.Input_Dialog	61
18.5.40	Gtk.Item	61
18.5.41	Gtk.Item_Factory	61
18.5.42	Gtk.Layout	61
18.5.43	Gtk.Link_Button	61
18.5.44	Gtk.List_Item	61
18.5.45	Gtk.Main	61
18.5.46	Gtk.Menu	62
18.5.47	Gtk.Menu_Item	62
18.5.48	Gtk.Menu_Tool_Button	62
18.5.49	Gtk.Notebook	62
18.5.50	Gtk.List	62
18.5.51	Gtk.Object	62
18.5.52	Gtk.Old_Editable	63
18.5.53	Gtk.Option_Menu	63
18.5.54	Gtk.Pixmap	63
18.5.55	Gtk.Preview	63
18.5.56	Gtk.Print_Operation	63
18.5.57	Gtk.Progress	63
18.5.58	Gtk.Progress_Bar	63

18.5.59	Gtk.Rc	63
18.5.60	Gtk.Recent_Manager	64
18.5.61	Gtk.Ruler	64
18.5.62	Gtk.Settings	64
18.5.63	Gtk.Scale_Button	64
18.5.64	Gtk.Selection	64
18.5.65	Gtk.Scrolled_Window	65
18.5.66	Gtk.Socket / Gtk.Plug	65
18.5.67	Gtk.Status_Icon	65
18.5.68	Gtk.Style	65
18.5.69	Gtk.Text	65
18.5.70	Gtk.Text_Attributes	65
18.5.71	Gtk.Text_View	65
18.5.72	Gtk.Tree_Dnd	66
18.5.73	Gtk.Tree_Model	66
18.5.74	Gtk.Tree_View_Column	66
18.5.75	Gtk.Tips_Query	66
18.5.76	Gtk.Tool_Item	66
18.5.77	Gtk.Toolbar	66
18.5.78	Gtk.Tooltips	67
18.5.79	Gtk.Tree_View	67
18.5.80	Gtk.VRuler	67
18.5.81	Gtk.Widget	67
18.5.82	Gtk.Window	69
18.6	GtkAda	69
18.6.1	Gtkada.MDI	69
18.6.2	Gtkada.Properties	69
18.7	Gnome	69
18.7.1	Gnome.App_Bar	69
18.7.2	Gnome.Gentry	69
19	GNU Free Documentation License	71
19.1	PREAMBLE	71
19.2	APPLICABILITY AND DEFINITIONS	71
19.3	VERBATIM COPYING	72
19.4	COPYING IN QUANTITY	72
19.5	MODIFICATIONS	73
19.6	COMBINING DOCUMENTS	74
19.7	COLLECTIONS OF DOCUMENTS	74
19.8	AGGREGATION WITH INDEPENDENT WORKS	74
19.9	TRANSLATION	75
19.10	TERMINATION	75
19.11	FUTURE REVISIONS OF THIS LICENSE	75
19.12	ADDENDUM: How to use this License for your documents	75

INTRODUCTION: WHAT IS GTKADA ?

GtkAda is a high-level portable graphical toolkit, based on the GTK toolkit, one of the official GNU toolkits. It makes it easy to create portable user interfaces for multiple platforms, including Linux (on both Wayland and X11) and Windows.

Although it is based on a C library, GtkAda uses some advanced Ada features such as tagged types, generic packages, access to subprograms, and exceptions to make it easier to use and design interfaces. For efficiency reasons, it does not use controlled types, but takes care of all the memory management for you in other ways.

As a result, this library provides a *secure, easy to use* and *extensible* toolkit.

Compared to the C library, GtkAda provides type safety (especially in the callbacks area), and object-oriented programming. As opposed to common knowledge, it requires *less* type casting than with in C. Its efficiency is about the same as the C library through the use of inline subprograms.

Under some platforms, GtkAda also provides a bridge to use OpenGL, with which you can create graphical applications that display 3D graphics, and display them in a GtkAda window, as with any other 2D graphics. This manual does not document OpenGL at all, see any book on OpenGL, or the specification that came with your OpenGL library, for more information.

The following Internet sites will always contain the latest public packages for *GtkAda*, *GTK* and *Cairo*

- <https://github.com/AdaCore/gtkada>
- <https://www.gtk.org/>
- <https://www.cairographics.org/>

This toolkit was tested on the following systems:

- GNU Linux/x86-64
- GNU Linux/AArch64
- Windows/x86-64

with recent versions of the *GNAT* compiler.

This version of GtkAda is known to be compatible with *GTK 4.22.x*. This release may not be compatible with older versions of GTK 4.

This document does not describe all the widgets available in GtkAda, nor does it try to explain all the subprograms. The GtkAda Reference Manual provides this documentation instead, as well as the GtkAda sources spec files themselves, whose extension is `.ads`.

No complete example is provided in this documentation. Instead, please refer to the examples that you can find in the `share/examples/gtkada/gtkada_demo/` directory in the GtkAda distribution, since these are more up-to-date (and more extensive). They are heavily commented, and are likely to contain a lot of information that you might find interesting.

If you are interested in getting support for GtkAda—including priority bug fixes, early releases, help in using the toolkit, help in designing your interface, and on site consulting—please contact AdaCore (<mailto:sales@adacore.com>).

GETTING STARTED WITH GTKADA

This chapter describes how to start a new GtkAda application. It explains the basic features of the toolkit, and shows how to compile and run your application.

It also gives a brief overview of the extensive widget hierarchy available in GtkAda.

2.1 How to build and install GtkAda

This section explains how to build and install GtkAda on your machine.

On Windows systems, we provide an automatic installer that installs GtkAda along with dependent components like GTK libraries.

On Unix systems, just follow the instructions listed in the `INSTALL` file. Note that the `doinstall` script will compile GtkAda, so make sure the version of GNAT that you want to use is in your `PATH`.

2.2 How to distribute a GtkAda application

Since GtkAda depends on GTK, you usually need to distribute some GTK libraries along with your application.

Under some OSes such as Linux, GTK comes preinstalled, so in this case, a simple solution is to rely on the preinstalled GTK libraries. See below for more information on the `gtkada` library itself.

Under other unix systems, GtkAda usually comes with a precompiled set of GTK libraries that have been specifically designed to be easily redistributed.

In order to use the precompiled GTK binaries that we distribute with GtkAda, you need to distribute all the GTK `.so` libraries along with your application, and use the `LD_LIBRARY_PATH` environment variable to point to these libraries.

The list of libraries needed is `<gtkada-prefix>/lib/lib*.so.?` or `<gtkada-prefix>/lib64/lib*.so.?` along with your executable, and set `LD_LIBRARY_PATH`.

You may also need the `libgtkada-xxx.so` file. This dependency is optional since `gtkada` supports both static and dynamic linking, depending on how your project sets up the library type in `gtkada.gpr`. You might chose to link with the static library `libgtkada.a` for convenience.

Under Windows, you need to distribute the following files and directories along with your application, and respect the original directory set up:

- `bin/*.dll`
- `etc/`
- `lib/gtk-2.0`

2.3 How to use GtkAda

On Unix systems, to use GtkAda, you need to have your `PATH` and `LD_LIBRARY_PATH` environment variables set, as explained above:

```
PATH=$prefix/bin:$PATH
LD_LIBRARY_PATH=$prefix/lib:$LD_LIBRARY_PATH
export PATH LD_LIBRARY_PATH
```

Set the following variables as well when using a custom GTK build (but not if you are using the system's libraries):

```
GDK_PIXBUF_MODULE_FILE=$prefix/lib/gdk-pixbuf-2.0/2.10.0/loaders.cache
GDK_PIXBUF_MODULEDIR=$prefix/lib/gdk-pixbuf-2.0/2.10.0/loaders/
export GDK_PIXBUF_MODULEDIR GDK_PIXBUF_MODULE_FILE

FONTCONFIG_FILE=$prefix/etc/fonts/fonts.conf
export FONTCONFIG_FILE

XDG_DATA_DIRS=$XDG_DATA_DIRS:$prefix/share
export XDG_DATA_DIRS
```

`GDK_PIXBUF_MODULE_FILE` contains the paths to find the `libpixbufloader-*` libraries. By default, the paths are relative to the executable loading the libraries: thus, `GDK_PIXBUF_MODULEDIR` must be installed relatively to the executable. When the paths are absolute, the location of `GDK_PIXBUF_MODULEDIR` doesn't matter when loading the libraries however it will be necessary to re-generate `GDK_PIXBUF_MODULE_FILE` on each host. Two executables are packaged with GtkAda to re-generate the modules' paths: `gdk-pixbuf-query-loaders`, which generates relative paths, and `gdk-pixbuf-query-loaders-absolute`, which generates absolute paths. Both should be launched with the `-update-cache` option to re-generate the proper cache file (by default it will just output the contents on stdout).

2.4 Organization of the GtkAda package

In addition to the full sources, the GtkAda package contains a lot of heavily commented examples. If you haven't been through those examples, we really recommend that you look at them and try to understand them, since they contain some examples of code that you might find interesting for your own application.

- `gtkada_demo/` directory:

This directory contains the *GtkAda Demo* that showcases most of the widgets in GtkAda. It gives you a quick overview of what can be found in the toolkit, as well as some detailed information on the widgets and their parameters.

Each demo is associated with contextual help pointing to aspects worth studying.

It also contains an OpenGL demo, if GtkAda was compiled with support for OpenGL.

This program is far more extensive than its C counterpart, and the GtkAda team has added a lot of new examples.

This directory also contains the application *testcairo* which demonstrates the use of various Cairo functions in GtkAda.

- `docs/` directory:

It contains the HTML and PDF versions of the documentation you are currently reading. Note that the documentation is divided into two subdirectories, one containing the user guide, which you are currently reading, the other containing the reference manual, which gives detailed information on all the widgets found in GtkAda. The `docs` directory also contains a subdirectory with some slides that were used to present GtkAda at various shows.

2.5 How to compile an application with GtkAda

This section explains how you can compile your own applications.

Note: If GNAT is already installed on your system, it is recommended to install GtkAda in a separate directory to avoid potential file conflicts.

A set of project files is installed along with GtkAda.

You need to make the directory that contains these project files visible to the compiler. This is done by adding the directory to the *GPR_PROJECT_PATH* environment variable. Assuming you have installed the library in *prefix*, the directory you need to add is *prefix/lib/gnat*.

On Unix, this is done with:

```
csh:
  setenv GPR_PROJECT_PATH $prefix/lib/gnat:$GPR_PROJECT_PATH
sh:
  GPR_PROJECT_PATH=$prefix/lib/gnat:$GPR_PROJECT_PATH
  export GPR_PROJECT_PATH
```

To build your own application, you should then setup a project file (see the GNAT documentation for more details on project files), which simply contains the statement:

```
with "gtkada";
```

This will automatically set the right compiler and linker options, so that your application is linked with GtkAda.

By default, the linker will use GtkAda's shared library, if it was built. If you would prefer to link with the static library, you can set the environment variable:

```
LIBRARY_TYPE=static
export LIBRARY_TYPE
```

before launching the compiler or linker, which will force it to use the static library instead.

2.6 Architecture of the toolkit

The GTK toolkit has been designed from the beginning to be portable. It is made of two libraries: *gtk* and *gdk*. In addition, GtkAda provides binding to three supporting libraries: *pango*, *cairo* and *glib*.

Glib is a non-graphical library that includes support for lists, h-tables, threads, and so on. It is a highly optimized, platform-independent library. Since most of its contents are already available in Ada (or in the GNAT.* hierarchy in the GNAT distribution), GtkAda does not include a complete binding to it. For the parts of *Glib* that we do depend on, we provide *Glib.** packages in the GtkAda distribution.

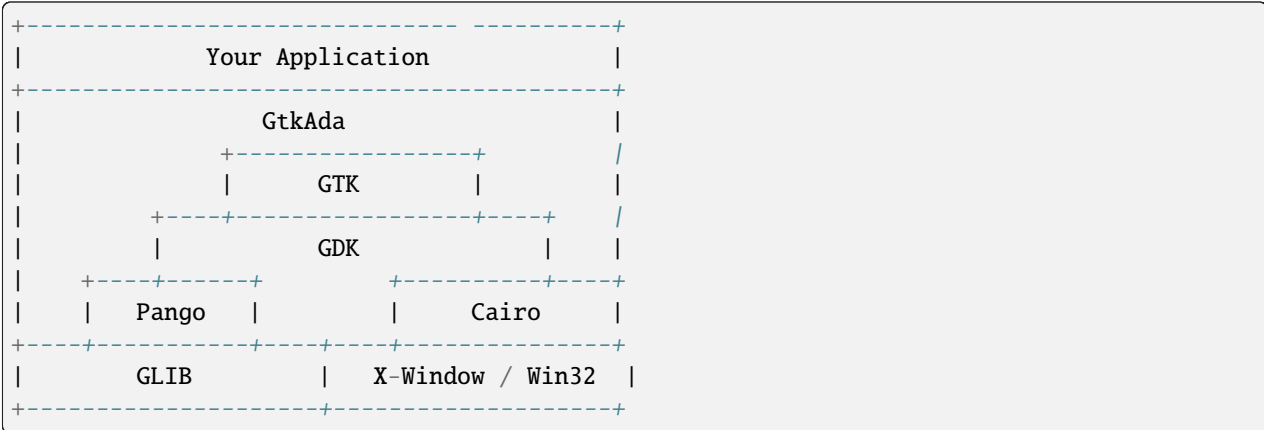
Gdk is the platform-dependent part of GTK, and so there are different implementations (for instance, for Win32 and X11 based systems) that implement a common API. *Gdk* provides basic graphical functionality to, for instance, draw lines, rectangles and pixmaps on the screen, as well as manipulate colors. The *Gdk.** packages provide a full Ada interface to *Gdk*.

