GtkAda Reference Manual

Version 2.14.2
Document revision level $Revision: 134935 $
Date: $Date: 2008-12-17 10:27:57 +0100 (Wed, 17 Dec 2008) $

E. Briot, J. Brobecker, A. Charlet

1 Package Glib

This package provides definitions for the basic types used in Glib, Gdk and Gtk.

1.1 Types

subtype Allocation_Int **is** Gint;

Provided for better compatibility between GtkAda 1.2 and 2.0

type Boolean_Array **is array** (Natural **range** <>) **of** Boolean;

type Boxed_Copy **is access function**
(Boxed : System.Address) **return** System.Address;

type Boxed_Free **is access procedure**
(Boxed : System.Address);

type C_Proxy **is access all** C_Dummy;

General proxy for C structures. This type is used instead of System.Address so that the variables are automatically initialized to 'null'. The value pointed to is irrelevant, and in fact should not be accessed. It has thus been made limited private with no subprogram to access it. C_Proxy is a public type so that one can compare directly the value of the variables with 'null'.

type GQuark **is new** Guint32;

Represents a string internally in GtkAda. Once you know the equivalent for a string, you can always use it instead of the string, which provides a faster access for all the functions that use htables in GtkAda. There is a global htable that contains all the quarks defined in your application and GtkAda itself.

type GTime_Val **is record**
TV_Sec : Glong;
TV_Usec : Glong;
end record;

type GTime_Val_Access **is access all** GTime_Val;

type GType **is new** Gsize;

This type describes an internal type in Glib. You shouldn't have to use it in your own applications, however it might be useful sometimes. Every object type is associated with a specific value, created dynamically at run time the first time you instantiate an object of that type (thus if you have never used e.g a Gtk_File_Selection, it won't have any GType associated with it). You can get the exact type value for each type by using the functions Get_Type provided in all the packages in GtkAda. You can get the specific value for an existing widget by using the function Glib.Object.Get_Type.

type GType_Array **is array** (Guint **range** <>) **of** Glib.GType;

type GType_Class **is private**;

An opaque structure used as the base for all classes in glib and gtk+. See also Glib.Object.GObject_Class for a more useful child of this type.

```
type G_Destroy_Notify is access procedure
          (Data : Glib.C_Proxy);
```

```
type G_Destroy_Notify_Address is access procedure
          (Data : System.Address);
```

type Gboolean **is new** Gint;

type Gboolean_Array **is array** (Natural **range** <>) **of** Gboolean;

type Gchar **is new** C.char;

subtype Gcolor_Int **is** Guint16;

Provided for better compatibility between GtkAda 1.2 and 2.0

type Gdouble **is new** C.double;

type Gdouble_Array **is array** (Natural **range** <>) **of** Gdouble;

```
type Gfloat is new C.C_float;

type Gfloat_Array is array (Natural range <>) of Gfloat;

type Gint is new C.int;

type Gint16 is range -(2 ** 15) .. (2 ** 15 - 1);

type Gint32 is range -(2 ** 31) .. (2 ** 31 - 1);

type Gint64 is range -(2 ** 63) .. (2 ** 63 - 1);

type Gint8 is range -(2 ** 7) .. (2 ** 7 - 1);

type Gint_Array is array (Natural range <>) of Gint;

type Glong is new C.long;

type Glong_Array is array (Natural range <>) of Glong;

subtype Grange_Float is Gdouble;
```

Needed for better compatibility between GtkAda 1.2 and 2.0

```
type Gshort is new C.short;

type Gshort_Array is array (Natural range <>) of Gshort;

type Gsize is new C.size_t;
```

```
type Guchar is new C.unsigned_char;

type Guchar_Array is array (Natural range <>) of Guchar;

type Guchar_Array_Access is access Guchar_Array;

type Guint is new C.unsigned;

type Guint16 is mod 2 ** 16;

type Guint32 is mod 2 ** 32;

type Guint32_Array is array (Natural range <>) of Guint32;

type Guint64 is mod 2 ** 64;

type Guint8 is mod 2 ** 8;

type Guint_Array is array (Natural range <>) of Guint;

type Gulong is new C.unsigned_long;

type Gulong_Array is array (Natural range <>) of Gulong;

type Gunichar is new Guint32;

type Gushort is new C.unsigned_short;
```

type Gushort_Array **is array** (Natural **range** <>) **of** Gushort;

type Long_Array **is array** (Natural **range** <>) **of** C.long;

type Param_Flags **is mod** 2 ** 6;

type Param_Spec **is new** Glib.C_Proxy;

type Param_Spec_Array **is array** (Natural **range** <>) **of** Param_Spec;
See Glib.Properties.Creation for more information on this type

type Property **is private**;

type Short_Array **is array** (Natural **range** <>) **of** C.short;

type Signal_Id **is private**;
This uniquely identifies a connection widget<->signal.

type Signal_Name **is new** String;
A signal name as used in connect, shared type between the Gtk and Glib layer.

type String_Ptr **is access all** String;

subtype UTF8_String **is** String;
A string that accepts only valid UTF8 sequences. Most Gtk+ function expect valid UTF8 strings instead of regular strings.