Pango is a modern font handling system. Bindings in GtkAda gives access to the API to manipulate font descriptions and text attributes.

Cairo is the low-level 2D drawing library used by *Gdk* to render widgets. *Cairo* provides a rich set of vector drawing features, supporting anti-aliasing, transparency, and 2D matrix transformations. The *Cairo.** packages provide a complete Ada binding to *Cairo*.

Gtk is the top level library. It is platform independent, and does all its drawing through calls to *Gdk* and *Cairo*. This is where the high-level widgets are defined. It also includes support for callbacks. Its equivalent in the GtkAda libraries are the *Gtk.** packages. It is made of a fully object-oriented hierarchy of widgets (see [Widgets Hierarchy](#)).

Since your application only calls GtkAda, it is fully portable, and can be recompiled as-is on other platforms:



Although the packages have been evolving a lot since the first versions of GtkAda, the specs are stabilizing now. We will try as much as possible to provide backward compatibility whenever possible.

Since GtkAda is based on GTK we have tried to stay as close to it as possible while using high-level features of the Ada language. It is thus relatively easy to convert external examples from C to Ada.

We have tried to adopt a consistent naming scheme for Ada identifiers:

- The widget names are the same as in C, except that an underscore sign (`_`) is used to separate words, e.g:

```
Gtk_Button   Gtk_Color_Dialog_Button
```

- When a name clashes with an Ada keyword, a `G` is prepended to the package name, e.g.:

```
Gtk.GEntry.Gtk_Entry
```

- The function names are the same as in C, ignoring the leading `gtk_` and the widget name, e.g:

```
gtk_window_present      => Gtk.Window.Present
gtk_toggle_button_set_active => Gtk.Toggle_Button.Set_Active
```

- Most enum types have been grouped in the `gtk-enums.ads` file
- Some features have been implemented as generic packages. These are the timeout and idle functions (see *Glib.Main.Generic_Sources*), and the data that can be attached to any object (see *Glib.Object.User_Data*). Type safety is ensured through these generic packages.
- Callbacks were the most difficult thing to interface with. These are extremely powerful and versatile, since the callbacks can have any number of arguments and may or may not return values. These are once again implemented as generic packages, that require more explanation ([Signal handling](#)).

WARNING: all the generic packages allocate some memory for internal structures, and call internal functions. This memory is freed by GTK itself, by calling some Ada functions. Therefore the generic packages have to be instantiated at library level, not inside a subprogram, so that the functions are still defined when GTK needs to free the memory.

WARNING Before any other call to the GtkAda library is performed, *Gtk.Main.Init* must be invoked first. Most of the time, this procedure is invoked from the main procedure of the application, in which case no use of GtkAda can be done during the application elaboration.

2.7 Widgets Hierarchy

All widgets in *GtkAda* are implemented as tagged types. They all have a common ancestor, called *Glib.Object.GObject*. All visual objects have a common ancestor called *Gtk.Widget.Gtk_Widget*.

The following table describes the list of objects and their inheritance tree. As usual with tagged types, all the primitive subprograms defined for a type are also known for all of its children. This is a very powerful way to create new widgets, as will be explained in *Creating new widgets in Ada*.

Although GTK was written in C its design is object-oriented, and thus GtkAda has the same structure. The following rules have been applied to convert from C names to Ada names: a widget *Gtk_XXX* is defined in the Ada package *Gtk.XXX*, in the file `gtk-xxx.ads`. This follows the GNAT convention for file names. For instance, the *Gtk_Text_View* widget is defined in the package *Gtk.Text_View*, in the file `gtk-text_view.ads`.

Note also that most of the documentation for GtkAda is found in the spec files themselves.

It is important to be familiar with this hierarchy. It is then easier to know how to build and organize your windows. Most widgets are demonstrated in the `gtkada_demo/` directory in the GtkAda distribution.



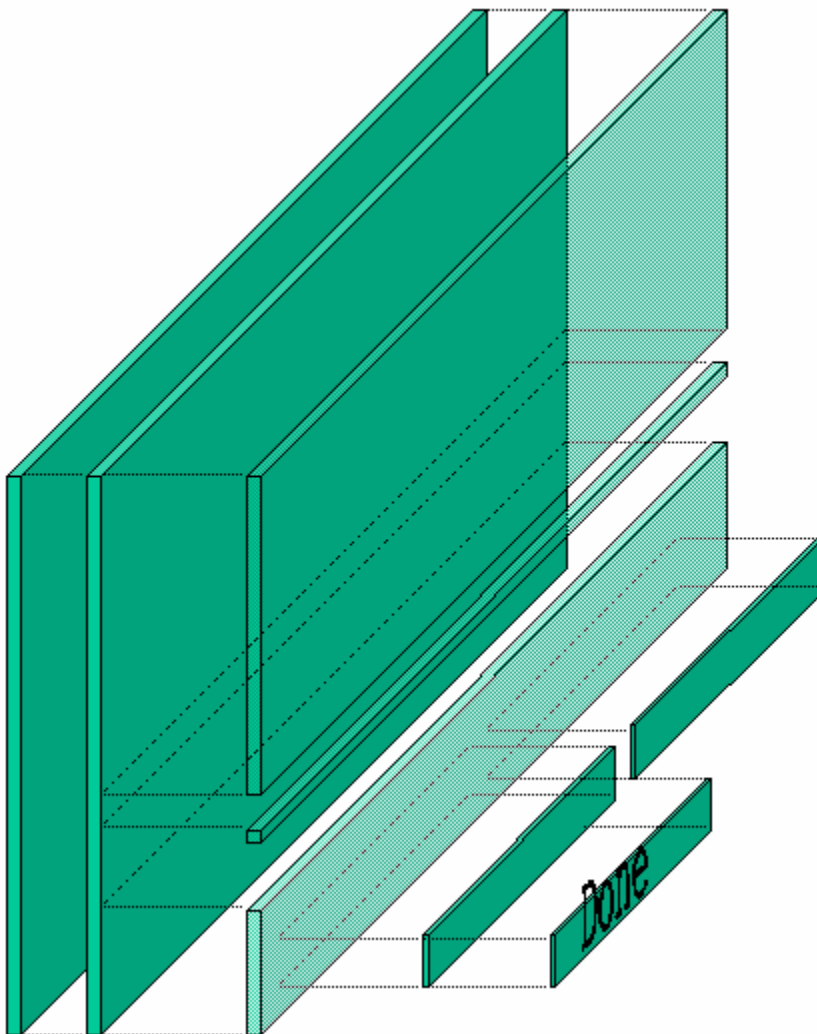
Fig. 1: Widgets Hierarchy

TRANSITIONING FROM GTKADA 3 TO GTKADA 4

GtkAda 4 binds GTK 4. The upstream [migration guide](#) describes the changes between GTK 3 and GTK 4; since GtkAda follows the C API closely (*gtk_window_present* becomes *Gtk.Window.Present*), its advice applies directly to Ada code.

HIERARCHICAL COMPOSITION OF A WINDOW

Interfaces in GtkAda are built in layers, as in Motif. For instance, a typical dialog is basically a `Gtk_Window`, that in turn contains a `Gtk_Box`, itself divided into two boxes and a `Gtk_Separator`, and so on.



Although this may seem more complicated than setting absolute positions for children, this is the simplest way to automatically handle the resizing of windows. Each container that creates a layer knows how it should behave when it is resized, and how it should move its children. Thus almost everything is handled automatically, and you don't have to do anything to support resizing.

All the containers are demonstrated in `gtkada_demo/`, in the GtkAda distribution. This should help you understand all

the parameters associated with the containers. It is very important to master these containers, since using the appropriate containers will make building interfaces a lot easier.

If you look at the widget hierarchy ([Widgets Hierarchy](#)), you can see that a `Gtk_Window` inherits from `Gtk_Bin`, and thus can have only one child. In most cases, the child of a `Gtk_Window` will thus be a `Gtk_Box`, which can have any number of children.

Some widgets in GtkAda itself are built using this strategy, starting with the very basic *Gtk_Button*.

For example, by default a `Gtk_Button` contains a `Gtk_Label`, which displays the text of the button (like ‘OK’ or ‘Cancel’).

However, it is easy to put something else in a button instead: *Set_Icon_Name* displays an icon, and *Set_Child* replaces the label with any widget of your choice.

SIGNAL HANDLING

In GtkAda, the interaction between the interface and the core application is done via signals. Most user actions on the graphical application trigger some signals to be emitted.

A signal is a message that an object wants to broadcast. It is identified by its name, and each one is associated with certain events which happen during the widget's lifetime. For instance, when the user clicks on a `Gtk_Button`, a 'clicked' signal is emitted by that button. More examples of signals can be found in the GtkAda reference manual.

It is possible to cause the application to react to such events by *connecting* to a signal a special procedure called a *handler* or *callback*. This handler will be called every time that signal is emitted, giving the application a chance to do any processing it needs. More than one handler can be connected to the same signal on the same object; the handlers are invoked in the order they were connected.

5.1 Predefined signals

Widgets, depending on their type, may define zero or more different signals. The signals defined for the parent widget are also automatically inherited; thus every widget answers many signals.

The easiest way to find out which signals can be emitted by a widget is to look at the GtkAda reference manual. Every widget will be documented there. The GtkAda RM explains when particular signals are emitted, and the general form that their handlers should have (although you can always add a *User_Data* if you wish, see below).

In general, your handlers should have the exact same profile that is documented (the GtkAda RM is automatically generated, so you can in fact find the same documentation directly in GtkAda's *.ads files).

However, if you connect to signals via the generic packages defined in `Gtk.Handlers` (see below), it is valid to pass a procedure that drops all arguments except the first one, i.e. the actual widget that emitted the signal. To get a better documented code, though, we recommend to always use the full profile for your handlers.

5.2 Connecting signals

There are currently two ways to connect widgets to signal handlers. One of them is much simpler to use, although it has some limited capabilities.

5.2.1 Connecting via the *On_** procedures

Each widget has a number of primitive operations (including inherited ones) for all the signals it might emit. In fact, for each signal there are two *On_<signal_name>* procedures that can be used to easily connect to the corresponding signal:

```
procedure Handler (Button : access Gtk_Button_Record'Class) is
begin
  Clicks_On_Button := Clicks_On_Button + 1;
end Handler;
```

```
Button.On_Clicked (Handler'Access);
```

The code above ensures that the procedure *Handler* is called whenever the button is clicked.

The *On_** procedures ensure that the profile of the handler is correct, and thus are type-safe.

The type of the first parameter to the handler is always the type where the signal is defined, not the type to which the handler is connected.

For instance, the “map” signal is defined for a *Gtk_Widget*. But if you connect this signal to a *Gtk_Button*, the first parameter of the handler is always of type *access Gtk_Widget_Record'Class*.

There is a second version of the *On_** procedures, which is used to pass a different object than the one the signal is connected to. In practice, this is the version that is used more often. For instance, clicking on a button will in general affect some other widget than the button itself, and you would typically pass the main window as a parameter to the handler. Here is an example, note how the type of the first parameter is different:

```
procedure Window_Handler (Win : access GObject_Record'Class) is
begin
  Clicks_On_Window := Clicks_On_Window + 1;
end Window_Handler;
```

```
Button.On_Clicked (Window_Handler'Access, Slot => Main_Window);
```

This subprogram also ensures that the handler is automaticall disconnected if the second object is destroyed.

5.2.2 Connecting via the *Gtk.Handlers* package

All signal handling work is performed internally using services provided by the *Gtk.Handlers* package. But this package can also be used directly by user applications.

This file is collection of several generic packages that need to be instantiated before you can actually start connecting handlers to widgets. A number of predefined instantiations are provided in *GtkAda.Handlers* to make it slightly easier.

Compared to the previous approach based on the *On_** procedures described above, this approach has a number of additional capabilities, at the cost of slightly more complex code:

- It is possible to retrieve a handle on the Widget/Signal/Handle tuple, so that you can later on disconnect the handler, or temporarily block the signal for instance.
- It is possible to pass additional user data to the handler. For instance, you could have a single handler connected to multiple check buttons. When you press any of the button, the handler is called and passes an additional integer to indicate which button was pressed. This is sometimes convenient, although it can often be avoided by creating new Ada tagged types derived from the standard GtkAda types.

- You have full control over the type of the first parameter to the handler. As discussed earlier, the *On_** subprograms force specific types (either a *GObject_Record* or the type on which the signal was defined). With the generic packages, you can avoid the often necessary type casts in the handler, although this approach does not guarantee more (or less) type safety.
- A very limited number of signals do not have a corresponding *On_** for circular dependency (or elaboration circularity) reasons. For those, you need to use the generic packages. However, we believe these signals are hardly ever used by user-level applications.

A short, annotated example of connecting signals follows; a complete example can be found in `create_file_selection.adb` (inside the `gtkada_demo/` directory). In our example, an application opens a file selector to allow the user to select a file. GtkAda provides a high-level widget called `Gtk_File_Selection` which can be used in this case:

```
declare
  Window : Gtk_File_Selection;
begin
  Gtk.File_Selection.Gtk_New (Window, Title => "Select a file");
end;
```

When the 'OK' button is pressed, the application needs to retrieve the selected file and then close the dialog. The only information that the handler for the button press needs is which widget to operate upon. This can be achieved by the following handler:

```
procedure OK (Files : access Gtk_File_Selection_Record'Class) is
begin
  -- Prints the name of the selected file.
  Ada.Text_IO.Put_Line ("Selected " & Get_Filename (Files));

  -- Destroys the file selector dialog
  Destroy (Files);
end Ok;
```

We now need to connect the object we created in the first part with the new callback we just defined. *Gtk.Handlers* defines four types of generic packages, depending on the arguments one expects in the callback and whether the callback returns a value or not. Note that you can not use an arbitrary list of arguments; these depend on the signal, as explained in the previous section.

In our example, since the callback does not return any value and does not handle any *User_Data* (that is, we don't pass it extra data, which will be specified at connection time), the appropriate package to use is *Gtk.Handlers.Callback*. We thus instantiate that package.

Remember that generic package instantiations in GtkAda must be present in memory at all times, since they take care of freeing allocated memory when finished. GtkAda generic package instantiations must therefore always be performed at the library level, and not inside any inner block:

```
package Files_Cb is new Handlers.Callback (Gtk_File_Selection_Record);
```

The *Files_Cb* package now provides a set of Connect subprograms that can be used to establish a tie between a widget and a handler. It also provides a set of other subprograms which you can use to emit the signals manually, although most of the time, the signals are simply emitted internally by GtkAda. We will not discuss the *Emit_By_Name* subprograms here.

The general form of handler, as used in *Gtk.Handlers*, expects some handlers that take two or three arguments: the widget on which the signal was applied, an array of all the extra arguments sent internally by GtkAda, and possibly some user data given when the connection was made.

This is the most general form of handler and it covers all the possible cases. However, it also expects the user to manually extract the needed values from the array of arguments. This is not always the most convenient solution. This

is why GtkAda provides a second package related to signals, *Gtk.Marshallers*.

The *Gtk.Marshallers* package provides a set of functions that can be used as callbacks directly for GtkAda, and that will call your application's handlers after extracting the required values from the array of arguments. Although this might sound somewhat complicated, in practice it simplifies the task of connecting signals. In fact, the techniques employed are similar to what is done internally by GTK in C. Because of the similarity of techniques, there is no overhead involved in using *Gtk.Marshallers* with Ada over the C code in GTK.

A set of functions *To_Marshaller* is found in every generic package in *Gtk.Handlers*. They each take a single argument, the name of the function you want to call, and return a handler that can be used directly in *Connect*.

The connection is then done with the following piece of code:

```
Files_Cb.Object_Connect
(Get_Ok_Button (Window), -- The object to connect to the handler
 "clicked",              -- The name of the signal
 Files_Cb.To_Marshaller (Ok'Access), -- The signal handler
 Slot_Object => Window);
```

Note that this can be done just after creating the widget, in the same block. As soon as it is created, a widget is ready to accept connections (although no signals will be emitted before the widget is shown on the screen).

We use *To_Marshaller* since our handler does not accept the array of arguments as a parameter, and we use the special *Object_Connect* procedure. This means that the parameter to our callback (Files) will be the *Slot_Object* given in *Object_Connect*, instead of being the button itself.

Compare the above code to the approach described in the first section, in particular when using the Ada 2012 dot notation:

```
Window.Get_Ok_Button.On_Clicked (Ok'Access, Window);
```

5.3 Handling user data

As described above, it is possible to define some data that is that passed to the callback when it is called. This data is called *user_data*, and is passed to the *Connect* or *Object_Connect* subprograms.

GtkAda will automatically free any memory it has allocated internally to store this user data. For instance, if you instantiated the generic package *User_Callback* with a String, it means that you want to be able to have a callback of the form:

```
procedure My_Callback (Widget : access Gtk_Widget_Record'Class;
                      User_Data : String);
```

and connect it with a call similar to:

```
Connect (Button, "Clicked", To_Marshaller (My_Callback'Access),
        User_Data => "any string");
```

GtkAda needs to allocate some memory to store the string (an unconstrained type). However, this memory is automatically freed when the callback is destroyed.

There are a few subtleties in the use of *user_data*, most importantly when the user data is itself a widget.

The following four examples do exactly the same thing: each creates two buttons, where clicking on the first one will destroy the second one. They all work fine the first time, while both buttons exist. However, some of them will fail if you press on the first button a second time.

Complete, compilable source code for these examples can be found in the distribution's `examples/user_data` directory, from which the code samples below are excerpted.

5.3.1 First case: simple user data

This code will fail: even after *Button2* is destroyed, the Ada pointer continues to reference memory that has been deallocated. The second call to *Destroy* will fail with a *Storage_Error*:

```
package User_Callback is new Gtk.Handlers.User_Callback
(Gtk_Widget_Record, Gtk_Widget);

procedure My_Destroy2
(Button : access Gtk_Widget_Record'Class; Data : Gtk_Widget) is
begin
  Destroy (Data);
end My_Destroy2;

begin
  User_Callback.Connect
    (Button1, "clicked",
     User_Callback.To_Marshaller (My_Destroy2'Access),
     Gtk_Widget (Button2));
end;
```

5.3.2 Second case: using *Object_Connect* instead

One of the solutions to fix the above problem is to use *Object_Connect* instead of *Connect*. In that case, GtkAda automatically takes care of disconnecting the callback when either of the two widgets is destroyed:

```
procedure My_Destroy (Button : access Gtk_Widget_Record'Class) is
begin
  Destroy (Button);
end My_Destroy;

begin
  Widget_Callback.Object_Connect
    (Button1, "clicked",
     Widget_Callback.To_Marshaller (My_Destroy'Access),
     Button2);
end;
```

5.3.3 Third case: manually disconnecting the callback

Using *Object_Connect* is not always possible. In that case, one of the possibilities is to store the *Id* of the callback, and properly disconnect it when appropriate. This is the most complex method, and very often is not applicable, since you cannot know for sure when the callback is no longer needed:

```
type My_Data3 is record
  Button, Object : Gtk_Widget;
  Id             : Handler_Id;
```

(continues on next page)

(continued from previous page)

```

end record;
type My_Data3_Access is access My_Data3;

package User_Callback3 is new Gtk.Handlers.User_Callback
  (Gtk_Widget_Record, My_Data3_Access);

procedure My_Destroy3
  (Button : access Gtk_Widget_Record'Class;
   Data   : My_Data3_Access) is
begin
  Destroy (Data.Button);
  Disconnect (Data.Object, Data.Id);
end My_Destroy3;

  Id : Handler_Id;
begin
  Data3 := new My_Data3' (Object => Gtk_Widget (Button1),
                        Button => Gtk_Widget (Button2),
                        Id      => (Null_Signal_Id, null));
  Id := User_Callback3.Connect
    (Button1, "clicked",
     User_Callback3.To_Marshaller (My_Destroy3'Access),
     Data3);
  Data3.Id := Id;
end;

```

5.3.4 Fourth case: setting a watch on a specific widget

GtkAda provides a function *Add_Watch*, that will automatically disconnect a callback when a given widget is destroyed. This is the function used internally by *Object_Connect*. In the example below, the callback is automatically disconnected whenever *Button2* is destroyed:

```

procedure My_Destroy2
  (Button : access Gtk_Widget_Record'Class; Data : Gtk_Widget) is
begin
  Destroy (Data);
end My_Destroy2;

  Id : Handler_Id;
begin
  Id := User_Callback.Connect
    (Button1, "clicked",
     User_Callback.To_Marshaller (My_Destroy2'Access),
     Gtk_Widget (Button2));
  Add_Watch (Id, Button2);
end;

```

STARTING AN APPLICATION WITH GTKADA

A GTK 4 program is built around an *application object*, of type *Gtk.Application.Gtk_Application*. The application object initializes GTK, loads resources such as *Gtk.Builder* UI definitions, then runs the main loop. It keeps track of the application's windows, and ends the program when the last of them is closed (or another method is used to tell it to hold, see below). It can also make sure that only one instance of the application runs at a time, if this behavior is desired.

You do not need to call *Gtk.Main.Init* or run a main loop yourself, as the application object does both.

6.1 The main procedure

See an example of main procedure which creates an application:

```
with Ada.Command_Line;
with Glib;              use Glib;
with Glib.Application; use Glib.Application;
with Gtk.Application;   use Gtk.Application;
with UG_Application;

procedure UG_Hello is
  App      : Gtk_Application;
  Status   : Gint;
begin
  Gtk_New (App, "com.example.hello", G_Application_Flags_None);
  App.On_Activate (UG_Application.On_Activate'Access);

  -- Run the application until its last window is closed
  Status := App.Run;
  Ada.Command_Line.Set_Exit_Status (Ada.Command_Line.Exit_Status (Status));
end UG_Hello;
```

The first argument to *Gtk_New* is the *application ID*. It names the application for the desktop and the session bus, and it must be a name such as `com.example.hello` (see *Glib.Application.Id_Is_Valid* for the exact rules). If you pass an empty string, the application cannot be single-instance (see *Command line and single instance*).

The second argument is a set of flags of type *Glib.Application.GApplication_Flags*, combined with `or`. Most applications should use *G_Application_Flags_None*.

Run initializes GTK, emits “startup” and then “activate”, and runs the main loop. This loop returns when the application ends, and its result is the exit status of the program. This example reads the command line from *Ada.Command_Line*.

6.2 Creating the main window

The application shows the user interface in the “activate” signal handler. This handler receives the application as a *Glib.Application.Gapplication*, which you can convert to a *Gtk_Application*:

```
procedure On_Activate (Self : access Gapplication_Record'Class) is
  App    : constant Gtk_Application := Gtk_Application (Self);
  Window : Gtk_Application_Window;
  Button : Gtk_Button;
begin
  -- "activate" is emitted again when the user launches the application
  -- while it is already running: show the existing window instead of
  -- creating a new one.
  if App.Get_Active_Window /= null then
    App.Get_Active_Window.Present;
    return;
  end if;

  Gtk_New (Window, App);
  Window.Set_Title ("Hello");
  Window.Set_Default_Size (300, 200);
  Window.On_Close_Request (On_Close_Request'Access);

  Gtk_New (Button, "Quit");
  Button.On_Clicked (On_Quit'Access, Slot => Window);
  Window.Set_Child (Button);

  Window.Present;
end On_Activate;
```

A few points to note:

- The window is a *Gtk.Application_Window.Gtk_Application_Window*. It is owned by the application. Similarly, you can make a plain *Gtk.Window.Gtk_Window* become owned by the application, with *Gtk.Application.Add_Window*.
- Call *Gtk.Window.Present* to show the window and brings it to the front. All widgets are visibly by default in GTK4. To hide a widget, call *Gtk.Widget.Set_Visible* with *False*.
- A window is a container that has only one child, set with *Gtk.Window.Set_Child*. To arrange several widgets in a window, make its child a container, typically a *Gtk.Box.Gtk_Box* (see *Hierarchical composition of a window*).
- The “activate” signal can be emitted more than once in the same process, for instance when the user launches an application that is already running (see *Command line and single instance*).

For a larger example, see the GtkAda demo: the code in `gtkada_demo/gtkada_demo.adb` creates the application, the contents are built in the subprogram *On_Activate* in `gtkada_demo/main_windows.adb`.

6.3 Ending the application

Run returns when the application has no windows left.

To close a window programmatically, call one of these subprograms:

- *Gtk.Window.Close*: this does what the close button of the title bar does: it emits “close-request” on the window, giving the window a chance to intercept the request, for instance to display a dialog to the user. See below.
- *Gtk.Window.Destroy* destroys the window without emitting “close-request”.

The Quit button of the example calls *Close*:

```
procedure On_Quit (Window : access GObject_Record'Class) is
begin
    Gtk_Window (Window).Close;
end On_Quit;
```

To stop a window from closing, connect a handler to “close-request”. For instance to display a confirmation dialog. The window is kept open if the handler for this signal returns *True*:

```
function On_Close_Request
(Self : access Gtk_Window_Record'Class) return Boolean is
begin
    -- Returning True stops the window from closing
    return Has_Unsaved_Changes;
end On_Close_Request;
```

To hide a window when it is closed, rather than destroying it, call *Gtk.Window.Set_Hide_On_Close*. A hidden window still belongs to the application, which therefore keeps running.