1.2 Subprograms

1.2.1 Conversion services

```
function To_Boolean_Array
  (A                  : in    Gboolean_Array)
   return Boolean_Array;
```

Convert a C-style boolean array into an Ada-style array.

```
function To_Gint
  (Bool               : in    Boolean)
   return Gint;
```

Convert an Ada boolean into a C int.

1.2.2 Quarks

```
function Quark_From_String
  (Id                 : in    String)
   return GQuark;
```

Return, or create the quark associated with the string.
Note that if the quark does not already exist, an entry is created for it in the global htable for quarks.

```
function Quark_Try_String
  (Id                 : in    String)
   return GQuark;
```

Return the quark associated with the string, if it exists.
If it does not exist, return Unknown_Quark.

1.2.3 Properties

This is only the definition of the property types. See Glib.Properties@* on how to get and set the value of properties for specific objects, or the package Glib.Properties.Creation for information on how to create new properties in your own widgets. Introspection is available, ie from an existing object you can find out the list of properties it supports. See the functions Glib.Object.Interface_List_Properties and Glib.Object.Class_List_Properties

```
function Build
  (Name               :       String)
   return Property;
```

You should use this function only if you are creating new widgets, and
their properties. Normal usage of properties doesn't require the use of this function. An ASCII.NUL character is automatically appended if necessary

```
function Property_Name
  (Prop               :       Property)
   return String;
```

Return the name of the property.
This name includes the trailing ASCII.Nul, and thus can be passed as is to C.

1.2.4 GType

```
function Parent
  (Typ                :       GType)
   return GType;
```

Return the parent type of Typ (eg if Typ is associated with a Gtk
widget, it returns the typ of its parent).

```
function Fundamental
  (Typ                :       GType)
   return GType;
```

Return the fundamental type for Type. In gtk+, the types are organized
into several hierarchies, similar to what is done for widgets. All of these hierarchies are based

on one of the fundamental types defined below. This function returns that fundamental type.

For instance, each enumeration type in gtk+ has its own GType. However, Fundamental will return GType_Enum in all of these cases.

```
function Type_Name
  (Type_Num           : in    GType)
   return String;
```

Return the name of the type (enumeration,...) associated with Typ.
If Fundamental (Typ) return GType_Enum, this returns the name of the enumeration type that Typ represents. This might be useful in debug messages.

```
function Type_From_Name
  (Name               : in    String)
   return GType;
```

Convert a string to the matching type.
Name should be the C GObject name rather than the Ada name: thus, use names such as GtkScrollbar or GtkButton for widgets.

```
function Get_Qdata
  (Typ                :       GType;
   Quark              :       GQuark)
   return Glib.C_Proxy;
```

Return the user data set for Typ

```
procedure Set_Qdata
  (Typ                :       GType;
   Quark              :       GQuark;
   Data               :       Glib.C_Proxy);
```

Associate some named data with Typ.

1.2.5 Boxed types

Boxed types are a convenient way to encapsulate Ada types through a C@* layer. An initialization and a finalization function can be provided. The most frequent usage of such types is in argument to signals and handlers (See the functions in Glib.Values), or to store such types in a Gtk_Tree_Model. This allows you for instance to store reference counted types where you want to be able to control what should happen when the cell is removed from the tree.

See an example with the subprogram Glib.Values.Set_Boxed

```
function Boxed_Type_Register_Static
  (Name                :       String;
   Copy                :       Boxed_Copy;
   Free                :       Boxed_Free)
   return GType;
```

Create a new boxed type

2 Package Glib.Convert

This package provides definitions for string conversions and i18n. See also Glib.Unicode.

2.1 Subprograms

```
function Convert_Error_Domain  return GQuark;
```

Return the error domain associated with Glib.Convert.

```
procedure Convert
  (Str                :      String;
   To_Codeset         :      String;
   From_Codeset       :      String;
   Bytes_Read         : out   Natural;
   Bytes_Written      : out   Natural;
   Error              :      GError_Access := null;
   Result             : out   String);
```

Convert a string from one character set to another.