Glib.Application.Quit ends the application at once, even if it still has windows. The windows do not get “close-request”.

Glib.Application.Hold keeps the application running when it has no windows. Each call must be matched by a call to *Glib.Application.Release*. This is useful for windowless applications or applications which need to keep running background tasks.

6.4 Startup and shutdown

Besides “activate”, the application emits two signals that you can connect to with *Glib.Application.On_Startup* and *Glib.Application.On_Shutdown*:

- “startup” is emitted once, before the first “activate”. GTK is initialized by then. It is a good place for one-off setup operations, such as loading your CSS files.
- “shutdown” is emitted once, after the main loop has ended and just before *Run* returns.

Both are emitted only in the primary instance (see the next section).

6.5 Command line and single instance

When the application has an ID, and the desktop provides a D-Bus session bus, (as most Linux desktops do), only one instance of the application runs at a time. The first process to start becomes the *primary instance*. When the application is started again, the new process asks the primary instance to emit “activate” (or “command-line”, see below) and then exits. To support this, the “activate” handler must be able to work with a window that already exists.

The flags passed to *Gtk_New* can change this behaviour. The most interesting ones are:

G_Application_Flags_None

The default.

G_Application_Non_Unique

Deactivate the “single window” concept: every process is a new instance.

G_Application_Handles_Command_Line

The primary instance emits “command-line” instead of “activate”. The handler receives the command line arguments, including those passed to a second instance.

To handle the command line, use *G_Application_Handles_Command_Line* and connect a handler to “command-line”:

```
Gtk_New (App, "com.example.editor", G_Application_Handles_Command_Line);
App.On_Activate (On_Activate'Access);
App.On_Command_Line (On_Command_Line'Access);
```

The handler reads the arguments from its *Command_Line* parameter. It must emit “activate” itself in order for a window to be shown, since the application no longer takes care of it. The value that the handler returns is the exit status of the process that received the command line. That process might not be the primary instance:

```
function On_Command_Line
  (Self      : access Gapplication_Record'Class;
   Command_Line : not null access Gapplication_Command_Line_Record'Class)
  return Gint
is
  Args : GNAT.Strings.String_List := Command_Line.Get_Arguments;
begin
  -- Args (Args'First) is the name of the program
  for J in Args'First + 1 .. Args'Last loop
    Open_File (Args (J).all);
  end loop;

  for Arg of Args loop
    GNAT.Strings.Free (Arg);
  end loop;

  Self.Activate;
  return 0; -- The exit status of the command line
end On_Command_Line;
```

Glib.Application.Gapplication_Command_Line also passes the current working directory of the process that received the command line (*Get_Cwd*), along with its environment (*Getenv*), if the application was created with the *G_Application_Send_Environment* flag. Prefer these rather than *Ada.Directories* and *Ada.Environment_Variables*, which only relate to the primary instance.

RESOURCE FILES

Resource files let you parametrize aspects of the widgets in a GtkAda application without having to recompile it.

In this file, it is possible to specify visual characteristics of widgets, such as their colors and fonts. Under X, the *xfontsel* command allows you to easily select a font. The FontSelection widget is also a simple way to select fonts.

Here is an example of a resource file:

```
# application.rc
#
# resource file for "Application"

# Buttons style
style "button"
{
  # BackGround Colors
  #           Red   Green  Blue
  bg[PRELIGHT] = { 0.0, 0.75, 0.0 } # Green when the mouse is on
                                     # the button
  bg[ACTIVE]   = { 0.75, 0.0, 0.0 } # Red on click
  # ForeGround Colors
  #           Red   Green  Blue
  fg[PRELIGHT] = { 1.0, 1.0, 1.0 } # White when the mouse is on
                                     # the button
  fg[ACTIVE]   = { 1.0, 1.0, 1.0 } # White on click
}

# All the buttons will have the style "button"
widget_class "*GtkButton*" style "button"

# Text style
style "text"
{
  font = "-adobe-courier-medium-r-normal-*15-*-*-*-*-*"
  text[NORMAL] = { 0.0, 0.0, 0.0 } # black
  fg[NORMAL]   = { 0.0, 0.0, 0.0 } # black
  base[NORMAL] = { 1.0, 1.0, 1.0 } # white : background color
}

# All Gtk_Text will have the "text" style
widget_class "*GtkText" style "text"
```


MEMORY MANAGEMENT

GtkAda takes care of almost all the memory management for you. Here is a brief overview of how this works, you'll have to check the sources if you want more detailed information. GTK (the C library) does its own memory management through reference counting, i.e. any widget is destroyed when it is no longer referenced anywhere in the application.

In GtkAda itself, a 'user_data' is associated with each object allocated by a *Gtk_New* procedure. A 'destroy' callback is also associated, to be called when the object to which the user_data belongs is destroyed. Thus, every time a C object is destroyed, the equivalent Ada structure is also destroyed.

Concerning widgets containing children, every container holds a reference to its children, whose reference counting is thus different from 0 (and generally 1). When the container is destroyed, the reference of all its children and grandchildren is decremented, and they are destroyed in turn if needed. So the deallocation of a widget hierarchy is also performed automatically.

TASKING WITH GTKADA

Note that GTK under Windows does not interact properly with threads, so the only safe approach under this operating system is to perform all your GTK calls in the same task.

On other platforms, the Glib library can be used in a task-safe mode by calling *Gdk.Threads.G_Init* and *Gdk.Threads.Init* before making any other Glib/Gdk calls. Gdk routines may then be called simultaneously by multiple tasks, thanks to task-safe construction of Gdk's internal data structures. However, Gdk objects such as hash tables are not automatically protected, so it is the application's responsibility to prevent simultaneous access to user-defined objects (e.g. by using protected objects).

When Gdk is initialized to be task-safe, GtkAda becomes task aware. There is a single global lock that you must acquire with *Gdk.Threads.Enter* before making any Gdk/Gtk call, and which you must release with *Gdk.Threads.Leave* afterwards.

Gtk.Main.Main should be called with the lock acquired (see example below), ensuring that all the functions executed in the task that started the main loop do not need to protect themselves again.

Beware that the GtkAda main loop (*Gtk.Main.Main*) can only be run inside one specific task. In other words, you cannot call *Gtk.Main.Main* from any task other than the one that started the outer level main loop.

Note that *Gdk.Threads* assumes that you are using a tasking run time that maps Ada tasks to native threads.

A minimal main program for a tasking GtkAda application looks like:

```
with Gdk.Threads;
with Gtk.Main;
with Gtk.Enums; use Gtk.Enums;
with Gtk.Window; use Gtk.Window;

procedure GtkAda_With_Tasks is
  Window : Gtk_Window;
begin
  Gdk.Threads.G_Init;
  Gdk.Threads.Init;
  Gtk.Main.Init;

  Gtk_New (Window, Window_Toplevel);
  Show (Window);

  Gdk.Threads.Enter;
  Gtk.Main.Main;
  Gdk.Threads.Leave;
end GtkAda_With_Tasks;
```

Callbacks require a bit of attention. Callbacks from GtkAda (signals) are made within the GtkAda lock. However, callbacks from Glib (timeouts, IO callbacks, and idle functions) are made outside of the GtkAda lock. So, within a signal handler you do not need to call *Gdk.Threads.Enter*, but within the other types of callbacks, you do.

PROCESSING EXTERNAL EVENTS

It often happens that your application, in addition to processing graphical events through the GtkAda main loop, also needs to monitor external events. This is the case if, for instance, you are running external processes and need to display their output, or if you are listening to incoming data on a socket. If you implement your own main loop to poll for these external events and then invoke the GUI, the GUI will enter its main loop and not return control back to you.

There are several ways to handle this situation:

- The cleanest solution, especially if you intend to make the GUI a major part of your application (as opposed to just popping up a few dialogs here and there), would be to use the GTK main loop as the infinite loop, instead of yours.

You can then use GTK ‘idle callbacks’ (which are called every time the GTK loop is not busy processing graphical events) or ‘timeout callbacks’ (which are called every *n* milliseconds), and in those callbacks do the work you were doing before in your own main loop (that assumes the check is relatively fast, otherwise the GUI will be frozen during that time). Such callbacks are created through the *Glib.Main* package.

- Another approach is to not start the GTK main loop, but to check periodically whether there are some events to be handled. See the subprogram *Glib.Main.Main_Context_Iteration*.

This second approach is not necessarily recommended, since you would basically duplicate code that’s already in GTK to manage the main loop, and you also get finer control using idle and timeout callbacks

OBJECT-ORIENTED FEATURES

GtkAda has been designed from the beginning to provide a full object oriented layer over GTK. This means that features such as type extension and dynamic dispatching are made available through the standard Ada language.

This section will describe how things work, how you can extend existing widgets, and even how to create your own widgets.

11.1 General description of the tagged types

11.1.1 Why should I use object-oriented programming ?

Every widget in the *Gtk.** packages in GtkAda is a tagged type with a number of primitive subprograms that are inherited by all of its children. Tagged types in Ada make it possible to perform safe, automatic type conversions without using explicit casts (such as is necessary when coding in C). It is also possible for the compiler to verify whether or not these type conversions are valid. Most errors are found at compile time, which leads to a safer and more robust application.

As a further example, imagine a table that has been populated by some widgets. It is possible to query for this table's children and operate on these widgets without knowing details about their type, their creator, and so on—the tagged objects that are returned contain all the information necessary. It becomes possible to use dynamic dispatching without ever having to cast to a known type.

Modifying a standard widget to draw itself differently or display different data is easy using tagged types. Simply create a new type that extends the current one (see the section *Using tagged types to extend GTK widgets* below).

Creating a new reusable widget from scratch is also possible. Create a new tagged type and specify properties of the widget—such as how it is to draw itself and how it should react to events. See the section *Creating new widgets in Ada* below.

Object oriented programming through the use of Ada tagged types makes GtkAda a very powerful, flexible, and safe tool for designing graphical interfaces.

11.1.2 Type conversions from C to Ada widgets

There are three kinds of widgets that you can use with GtkAda:

- *Ada widgets*: These are widgets that are written directly in Ada, using the object oriented features of GtkAda
- *Standard widgets*: These are the widgets that are part of the standard GTK and GtkAda distributions. This include all the basic widgets you need to build advanced interfaces.
- *third party C widgets* These are widgets that were created in C, and for which you (or someone else) created an Ada binding. This is most probably the kind of widgets you will have if you want to use third party widgets.

GtkAda will always be able to find and/or create a valid tagged type in the first two cases, no matter if you explicitly created the widget or if it was created automatically by GTK. For instance, if you created a widget in Ada, put it in a table, and later on extracted it from the table, then you will still have the same widget.

In the third case (third party C widgets), GtkAda is not, by default, able to create the corresponding Ada type.

The case of third party C widgets is a little bit trickier. Since GtkAda does not know anything about them when it is built, it can't magically convert the C widgets to Ada widgets. This is your job to teach GtkAda how to do the conversion.

We thus provide a 'hook' function which you need to modify. This function is defined in the package **Glib.Type_Conversion_Hooks**. This function takes a string with the name of the C widget (ex/ "GtkButton"), and should return a newly allocated pointer. If you don't know this type either, simply return **null**.

11.2 Using tagged types to extend GTK widgets

With this toolkit, it's possible to associate your own data with existing widgets simply by creating new types. This section will show you a simple example, but you should rather read the source code in the `gtkada_demo/` directory where we used this feature instead of using `user_data` as is used in the C version:

```
type My_Button_Record is new Gtk_Button_Record with record
  Count : Natural := 0;
  -- whatever data you want to associate with your button
end record;
type My_Button is access all My_Button_Record'Class;
```

With the above statements, your new type is defined. Every function available for *Gtk_Button* is also available for *My_Button*. Of course, as with every tagged type in Ada, you can create your own primitive functions with the following prototype:

```
procedure My_Primitive_Func (Myb : access My_Button_Record);
```

To instantiate an object of type *My_Button* in your application, with *Myb* declared as a *My_Button*, do the following:

```
Myb := new My_Button_Record;
Initialize (Myb, Label => "Hello"); -- from Gtk.Button
```

The first line creates the Ada type, whereas the *Initialize* call actually creates the C widget and associates it with the Ada type.

11.3 Creating new widgets in Ada

With GtkAda, you can create widgets directly in Ada. These new widgets can be used directly, as if they were part of GTK itself.

Creating new widgets is a way to create reusable components. You can apply to them the same functions as you would for any other widget, such as *Show*, *Hide*, and so on.

This section will explain how to create two types of widgets: composite widgets and widgets created from scratch. An example of the latter is provided with GtkAda, in `gtkada_demo/create_custom_widget.adb`. Please also refer to the GTK tutorial, which describes the basic mechanisms that you need to know to create a widget.

11.3.1 Creating composite widgets

A composite widget is a widget that does not do much by itself. Rather, this is a collection of subwidgets grouped into a more general entity. For instance, among the standard widgets, *Gtk_Font_Dialog_Button* belongs to this category.

The good news is that there is nothing special to know. Just create a new tagged type, extending one of the standard widgets (or even another of your own widgets), provide a *Gtk_New* function that allocates memory for this widget, and call the *Initialize* function that does the actual creation of the widget and the subwidgets. There is only one thing to do: *Initialize* should call the parent class's *Initialize* function, to create the underlying C widget.

11.3.2 Creating widgets from scratch

Creating a working widget from scratch requires a certain level of familiarity with the GtkAda signal mechanism and entails much work with low level signals. This is therefore not an activity recommended for novice GtkAda programmers.

Creating a widget from scratch is what you want to do if your widget should be drawn in a special way, should create and emit new signals, or otherwise perform differently than pre-existing widgets. The example in `gtkada_demo/create_custom_widget.adb` is a small widget that computes its own size and draws itself.

Since we are creating a totally new widget from scratch, with potentially its own signals, we need to do slightly more work. In particular, we need to provide a function *Get_Type* similar to what all the predefined widgets provide:

```
with Glib.Properties.Creation;   use Glib.Properties.Creation;
with Glib.Object;               use Glib.Object;
with Gtk.Scrollable;
with System;

package body My_Widgets is
  type My_Widget_Record is new Gtk_Button_Record with record
    ...
  end record;
  type My_Widget is access all My_Widget_Record'Class;

  Klass : aliased Ada_GObject_Class := Uninitialized_Class;

  PROP_H_ADJ : constant Property_Id := 1;
  PROP_V_ADJ : constant Property_Id := 2;
  -- internal identifier for our widget properties

  procedure Class_Init (Self : GObject_Class);
  pragma Convention (C, Class_Init);

  procedure Class_Init (Self : GObject_Class) is
  begin
    -- Set properties handler
    Set_Properties_Handlers (Self, Prop_Set'Access, Prop_Get'Access);

    -- Override inherited properties
    Override_Property (Self, PROP_H_ADJ, "hadjustment");
    Override_Property (Self, PROP_V_ADJ, "vadjustment");

    -- Override some the inherited methods (how to draw the widget)
    Set_Default_Snapshot_Handler (Self, On_Snapshot'Access);
```

(continues on next page)

(continued from previous page)

```

    -- Override the primitives to compute the widget size
    Set_Default_Measure_Handler (Self, ...);
end Class_Init;

function Get_Type return GType is
    Info : access GInterface_Info;
begin
    if Initialize_Class_Record
        (Ancestor      => Gtk.Button.Get_Type,
         Class_Record => Klass'Access,
         Type_Name     => "My_Widget",
         Class_Init    => Class_Init)
    then
        begin
            -- Add interfaces if needed
            Info := new GInterface_Info'(null, null, System.Address);
            Add_Interface (Klass, Gtk.Scrollable.Get_Type, Info);
        end if;
        return Klass.The_Type;
    end Get_Type;
end My_Widgets;

```

You should also create the usual functions `Gtk_New` and `Initialize`:

```

procedure Gtk_New (Self : out My_Widget) is
begin
    Self := new My_Widget_Record; -- create the Ada wrapper
    Initialize (Self);
end Gtk_New;

procedure Initialize (Self : not null access My_Widget_Record'Class) is
begin
    G_New (Self, Get_Type); -- allocate the C widget, unless done

    -- Initialize parent fields.

    Gtk.Button.Initialize (Self);

    -- Initialization of the Ada types

    Self.Field1 := ...;
end Initialize;

```

In the above example, the new part is the `Get_Type` subprogram. It takes three or four arguments:

- *Ancestor* This is the `GType` for the ancestor that is being extended.
- *Signals* This is an array of string access containing the name of the signals you want to create. For instance, you could create `Signals` with:

```
Signals : Gtkada.Types.Chars_Ptr_Array := "bullseye" + "missed";
```

This will create two signals, named “bullseye” and “missed”, whose callbacks’ arguments can be specified with the fourth parameter.

- *Class_Record* Every widget in C is associated with three records:

- An instance of *GType*, which is a unique identifier (integer) for all the class of widgets defined in the framework. This description also contains the name of the class, its parent type, the list of interfaces it inherits, and all the signals it defines.

These *GType* are often created early on when an application is launched, and provide the basic introspection capabilities in a GTK application.

In Ada, this type is created by the function *Get_Type* in the example above (which is why we need to add the interface in that function).

- An instance of *GObject_Class*, which contains implementation details for the class, defines the default signal handlers (how to draw a widget of the class, how to handle size negotiation,...), and defines any number of properties that can be configured on the widget (properties are a generic interface to access the components of a composite widget, as well as some of its behavior – they can be modified through introspection for instance in a GUI builder).

Such a type is created automatically by GTK just before it creates the first instance of that widget type. It will then immediately call the *Class_Init* function that might have been passed to *Glib.Object.Initialize_Class_Record*. At that point, you can add your own new properties, or override the default signal handlers to redirect them to your own implementation.

- A class instance record; there is one such record for each widget of that type. In GtkAda, the ‘instance record’ is simply your tagged type and its fields. It is created when you call any of the *Gtk_New* functions.

- *Parameters* This fourth argument is in fact optional, and is used to specify which kind of parameters each new signal is expecting. By default (ie if you don’t give any value for this parameter), all the signals won’t expect any argument, except of course a possible user_data. However, you can decide for instance that the first signal (“bullseye”) should in fact take a second argument (say a Gint), and that “missed” will take two parameters (two Gints).

Parameters should thus contain a value of:

```
(1 => (1 => Gtk_Type_Int, 2 => Gtk_Type_None),
 2 => (1 => Gtk_Type_Int, 2 => Gtk_Type_Int));
```

Due to the way arrays are handled in Ada, each component must have the same number of signals. However, if you specify a type of *Gtk_Type_None*, this will in fact be considered as no argument. Thus, the first signal above has only one parameter.

Note also that to be able to emit a signal such as the second one, ie with multiple arguments, you will have to extend the packages defined in *Gtk.Handlers*. By default, the provided packages can only emit up to one argument (and only for a few specific types). Creating your own *Emit_By_Name* subprograms should not be hard if you look at what is done in *gtk-marshallers.adb*. Basically, something like:

```
procedure Emit_With_Two_Ints
  (Object : access Widget_Type'Class;
   Name   : String;
   Arg1   : Gint;
   Arg2   : Gint);
pragma Import (C, Emit_With_Two_Ints,
  "gtk_signal_emit_by_name");

Emit_With_Two_Ints (Gtk.Get_Object (Your_Widget),
  "missed" & ASCII.NUL, 1, 2);
```

will emit the “missed” signal with the two parameters 1 and 2.

BINDING NEW WIDGETS

GtkAda comes with a Perl script to help you create a binding to a C widget (this is the script we have used ourselves). This will not fully automate the process, although it should really speed things up. You will probably need less than 15 min to create a new binding once you will get used to the way GtkAda works. Note that your C file should have the same format as is used by GTK itself.

To get started on a new binding, launch the script `contrib/binding.pl` as follows:

```
$ touch gtk-button.ads  
$ binding.pl ../include/gtk/gtkbutton.h > temporary
```

This dumps several kind of information on the standard output:

- List of subprograms defined in the `.h` file. Their documentation is also added, since `binding.pl` will parse the `.c` file as appropriate.
- List of properties and signals for the widget
- Tentative bodies for the subprograms These will often need adjustments, but provide a good start

You can also use this script to update existing bindings:

```
$ binding.pl ../include/gtk/*.h
```


DEBUGGING GTKADA APPLICATIONS

This chapter presents a number of technics that can be used when debugging GtkAda applications. First, the standard tools to debug Ada applications can be used:

Compile with -g

You should almost always include debugging information when compiling and linking your code. This gives you the possibility to use the debugger. See below the variable `GDK_DEBUG` for how to disable grabs.

bind with -E

Using this argument on the *gnatbind* or *gnatmake* command line will force the compiler to include backtraces when an exception is raised. These backtraces can be converted to symbolic backtraces by using the *addr2line* tool.

Link with -lgmem

Using this switch gives access to the *gnatmem* tool, that helps you to detect memory leaks or doubly-deallocated memory. The latter often results in hard-to-fix `Storage_Error` exceptions. See the GNAT User's guide for more information.

There are also a number of technics specific to GtkAda or GTK applications. For most of them, you might need to recompile these libraries with the appropriate switches to get access to the extended debugging features.

Use the `-sync` switch

Under unix systems, all applications compiled with GTK automatically support this switch, which forces events to be processed synchronously, thus making it easier to detect problems as soon as they happen. This switch is not relevant to Windows systems.

break on `g_log`

In the debugger, it is often useful to put a breakpoint on the glib function *g_log*. When GTK is linked dynamically, you will need to first start your application with *begin*, then put the breakpoint and continue the application with *cont*. This helps understand internal errors or warnings reported by GTK and glib

compile glib with `-disable-mem-pools`

Glib, the underlying layer that provides system-independent services to GTK, has an extensive and optimized system for memory allocation. Bigger chunks of Memory are allocated initially, and then subdivided by glib itself. Although this is extremely performant, this also make the debugging of memory-related problems (`storage_error`) more difficult. Compiling with the above switch forces glib to use the standard `malloc()` and `free()` system calls. On GNU/Linux systems, it might be useful to set the variable `MALLOC_CHECK_` to 1 to use error-detecting algorithms (see the man page for `malloc()`).

compile glib and GTK with `-enable-debug=yes`

It is recommended that you specify this switch on the *configure* command line when compiling these two libraries. In addition to compiling the libraries with debugging information for the debugger, additional runtime debug options (controllable via environment variables) become available. Specifying `-enable-debug=no` is not recommended for production releases (see glib or GTK documentation for details).

For these three variables, the possible values are given below. These are lists of colon-separated keywords. You can choose to remove any of these value from the variable

GOBJECT_DEBUG=objects:signals

This sets up the debugging output for glib. The value `objects` is probably the most useful, and displays, on exit of the application, the list of unfreed objects. This helps detect memory leaks. The second value `signals` will display all the signals emitted by the objects. Note that this results in a significant amount of output.

GDK_DEBUG=updates:nograbs:events:dnd:misc:xim:colormap:gdkrgb:gc:pixmap:image:input:cursor

This sets up the debugging output for gdk. The most useful value is `nograbs`, which prevents the application from ever grabbing the mouse or keyboards. If you don't set this, it might happen that the debugger becomes unusable, since you don't have access to the mouse when the debugger stops on a breakpoint. Another simpler solution is to debug remotely from another machine, in which case the grabs won't affect the terminal on which the debugger is running.

GTK_DEBUG=misc:plugsocket:text:tree:updates:keybindings

This sets up the debugging output for GTK. Almost all of these values are mostly for internal use by GTK developers, although `keybindings` might prove useful sometimes.

keyboard input buffers/freezes while debugging on Windows

GTK on Windows installs a low level hook to handle snapping via `set_up_low_level_keyboard_hook`. When the GTK application is stopped by the debugger, the GTK handler will become irresponsive and freeze the Keyboard input for `LowLevelHooksTimeout` ms. Workarounds: - Set `HKEY_CURRENT_USERControlPanelDesktopLowLevelHooksTimeout` registry to 30ms and restart the session. - Recompile GTK after removing `set_up_low_level_keyboard_hook` calls. See <https://gitlab.gnome.org/GNOME/gtk/-/issues/2015> for more details.

Import the C function `ada_gtk_debug_get_ref_count`

This function has the following Ada profile:

```
function Ref_Count (Object : System.Address) return Guint
with Import, Convention => C,
    External_Name => "ada_gtk_debug_get_ref_count";
```

and should be called on a widget in a manner similar to:

```
Count := Ref_Count (Get_Object (Widget));
```

and returns the internal reference counter for the widget. When this counter reaches 0, the memory allocated for the widget is automatically freed.

This is mostly a debugging aid for people writting their own containers, and shouldn't generally be needed. You shouldn't rely on the internal reference counter in your actual code, which is why it isn't exported by default in GtkAda.

HOW TO REPORT BUGS

GtkAda is a mature, stable toolkit that is heavily and widely used on a variety of platforms. We test GtkAda using an Ada version of the `gtkada_demo.c` file found in the GTK distribution. For code examples that demonstrate the use of this toolkit, look within the `gtkada_demo/` directory.

There are two kinds of problems you can encounter:

- If the GTK library itself was compiled with warnings turned on, you may get some warning messages, mainly because of types problems. These warnings should not appear, as we have tried to be as type safe as possible in this package. To know exactly where the problem is, compile your program with debug information, run `gdb`, and set a breakpoint on the function `g_log`. Then run your program as usual, using the `run` command. Then send us the result of the `where` command. Here is a summary:

```
$ gprbuild -Pyour_project.gpr -g
$ gdb <your_program_name>
(gdb) break main
(gdb) run
(gdb) break g_log
(gdb) continue
....
(gdb) where
```

- In some (hopefully) rare cases, you can even get a segmentation fault within GTK. That means there is definitely something wrong either in your program or in the toolkit. Please check your program carefully and, if you think this is a problem in GtkAda itself, send us an e-mail.