Str: String to convert Result: String converted, if no error. To_Codeset: Name of character set into which to convert Str From_Codeset: Character set of Str. Bytes_Read: Number of bytes in the input string that were successfully converted. Even if the conversion was successful, this may be less than Len if there were partial characters at the end of the input. If the error Illegal_Sequence occurs, the value stored will the byte offset after the last valid input sequence. Bytes_Written: Number of bytes stored in the output buffer. Error: Location to store the error occuring, ignored if null. Any of the errors in Convert_Error_Domain may occur.

```
function Convert
  (Str                :      String;
   To_Codeset         :      String;
   From_Codeset       :      String;
   Error              :      GError_Access := null)
   return String;
```

Same as above, but return a String directly.

```
procedure Convert
  (Str                :      chars_ptr;
   Len                :      Natural;
   To_Codeset         :      String;
   From_Codeset       :      String;
   Bytes_Read         : out   Natural;
   Bytes_Written      : out   Natural;
   Error              :      GError_Access := null;
   Result             : out   String);
```

Same as Convert procedure, but take a C string as input.

```
function Convert
  (Str                :      String;
   To_Codeset         :      String;
   From_Codeset       :      String;
   Bytes_Read         : access Natural;
   Bytes_Written      : access Natural;
   Error              :      GError_Access := null)
   return chars_ptr;
```

Same as Convert procedure, but return the result as a C string.

```
function Convert
  (Str                : chars_ptr;
   Len                : Natural;
   To_Codeset         : String;
   From_Codeset       : String;
   Bytes_Read         : access Natural;
   Bytes_Written      : access Natural;
   Error              : GError_Access := null)
   return chars_ptr;
```

Same as Convert procedure, but take and return the result as a C string.

```
procedure Locale_To_UTF8
  (OS_String          : String;
   Bytes_Read         : out  Natural;
   Bytes_Written      : out  Natural;
   Error              : GError_Access := null;
   Result             : out  String);
```

Convert a string which is in the encoding used for strings by the C runtime (usually the same as that used by the operating system) in the current locale into a UTF-8 string.

OS_String: A string in the encoding of the current locale Bytes_Read: Number of bytes in the input string that were successfully converted. Even if the conversion was successful, this may be less than Len if there were partial characters at the end of the input. If the error Illegal_Sequence occurs, the value stored will the byte offset after the last valid input sequence. Bytes_Written: Number of bytes stored in Result. Error: Location to store the error occuring, ignored if null. Any of the errors in Convert_Error_Domain may occur.

```
function Locale_To_UTF8
  (OS_String          : String;
   Bytes_Read         : access Natural;
   Bytes_Written      : access Natural;
   Error              : GError_Access := null)
   return chars_ptr;
```

Same as procedure Locale_To_UTF8, but return the raw C string for efficiency. The caller is responsible for freeing the resulting string.

```
function Locale_To_UTF8
  (OS_String          : String)
   return String;
```

Same as procedure Locale_To_UTF8, but return only the String.

```
procedure Locale_From_UTF8
  (UTF8_String        : String;
   Bytes_Read         : out  Natural;
   Bytes_Written      : out  Natural;
   Error              : GError_Access := null;
   Result             : out  String);
```

Convert a string from UTF-8 to the encoding used for strings by the C runtime (usually the same as that used by the operating system) in the current locale.

UTF8_String: A UTF-8 encoded string Bytes_Read: Number of bytes in the input string that were successfully converted. Even if the conversion was successful, this may be less than Len if there were partial characters at the end of the input. If the error Illegal_Sequence occurs, the value stored will the byte offset after the last valid input sequence. Bytes_Written: Number of bytes stored in the output buffer. Error: Location to store the error occuring, ignored if null. Any of the errors in Convert_Error_Domain may occur.

```
function Locale_From_UTF8
  (UTF8_String        :      String;
   Bytes_Read         : access Natural;
   Bytes_Written      : access Natural;
   Error              :      GError_Access := null)
   return chars_ptr;
```

Same as procedure Locale_From_UTF8, but return the raw C string for efficiency. The caller is responsible for freeing the resulting string. Use the C "free" function to free this.

```
function Locale_From_UTF8
  (UTF8_String        :      String)
   return String;
```

Same as procedure Locale_From_UTF8, but return only the String.

```
function Filename_To_UTF8
  (OS_String          :      String;
   Error              :      GError_Access := null)
   return String;
```

Convert a string which is in the encoding used for filenames into a UTF-8 string.

```
function Filename_From_UTF8
  (UTF8_String        :      String;
   Error              :      GError_Access := null)
   return String;
```

Convert a string from UTF-8 to the encoding used for filenames.

```
function Filename_From_URI
  (URI                :      String;
   Hostname           : access chars_ptr;
   Error              :      GError_Access := null)
   return String;
```

Convert an escaped UTF-8 encoded URI to a local filename in the encoding used for filenames.

URI: A uri describing a filename (escaped, encoded in UTF-8). Hostname: Location to store hostname for the URI. If there is no hostname in the URI, null will be stored in this location. Error: Location to store the error occuring, ignored if null. Any of the errors in Convert_Error_Domain may occur.

```
function Filename_To_URI
  (Filename           :      String;
   Hostname           :      String := "";
   Error              :      GError_Access := null)
   return String;
```

Convert an absolute filename to an escaped UTF-8 encoded URI.