If you are a supported user of GNAT, send mail to <mailto:report@gnat.com> to report errors, otherwise send mail to the GtkAda list (<mailto:gtkada@lists.adacore.com>) explaining exactly what you are doing, what is the expected result and what you actually get. Please include the required sources to reproduce the problem, in a format usable by *gnatchop* (basically, insert all the required sources at the end of the mail). Please try to provide as small as possible a subset of your sources.

Of course, we will welcome any patch you can provide, so that this toolkit may be as useful as possible.

SYSTEM PACKAGE DEPENDENCIES

On Linux, GtkAda requires a set of system development packages to be installed. This set depends on the Linux distribution. The following sections provide lists of packages which need to be installed on particular distributions. The installation procedure of GtkAda will fail if some of packages are missing.

15.1 Red Hat Enterprise Linux 7 & 8

- glibc-devel
- glibc-headers
- kernel-headers
- libX11-devel
- libXau-devel
- libXcomposite-devel
- libXcursor-devel
- libXdamage-devel
- libXext-devel
- libXfixes-devel
- libXi-devel
- libXinerama-devel
- libXrandr-devel
- libXrender-devel
- libuuid-devel
- libxcb-devel
- xorg-x11-proto-devel

15.2 Red Hat Enterprise Linux 9

- glibc-devel
- libXcomposite-devel
- libXcursor-devel
- libXdamage-devel
- libXinerama-devel
- libXrandr-devel
- libXtst-devel
- libglvnd-devel

15.3 Ubuntu 18.04 & 20.04

- libc6-dev
- libdrm-dev
- libegl1-mesa-dev
- libgl1-mesa-dev
- libgles1
- libglvnd-core-dev
- libglvnd-dev
- libopengl0
- libpthread-stubs0-dev
- libwayland-dev
- libx11-dev
- libx11-xcb-dev
- libxau-dev
- libxcb-dri2-0-dev
- libxcb-dri3-dev
- libxcb-glx0-dev
- libxcb-present-dev
- libxcb-randr0-dev
- libxcb-render0-dev
- libxcb-shape0-dev
- libxcb-shm0-dev
- libxcb-sync-dev
- libxcb-xfixes0-dev
- libxcb1-dev

- libxcomposite-dev
- libxcursor-dev
- libxdamage-dev
- libxdmcp-dev
- libxext-dev
- libxf86dev
- libxi-dev
- libxinerama-dev
- libxrandr-dev
- libxrender-dev
- libxshmfence-dev
- libxtst-dev
- libxxf86vm-dev
- mesa-common-dev
- uuid-dev
- x11proto-composite-dev
- x11proto-core-dev
- x11proto-damage-dev
- x11proto-dev
- x11proto-fixes-dev
- x11proto-input-dev
- x11proto-randr-dev
- x11proto-xext-dev
- x11proto-xf86vidmode-dev
- x11proto-xinerama-dev

USING GTKPARASITE TO INSPECT AND MODIFY RUNNING GTKADA APPLICATIONS

GtkParasite is a tool that enables one to inspect running GTK applications. It's usable with GtkAda application without any Ada specific recommendations.

When it is installed, you have to make sure that your application will be able to access GtkParasite library. This means either :

- Linking against the GtkParasite library during compilation of your application.
- Make sure that the dynamic library file is accessible at runtime, for example by adding its path to the LD_LIBRARY_PATH environment variable before running your application.

Then, to run GtkParasite with your application:

```
$ GTK_MODULES=gtkparasite yourapp
```

You can find more information about how to use GtkParasite on it's web page here : [GtkParasite Home](#)

BIBLIOGRAPHY

We recommend the following documents. Most of them were written with C in mind, but should be easily adapted after you've read the rest of this document.

- [1] “GTK+/Gnome Application Development” – Havoc Pennington This book, by one of the main authors of the the GNOME environment, describes in detail some of the inner mechanisms of GTK, including signal handling, and a complete description of all the widgets and all the events found in *Gdk.Event*.

It is worth noting that this book has been published under the Open Publication License. You can get an electronic copy of it at <http://www.opencontent.org/>.

TRANSITIONING FROM GTKADA 2 TO GTKADA 3

18.1 General

GtkAda 3.x is a binding to the C library gtk+ 3.x. This is a major release, with several incompatible changes. Most of those incompatibilities are due to major changes in the C library. Mostly, the gtk+ developers have performed a general cleanup, removing old types and subprograms that were rarely used and belong to more specialized libraries.

They have also made significant changes in the internals of the library. A lot of these changes should not impact typical user code, although they will if you are writing your own container widgets.

The gtk+ developers have documented various things that will likely need to be changed in user applications. The page at <https://docs.gtk.org/gtk3/migrating-2to3.html> provides a migration guide. Its code samples are in C, but should be applicable to Ada quite easily.

GtkAda itself has also undergone its own changes. One of the most significant is that most of the binding is now automatically generated from XML files provided by the gtk+ developers. This ensures that the binding is much more complete than it was before, and will be much easier to evolve when new releases of gtk+ are made available.

It also means that users can, theoretically at least, automatically bind a number of libraries from the gtk+/GNOME ecosystem. The automatic generation relies on XML files, called GIR files from their `.gir` extension. If you wish to parse other files, you should likely modify the toplevel Makefile (the `generate` target), as well as the file `contrib/data.py` to list which types should be bound. We do not necessarily encourage you to generate your own bindings, and this generation is likely to be more than just modifying one or two files...

18.1.1 Interfaces

One other advantage of the automatic generation is that it allows us to provide more advanced features in the binding.

For instance, gtk+ has the notion of interfaces (which play a similar role to Ada05 interfaces).

In GtkAda interfaces no longer require an explicit “with” of the interface package, and a cast to the interface type (with “-” and “+”). Instead, each package now contains the list of subprograms inherited from the various interfaces.

So basically, all subprograms inherited from an interface become available as primitive operations in the types that implement the interface.

We also expect to simplify the handling of signals and signal handlers.

18.1.2 Ada 2012

GtkAda 3 makes use of Ada 2012 and requires GtkAda applications to be compiled in Ada 2012 mode (e.g. using the `-gnat2012` switch).

This makes it possible to use the object-dotted notation when calling primitive operations. For instance, the following code:

```
Gtk.Window.Set_Default_Size (Window, 800, 600);
```

can be replaced with:

```
Window.Set_Default_Size (800, 600);
```

18.2 Pango

18.2.1 Pango.Font

The type `Pango_Font_Metrics` is now declared in its own package `Pango.Font_Metrics`.

The type `Pango_Font_Face` is now declared in its own package `Pango.Font_Face`.

The type `Pango_Font_Family` is now declared in its own package `Pango.Font_Family`.

The type `Pango_Language` is now declared in its own package `Pango.Language`.

18.3 Glib

18.3.1 Glib.Object

`Initialize_Class_Record`'s profile was changed to follow more closely what is done for C applications. The previous implementation prevented applications from implementing interfaces because some internal glib+ data had to be initialized too early. See `glib-object.ads` for an extensive documentation.

18.3.2 Glib.G_Icon

This type is now a `GType_Interface`. Instead of using `Null_G_Icon`, use `Glib.Types.Null_Interface`.

18.4 Gdk

18.4.1 Gdk.Bitmap

This package has been removed: Cairo packages should be used for drawing, and `Gdk.Pixbuf` for representing image data in memory.

18.4.2 Gdk.Color

Alloc no longer exists, and is not necessary since all drawing is now done internally using Cairo which directly manipulates red/green/blue.

18.4.3 Gdk.Cursor

The `Gdk_New` function working on `Gdk_Pixmap` has been removed. Use `Gdk.Pixbuf.Gdk_New_From_Pixbuf` to create a cursor from a `pixbuf`.

The `Gdk_New` function working on a `String` has also been removed.

A `Gdk_Cursor` is now derived from a `Glib.Object`. This has little impact on programs, except that `Null_Cursor` can be replaced simply by “null”.

`Destroy` was removed, and should be replaced with `Unref`.

18.4.4 Gdk.Dnd

The functions for handling `Drag_Contexts` have been moved to new package `Gdk.Drag_Contexts`.

The `Gdk_Drag_Context` itself now inherits from `GObject`, which means that it no longer requires its own `Ref/Unref` functions.

`Drag_Find_Window` has been removed, use `Drag_Find_Window_For_Screen` instead.

`Drag_Get_Protocol` has been replaced with `Drag_Context_Get_Protocol`.

18.4.5 Gdk.Drawable

All `Draw_*` subprograms have been removed: use Cairo for low-level drawing.

The type `Gdk_Drawable` no longer exists.

18.4.6 Gdk.Event

A lot of the getters (and all of the setters) were removed. Instead, the `Gdk_Event` type fields can now be edited directly. This is slightly more efficient, and more importantly better documents which fields are valid for which event types.

The APIs to `Get_Message_Type`, `Set_Message_Type`, `Get_Data`, and `Set_Data` have been removed without replacement.

`Get_Graphics_Expose` and `Send_Client_Message` have been removed with no replacement.

`Deep_Copy` was removed. It is now possible to simply use “:-” on the record type itself.

`Get` and `Peek` are now functions instead of procedures with a single out parameter.

`Is_Created` has been removed (you can compare with null) `Send_Client_Message_To_All` has been removed (deprecated in gtk+)

`Allocate` has been removed. Instead, users should directly use `Gdk.Event.Gdk_New` and set the appropriate fields.

`Get_X` and `Get_Y` were replaced by `Get_Coords`. `Get_X_Root` and `Get_Y_Root` were replaced by `Get_Root_Coords`

`Get_Button`, `Get_State`, `Get_Key_Val` and `Get_Keycode` were kept (so you do not have to directly access the field of `Gdk_Event`). However, they no longer raise an exception if you pass them an invalid event type, but return an out-of-range value.

18.4.7 Gdk.Font

This package has been removed: use `Pango.Font` for fonts manipulation, `Cairo.Font_Face` and `Cairo.Font_Options` for text rendering.

18.4.8 Gdk.GC

This package has been removed: Cairo packages should be used for drawing.

18.4.9 Gdk.Image

This package has been removed: use a `Gdk.Pixbuf` instead.

18.4.10 Gdk.Main

`Set_Locale` functions are no longer needed and have been removed.

Functions `Set_Use_Xshm` and `Get_Use_Xshm` have been removed.

18.4.11 Gdk.Pixbuf

`Render_Threshold_Alpha`, `Render_To_Drawable`, `Render_To_Drawable_Alpha`, `Render_Pixmap_And_Mask`, `Render_Pixmap_And_Mask_For_Colormap` have been removed.

Use APIs provided by `Gdk.Cairo` to draw a `pixbuf` on a `Gdk_Drawable`.

`Get_From_Drawable` has been removed, use `Get_From_Surface` or `Get_From_Window`.

18.4.12 Gdk.Pixmap

This package has been removed: Cairo packages should be used for drawing, and `Gdk.Pixbuf` for representing image data in memory.

18.4.13 Gdk.Region

This package has been removed and replaced with `Cairo_Region`.

18.4.14 Gdk.RGB

This package is deprecated in gtk3. Use `Pixmaps/Cairo` for drawing, and use `Gdk.Pixbuf` for offscreen image manipulation and rendering to drawables.

Instead of `Gdk.Rgb.Get_Cmap`, use `Gtk.Widget.Get_Default_Colormap`.

18.4.15 Gdk.Window

A `Gdk_Window` now derives from `GObject`. This is mostly transparent for applications, unless you are passing a `Gdk_Window` directly to C code, in which case you must use `Get_Object()` on it.

`Copy_Area` and `Set_Back_Pixmap` have been removed: use `Gdk_Drawable` and `Gdk.Cairo` functions instead.

`Clear_Area` and `Clear_Area_E` were removed. Use `Cairo` for all drawings.

`Get_Desk_Relative_Origin`: this function has been removed without a replacement.

`Get_Toplevels` has been removed, use `Gtk.Window.List_Toplevels` instead.

`Set_Hints` has been removed. Depending on what you are trying to do, use `Gtk.Window.Resize`, `Gtk.Window.Set_Size_Request`, `Gtk.Window.Move`, `Gtk.Window.Parse_Geometry`, and `Gtk.Window.Set_Geometry_Hints`.

`Window_At_Pointer` was renamed to `At_Pointer`.

`Get_Origin` is now a procedure, because the return value had no meaning anyway.

`Get_Geometry`: no longer returns the color depth of the window, which is no longer relevant to gtk+.

The first parameter of the various methods was renamed “`Self`” instead of “`window`” to avoid a number of cases where we would end up with duplicate parameter names.

18.4.16 Gdk.Window_Attr

Parameter “`Colormap`” has been removed from procedure `Gdk_New`. This parameter is no longer needed.

`Set_Colormap` and `Get_Colormap` should no longer be needed and have been removed as well.

18.5 Gtk

18.5.1 Gtk.Action

`Block_Activate_From`, `Unblock_Activate_From`, `Connect_Proxy`, `Disconnect_Proxy`: these obsolete sub-programs have been removed without a replacement.

`Get_Action` has been removed without a replacement.

`Convert` has been removed, use `Glib.Object.GetUser_Data` instead.

18.5.2 Gtk.Aspect_Frame

Direct accessors `Get_Xalign`, `Get_Yalign` and `Get_Ratio` have been removed: use the corresponding properties instead.

18.5.3 Gtk.Assistant

The values in `Gtk_Assistant_Page_Type` were renamed for consistency, removing their `Gtk_` prefix.

The package `Generic_Assistant_Functions` has been renamed to `Set_Forward_Page_Func_User_Data`.

18.5.4 Gtk.Builder

`Add_From_File` now returns a `Guint` and the error as a parameter.

`Get_Widget` has been removed (use `Get_Object` instead, and cast to the appropriate type)

18.5.5 Gtk.Button_Box

`Set_Child_Size` was removed. Equivalent behavior can only be done by changing the theme properties `child-min-width` and `child-min-height`.

18.5.6 Gtk.Cell_Layout

`Get_Cell_Renderers` has been renamed to `Get_Cells`.

18.5.7 Gtk.Cell_Renderer

The `Render` subprogram is now called with a `Cairo_Context` rather than a `Gdk_Window`.

18.5.8 Gtk.Cell_View

`Get_Cell_Renderers` is obsolete, use the `Gtk.Cell_Layout` interface and `Gtk.Cell_Layout.Get_Cells`.

18.5.9 Gtk.Clist

This widget has been removed: use a `Gtk.Tree_View` instead.

18.5.10 Gtk.Container

Procedure `Propagate_Expose` has been removed and will be replaced with `Propagate_Draw`.

`Class_Find_Child_Property`, `Class_list_Child_Properties` and `Class_Install_Child_Property` are no longer bound.

`Children` was removed (use `Get_Children` instead).

18.5.11 Gtk.Color_Button

The function `Get_Color` returning `Gdk.Color.Gdk_Color` is now a procedure with an out parameter.

18.5.12 Gtk.Color_Selection

`Get_Color` and `Set_Color` have been removed: use `Get_Current_Color` and `Set_Current_Color` instead.

18.5.13 Gtk.Color_Selection_Dialog

Subprogram `Get_Colorsel` has been renamed `Get_Color_Selection`, to match the Gtk+ naming.

`Get_OK_Button`, `Get_Cancel_Button`, `Get_Help_Button` have been removed. Instead, use:

```
Gtk_Button (Glib.Properties.Get_Property (Dialog, Ok_Button_Property)),
Gtk_Button (Glib.Properties.Get_Property (Dialog, Cancel_Button_Property)),
Gtk_Button (Glib.Properties.Get_Property (Dialog, Help_Button_Property))
```

18.5.14 Gtk.Combo

This widget has been removed: use a `Gtk.Combo_Box` instead.

18.5.15 Gtk.Combo_Box

The “text only” variant has been moved to the new package `Gtk.Combo_Box_Text`.

18.5.16 Gtk.Combo_Box_Entry

This widget has been removed: use a `Gtk.Combo_Box` instead.

18.5.17 Gtk.Clipboard

The base type is now a `GObject_Record` instead of an opaque type: use the `GObject` facilities for lifecycle management.

There are now separate “User_Data” generic version for callback-based methods.

18.5.18 Gtk.Ctree

This widget has been removed: use a `Gtk.Tree_View` instead.

18.5.19 Gtk.Curve

This widget has been removed, with no direct replacement. Use drawing functionality from Cairo instead.

18.5.20 Gtk.Dialog

Subprogram `Get_Vbox` was replaced with `Get_Content_Area`.

Subprogram `Set_Has_Separator` has been removed: use the corresponding flag in the call to `Gtk_New/Initialize` instead.

18.5.21 Gtk.Dnd

`Source_Set_Icon` has been removed: use `Source_Set_Icon_Pixbuf` instead. `Set_Icon_Pixmap` has been removed: use `Set_Icon_Pixbuf` instead.

Obsolete `Set_Default_Icon` working on `Gdk.Pixmap` has been removed without a replacement.

18.5.22 Gtk.Editable

The type representing a `Gtk_Editable_Record` has been changed from a `Widget` (which is a `GObject`) to an interface (a `System.Address`). Therefore the `Gtk_Editable_Record` type has been eliminated. User code referencing only the `Gtk_Editable` type should function unchanged.

Code using the tag as a test before converting a widget to a `Gtk.Editable` can now work using the `Implements_Editable` package.

For instance, if `Widget` is a `GObject_Record`, the following code

```
if Widget.all in Gtk_Editable_Record'Class then
  Cut_Clipboard (Gtk_Editable (Widget));
```

becomes

```
if Is_A (Widget.Get_Type, Gtk.Editable.Get_Type) then
  Cut_Clipboard (+Widget);
```

where the function “+” is defined by instantiating `Implements_Editable`:

```
package Implements_Editable is new Glib.Types.Implements
  (Gtk.Editable.Gtk_Editable, GObject_Record, GObject);
function "+"
  (Widget : access GObject_Record'Class)
  return Gtk.Editable.Gtk_Editable
  renames Implements_Editable.To_Interface;
```

The `Select_Region` subprogram parameter name `The_End` has been normalized to `End_Pos`.

18.5.23 Gtk.Entry_Completion

The “match-selected” and “cursor-on-match” signals were erroneously given the internal filter model instead of the users model. This oversight has been fixed in GTK+ 3; if you have handlers for these signals, they will likely need slight adjustments.

18.5.24 Gtk.Enums

The following types were removed

- GtkAnchorType
- GtkCurveType
- GtkMetricType
- GtkGridLines
- GtkUpdateType
- GtkVisibility
- GtkSideType
- GtkMatchType
- GtkPreviewType
- GtkSubmenuDirection
- GtkSubmenuPlacement
- GtkTreeViewMode

Gtk_Icon_Size is no longer an enumeration type, but an integer, so that new sizes can be defined through Gtk.Icon_Factory.Icon_Size_Register.

18.5.25 Gtk.File_Chooser_Button

Subprograms Gtk_New_With_Backend and Initialize_With_Backend have been removed: use Gtk_New and Initialize instead.

18.5.26 Gtk.File_Chooser_Dialog

Subprograms Gtk_New_With_Backend and Initialize_With_Backend have been removed: use Gtk_New and Initialize instead.

18.5.27 Gtk.File_Chooser_Widget

Subprograms Gtk_New_With_Backend and Initialize_With_Backend have been removed: use Gtk_New and Initialize instead.

18.5.28 Gtk.File_Selection

This package has been replaced by `Gtk.File_Chooser`. You may also use `Gtkada.File_Selection` for a simple interface to the `Gtk.File_Chooser`.

18.5.29 Gtk.Fixed

Subprograms `Set_Has_Windows` and `Get_Has_Windows` are now in `Gtk.Widget`.

18.5.30 Gtk.Gamma_Curve

This widget has been removed without any replacement.

18.5.31 Gtk.GC

This package has been removed: Cairo packages should be used for drawing.

18.5.32 Gtk.GEntry

The names for `Gtk_Entry_Record` parameters have been normalized across the board to “`The_Entry`”.

`Append_Text` has been removed: use `Set_Text` and `Get_Text` instead.

18.5.33 Gtk.GRange

`Set_Update_Policy` has been removed, with no replacement. If you require delayed updates, you will need to code it yourself.

18.5.34 Gtk.Handle_Box

This package is now marked as deprecated in C, and is likely to be removed in future versions of gtk+, so we encourage you to stop using it as well.

18.5.35 Gtk.HRuler

This widget has been removed without any replacement.

18.5.36 Gtk.Icon_Factory

`Gtk_Icon_Set` and `Gtk_Icon_Source` have been moved to their own packages. Functions `Gtk_New` are now procedures.

18.5.37 Gtk.Image

The subprograms working with `Gdk_Pixmap` have been removed, use the variants working on `Gdk_Pixbuf` instead.

18.5.38 Gtk.Image_Menu_Item

All controlling parameters were renamed to `Self`. There was no consistency before.

`Gtk_New_From_Stock` now requires an `Accel_Group` parameter, which can be set to null.

18.5.39 Gtk.Input_Dialog

This package is no longer part of gtk+, so this binding has been removed without replacement.

18.5.40 Gtk.Item

This obsolete package has been removed with no replacement.

18.5.41 Gtk.Item_Factory

This obsolete package has been removed in favor of `Gtk.UI_Manager`.

18.5.42 Gtk.Layout

`Get_Width` and `Get_Height` have been removed, use `Get_Size` instead.

18.5.43 Gtk.Link_Button

All widget parameter names have been normalized to “`Self`”.

The `Set Uri_Hook` function has been eliminated, and along with it the `Uri_Func` type and the `Generic Uri_Hook` package. Register a callback for the button’s “clicked” signal instead.

18.5.44 Gtk.List_Item

This widget has been removed: use a `Gtk.Tree_View` instead.

18.5.45 Gtk.Main

`Do_Event` was renamed `Main_Do_Event`.

Grab_Add and Grab_Removed are available in `Gtk.Widget` (as was already the case with gtk2).

The `Quit` package has been removed without replacement.

The `Idle` and `Timeout` handling been removed: use equivalent functions in package `Glib.Main` instead.

18.5.46 Gtk.Menu

User_Menu_Popup has been replaced by Popup_User_Data.

The version of Popup was took an access to C_Gtk_Menu_Positon_Func has been removed. If you need to pass User_Data to the callback, you need to instantiate the package Popup_User_Data. Note that in this package the position of the Data parameter has changed.

18.5.47 Gtk.Menu_Item

For subprogram Set_Right_Justified, the parameter "Justify" has been renamed to "Right_Justified".

The obsolete procedures Remove_Submenu, Set_Right_Justify, and Right_Justify have been removed. Instead, use Set_Submenu, Set_Right_Justified, or Set_Right_Justified with Justify=True, respectively.

Calling Gtk_New with one Menu_Item argument has the same effect now as before. However, from this version on, if a Label argument exists (even if set to ""), a Gtk_Label child will be created with the given value.

18.5.48 Gtk.Menu_Tool_Button

Set_Arrow_Tooltip has been removed, use Set_Arrow_Tooltip_Markup or Set_Arrow_Tooltip_Text instead.

18.5.49 Gtk.Notebook

Get_Children has been removed: call Gtk.Container.Get_Children instead.

Set_Tab_Label_Packing has been removed (this is left under control of the theme).

Set_Page has been removed, use Set_Current_Page instead.

Insert_Page now returns the number of the page that has been inserted.

18.5.50 Gtk.List

This package has been removed: use a Gtk_Tree_View instead.

18.5.51 Gtk.Object

Gtk.Object has been removed in gtk+-3.

The following subprograms and declarations are now in Gtk.Widget:

- Flags
- Unset_Flags
- Floating
- In_Destruction_Is_Set
- Signal_Destroy

The subprogram Gtk.Object.Sink has been removed: use Glib.Object.Ref_Sink instead.

18.5.52 Gtk.Old_Editable

This obsolescent API has been removed, use `Gtk.Editable` where relevant.

18.5.53 Gtk.Option_Menu

`Gtk.Option_Menu` has been removed. Using `Gtk.Combo_Box` instead is recommended.

18.5.54 Gtk.Pixmap

This widget has been removed and is generally replaced with a `Gtk.Image`.

18.5.55 Gtk.Preview

This widget has been removed without replacement.

18.5.56 Gtk.Print_Operation

`Get_Status` was renamed to `Get_Status_String` when it returns a string, to match the gtk+ API.

18.5.57 Gtk.Progress

This widget has been removed without any replacement.

18.5.58 Gtk.Progress_Bar

This widget is now derived from `Gtk.Widget` directly, rather than from `Gtk.Progress` (which has been removed).

The enumeration type `Gtk_Progress_Bar_Orientation` has been removed, and this widget now implements the `Gtk_Orientable` interface. To fully achieve the same functionality as the GtkAda 2.x `Get_Orientation/Set_Orientation` subprograms, it is now necessary to call `Get_Orientation/Set_Orientation` along with `Get_Inverted/Set_Inverted`.

Procedure `Set_Pulse_Step`'s "Step" parameter has been renamed to "Formal."

`Set_Ellipsize` and `Get_Ellipsize` parameter names have been normalized from "Pbar" to "Progress_Bar".

If you intend to show text over the progress bar, you need to call `Set_Text` as before, but also call `Set_Show_Text(True)`.

18.5.59 Gtk.Rc

This package is now mostly obsolete. The gtk+ library no longer supports the *.rc files, since it uses CSS-like files instead.

18.5.60 Gtk.Recent_Manager

The type `Gtk_Recent_Info` is now bound in its own package.

18.5.61 Gtk.Ruler

This widget has been removed without any replacement.

18.5.62 Gtk.Settings

Properties are now named with the suffix “_Property”. For instance, `Gtk_Theme_Name` is now `Gtk_Theme_Name_Property`.

18.5.63 Gtk.Scale_Button

This package now conforms to the API conventions practiced throughout the rest of the toolkit. `Gtk_New` is implemented as a procedure rather than as a function, and the use of `GNAT.Strings.String_List` replaces `Gtkada.Types.Chars_Ptr_Array` throughout.

18.5.64 Gtk.Selection

This package has been renamed `Gtk.Selection_Data`, for homogeneity with the naming conventions.

`Gtk.Selection.Selection_Data` is now called `Gtk.Selection_Data.Gtk_Selection_Data`.

Handling of `Target_Lists` has been moved to the new package `Gtk.Target_List`, along with `Target_Entry_Array`.

The type `Gtk_Target_Entry` has been moved to the new package `Gtk.Target_Entry`.

The way of obtaining the selection data from callbacks using the `Args/GValues` approach has changed, from:

```
Data : constant Gtk.Selection.Selection_Data :-
  Gtk.Selection.Selection_Data (Get_Proxy (Nth (Args, 2)));
```

to:

```
Data : constant Gtk.Selection_Data.Gtk_Selection_Data :-
  From_Object (Get_Address (Nth (Args, 2)));
```

The type `Target_Flags` has been moved to `Gtk.Enums.Gtk_Target_Flags`.

The flag corresponding to `Target_No_Constraint` has been removed: use the value 0 instead.

18.5.65 Gtk.Scrolled_Window

Set_Policy's parameters were renamed to Hscrollbar_Policy and Vscrollbar_Policy instead of H_Scrollbar_Policy and V_Scrollbar_Policy.

18.5.66 Gtk.Socket / Gtk.Plug

The binding for these two packages was removed. They are not portable across platforms, and require access to the low-level X11 window ID, for which we do not provide a binding.

18.5.67 Gtk.Status_Icon

Status_Icon widget parameter names have been normalized to "Status_Icon".

Get_Blinking and Set_Blinking have been removed, it is no longer possible to make the status icon blink.

18.5.68 Gtk.Style

All functions based on Gdk.GC or Gdk.Pixmap have been removed. This package is deprecated (but not removed yet) in gtk3 Use functions in Gtk.Style_Context instead.

A number of drawing functions have been removed: use the Paint_* functions instead.

Replace a call to Get_Font with:

```
with Gtk.Style_Context; use Gtk.Style_Context;  
Get_Style_Context (Widget).Get_Font (Gtk_State_Flags_Normal);
```

18.5.69 Gtk.Text

This obsolescent API has been removed: use a Gtk.Text_View/Gtk.Text_Buffer instead.

18.5.70 Gtk.Text_Attributes

Set_Fg_Stipple, Get_Fg_Stipple, Set_Bg_Stipple, Get_Bg_Stipple have been removed without a replacement.

18.5.71 Gtk.Text_View

The functions Get/Set_Disable_Scroll_On_Focus have no effect in recent versions of gtk+ and have been removed.

18.5.72 Gtk.Tree_Dnd

This package was removed, and its contents split into `Gtk.Tree_Drag_Source` and `Gtk.Tree_Drag_Source`.

The ``Drag_Dest_`` and ``Drag_Source_`` prefixes were removed from the subprogram, so for instance `Drag_Dest_Drag_Data_Received` has become `Drag_Data_Received`.

18.5.73 Gtk.Tree_Model

A `Gtk_Tree_Model` is now an interface (implemented by `Gtk_List_Store` and `Gtk_Tree_Store`), no longer a tagged type. It means that in callbacks that receive a `Gtk_Tree_Model` parameter, you can no longer cast this parameter to a `Gtk_Tree_Store` for instance. Instead, you need to do the following:

```
-- Model is the parameter, of type Gtk_Tree_Model
Tree : constant Gtk_Tree_Store :- Gtk_Tree_Store (-Model);
```

`Gtk_New`, for a `Gtk_Tree_Path`, are now procedures instead of functions, to follow the usual GtkAda convention.

`Gtk_Tree_Row_Reference` has been moved to its own package `Gtk.Tree_Row_Reference`.

`Gtk_New` and `Gtk_New_First` (for a tree path) now take a “out” parameter, for consistency with the rest of the API.

18.5.74 Gtk.Tree_View_Column

`Get_Cell_Renderers` is obsolete, use the `Gtk.Cell_Layout` interface and `Gtk.Cell_Layout.Get_Cells`.

18.5.75 Gtk.Tips_Query

This obsolete package has been removed.

18.5.76 Gtk.Tool_Item

`Set_Tooltip` has been removed: use `Set_Tooltip_Text` and `Set_Tooltip_Markup` instead.

18.5.77 Gtk.Toolbar

All `Gtk_Toolbar` widget parameter names have been normalized to “Toolbar”.

Subprograms `Append_*`, `Prepend_*` and `Insert_*` have been removed: use `Insert` instead.

Subprograms `Get_Tooltips/Set_Tooltips` have been removed. Use the `Gtk_Enable_Tooltips` property instead.

18.5.78 Gtk.Tooltips

The package `Gtk.Tooltips` has been removed, in favor of `Gtk.Tooltip`.

For creating simple tooltips on all GtkAda widgets, the easiest is to use `Gtk.Widget.Set_Tooltip_Text` or `Gtk.Set_Tooltip_Markup`. See the example in `gtkada_demo/create_tooltip.adb`.

18.5.79 Gtk.Tree_View

Procedure `Create_Row_Drag_Icon` now returns a `Cairo_Surface`.

`Get_Hadjustment`, `Set_Hadjustment`, `Get_Vadjustment`, `Set_Vadjustment` have been removed: use the equivalent properties.

`Widget_To_Tree_Coords` and `Tree_To_Widget_Coords` have been removed: use `Convert_Widget_To_Tree_Coords` and `Convert_Tree_To_Widget_Coords`.

18.5.80 Gtk.VRuler

This widget has been removed without any replacement.

18.5.81 Gtk.Widget

The old `Draw` function no longer exists, and should be replaced with calls to `Queue_Draw_Area`. However, a new `Draw` function was added with a different profile and different semantic.

Function ```Get_Snapshot``` has been removed. `Draw` should be used instead.

`Hide_All` has been removed: use `Hide` instead.

`Set_Extension_Events` and `End_Extension_Events` are no longer needed and have been removed.

`Set_Colormap` and `Get_Colormap` are no longer needed and have been removed.

`Set_Scroll_Adjustments` has been removed without a replacement.

`Shape_Combine_Mask`, `Input_Shape_Combine_Mask` and `Reset_Shapes` have been removed without replacements.

`Set_Uposition` has been removed: use the properties of the containing widget to fix the position of contained widgets. The functions in `Gtk.Window`, for instance `Gtk.Window.Move`, should be used for top-level widgets.

`Set_USize` has been removed: use `Set_Size_Request` instead.

`Size_Request` is now obsolescent. The recommend replacement is to use `Get_Preferred_Width` and `Get_Preferred_Height`.

`Set_Default_Colormap`, `Get_Default_Colormap`, `Push_Colormap` and `Pop_Colormap` were removed. They are no longer needed, since all drawing is done through Cairo which doesn't use a colormap but directly the red/green/blue components.

`Queue_Clear` and `Queue_Clear_Area` have been removed, call `Queue_Draw` and `Queue_Draw_Area` instead.

The signal "expose_event" no longer exists. It has been replaced with the "draw" signal which provides a preconfigured `Cairo_Context` suitable for the drawing (including the clip area that is used to speed up the rendering).

`Activate` is now a function.

`Child_Focus`: removed default value for `Direction` parameter (was `Dir_Tab_Forward`)

`Get_Allocation_Height` and `Get_Allocation_Width` are now named `Get_Allocated_Height` and `Get_Allocated_Width`.

`Get_Allocation_X` and `Get_Allocation_Y` were removed, and can be accessed through `Get_Allocation.X` and `Get_Allocation.Y` instead.

A lot of flags (`Can_Focus`, `Can_Default`, ...) now have explicit setters and getters. This removed a number of subprograms, like:

- `Double_Buffered_Is_Set` (see `Get_Double_Buffered`)
- `Can_Focus_Is_Set` (see `Get_Can_Focus`)
- `Mapped_Is_Set` (see `Get_Mapped`)
- `Realized_Is_Set` (see `Get_Realized`)
- `Has_Default_Is_Set` (see `Has_Default`)
- `Has_Focus_Is_Set` (see `Has_Focus`)
- `Has_Grab_Is_Set` (see `Has_Grab`)
- `Rc_Style_Is_Set` (see `Has_Rc_Style`)
- `In_Destruction_Is_Set` (see `In_Destruction`)
- `Drawable_Is_Set` (see `Is_Drawable`)
- `No_Window_Is_Set` (see `Has_Window`)

`Size_Allocate` now takes an “in out” parameter for the allocation

`Set_Flags` was renamed `Set_State_Flags` `Unset_Flags` was renamed `Unset_State_Flags` `Flags` and `Flag_Is_Set` must be replaced with a call to `Get_State_Flags`

`Get_Child_Requisition` is now a procedure with an in out parameter. It is obsolescent.

`Default_Motion_Notify_Event` was removed.

`Has_Default_Motion_Notify_Handler` was removed.

`Get_Default_Visual` was removed.

`Restore_Default_Style` was removed (use `Set_Style` with a null parameter instead).

`Class_Find_Style_Property`, `Class_List_Style_Properties` and `Class_Install_Style_Property` were removed. They are mostly of interest when writing theme engines.

`Class_Path` and `Path` were replaced with `Get_Path`.

`Allow_Shrink_Property` and `Allow_Grow_Property` have been removed: use `Get_Expand` and `Get_Vexpand` instead.

`Render_Icon` has been replaced by `Render_Icon_Pixbuf`.

18.5.82 Gtk.Window

`Set_Has_Frame`, `Get_Has_Frame`, `Set_Frame_Dimensions`, `Get_Frame_Dimensions`: these special-purpose subprograms have been removed without replacement.

`Get_Gravity`, `Set_Gravity`: these have been removed, use the property `Gravity_Property` instead.

`Resize` no longer accepts parameters set to -1 to indicate the preferred size of the window. This was a GtkAda extension, which can be achieved using `Get_Preferred_Size` and passing the result to `Size`.

`Group_Add_Window` was renamed to `Add_Window`. `Group_Remove_Window` was renamed to `Remove_Window`. `Group_List_Windows` was renamed to `List_Windows`.

`Initialize` now has the same default value for its `The_Type` parameter as `Gtk_New`.

18.6 GtkAda

18.6.1 Gtkada.MDI

`Set_Dnd_Message` no longer has a special handling for “#”, which was
used to indicate whether the window would be preserved or hidden when changing perspectives. Instead, a different color is used to highlight the target area (and this highlighting is now done using transparency).

18.6.2 Gtkada.Properties

This package has been removed. It used to provide a dialog allowing you to view and edit the properties of widgets in your application, live. This is now provided by the GTK Inspector, which is built into GTK itself.

18.7 Gnome

18.7.1 Gnome.App_Bar

Subprogram `Appbar_Get_Progress` has been removed without replacement.

18.7.2 Gnome.Gentry

This package has been removed without replacement.

GNU FREE DOCUMENTATION LICENSE

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

19.1 PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document ‘free’ in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ‘copyleft’, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

19.2 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The ‘Document’, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ‘you’.

A ‘Modified Version’ of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ‘Secondary Section’ is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ‘Invariant Sections’ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The ‘Cover Texts’ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A ‘Transparent’ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not ‘Transparent’ is called ‘Opaque’.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The ‘Title Page’ means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ‘Title Page’ means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

19.3 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

19.4 COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

19.5 MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled 'History', and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled 'History' in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the 'History' section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- In any section entitled 'Acknowledgements' or 'Dedications', preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled 'Endorsements'. Such a section may not be included in the Modified Version.
- Do not retitling any existing section as 'Endorsements' or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled 'Endorsements', provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

19.6 COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled ‘History’ in the various original documents, forming one section entitled ‘History’; likewise combine any sections entitled ‘Acknowledgements’, and any sections entitled ‘Dedications’. You must delete all sections entitled ‘Endorsements.’

19.7 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

19.8 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an ‘aggregate’, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

19.9 TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

19.10 TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

19.11 FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ‘or any later version’ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

19.12 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission **is** granted to copy, distribute **and/or** modify this document under the terms of the GNU Free Documentation License, Version **1.1** **or any** later version published by the Free Software Foundation; **with** the Invariant Sections being LIST THEIR TITLES, **with** the Front-Cover Texts being LIST, **and with** the Back-Cover Texts being LIST. A copy of the license **is** included **in** the section entitled ‘GNU Free Documentation License’.

If you have no Invariant Sections, write ‘with no Invariant Sections’ instead of saying which ones are invariant. If you have no Front-Cover Texts, write ‘no Front-Cover Texts’ instead of ‘Front-Cover Texts being LIST’; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Copyright C 1998-2000, Emmanuel Briot, Joel Brobecker, Arnaud Charlet

Copyright C 2000-2026, AdaCore

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘GNU Free Documentation License’.