Filename: An absolute filename specified in the encoding used for filenames by the operating system. Hostname: A UTF-8 encoded hostname, or "" for none. Error: Location to store the error occuring, ignored if null. Any of the errors in Convert_Error may occur.

```
function Escape_Text
  (S                  :      String)
   return String;
```

Escape the text so that it is interpreted as-is by the Pango markup language

3 Package Glib.Error

This package provides definitions for the error handling mechanism used in Glib, Gdk and Gtk.

3.1 Types

type GError **is new** C_Proxy;

type GError_Access **is access all** GError;

3.2 Subprograms

```
function Error_New
  (Domain             :      GQuark;
   Code               :      Gint;
   Message            :      String)
   return GError;
```

Create a new GError object.

```
procedure Error_Free
  (Error              :      GError);
```

Free the memory associated with a GError.

```
function Error_Copy
  (Error              :      GError)
   return GError;
```

Duplicate a GError object.

```
function Error_Matches
  (Error              :      GError;
   Domain             :      GQuark;
   Code               :      Gint)
   return Boolean;
```

Return whether a given GError matches a domain/code.

```
function Get_Domain
  (Error              :      GError)
   return GQuark;
```

Return the domain associated with a GError.

```
function Get_Code
  (Error              :      GError)
   return Gint;
```

Return the code associated with a GError.

```
function Get_Message
  (Error              :      GError)
   return String;
```

Return the message associated with a GError.

4 Package Glib.GSlist

This packages provides the implementation of a generic single-linked list. One instantiation is found in Gtk.Widget.Widget_Slist for a list of widgets.

See the documentation of Glib.Glist for more information, it provides the same API as this package. Single linked lists are traversed the same way as double-linked lists, even though most subprograms are less efficient than their double-linked counterparts.

5 Package Glib.Generic_Properties

Note: this package need only be used and understood by people who want to create their own new widgets and their associated properties. Normal usage of properties doesn't require any deep understanding of this package.

This package provides two generic subpackages that make it easy to declare properties. Each of these packages define two types:

@itemize @bullet @item Property_RO : this type should be used for a read-only property of the given type. @item Property : This is for read-write properties

@end itemize Each of these two types is associated with one or two primitive operations Get_Property and Set_Property, that allows the modification of properties of this type.

As a user and creator of new widgets, you should always use the Generic_Enumeration_Property package, since it also registers the enumeration type with gtk+ for a full compatibility with C.

5.1 Types

type Discrete_Type **is**
(<>);

type Enumeration **is**
(<>);

type Property **is new** Properties.Property;

type Property_RO **is new** Properties.Property_RO;

5.2 Subprograms

5.2.1 Generic package for discrete type properties

This package should be used to implement the@* Get_Property and Set_Property subprograms for all properties related to enumeration types and simple types. This should be used only for types defined in GtkAda or gtk+ themselves, not for types that you define yourself. Use Generic_Discrete_Type instead.

```
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property;
   Value              :      Discrete_Type);
```

Set a property of Object based on Enumeration_Type.

```
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property)
   return Discrete_Type;
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_RO)
   return Discrete_Type;
```

Get a property from Object

5.2.2 Types

```
function Get_Type              return Glib.GType;
```

Return the internal gtk+ type associated with the Ada enumeration
Enumeration. You don't need to use such a function for the types defined in standard in GtkAda. Use Glib.Type_From_Name instead.

```
function Gnew_Enum
  (Name, Nick, Blurb  :      String;
   Default            :      Enumeration
                      := Enumeration'First;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
```

Create a new param_spec (to describe properties), based on the
Ada enumeration type Enumeration. This function is used when creating the property with Install_Property on an object. Name, Nick and Blurb should describe the property, not its type.

5.2.3 Values

```
function Get_Enum
  (Value              :      Glib.Values.GValue)
   return Enumeration;
```

Return the enumeration contained in Value, assuming it is of type
Enumeration

```
procedure Set_Enum
  (Value              : in out Glib.Values.GValue;
   Enum               :      Enumeration);
```

Set the enumeration value for Value. This properly initializes the
type of Value, so you don't need to call Init yourself.

5.2.4 Generic package for record types properties

This package should be used to implement the@* Get_Property and Set_Property subprograms for all properties related to record type, like Gdk_Color and Gdk_Rectangle. This should be used only for types defined in GtkAda or gtk+ themselves, not for types that you define yourself.

```
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property;
   Value              :      Boxed_Type);
```

Set a property of Object based on Enumeration_Type.

```
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property)
   return Boxed_Type;
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_RO)
   return Boxed_Type;
```

Get a property from Object.
Unset_Value is raised if the property is not set

```
procedure Set_Value
  (Value              : out   Glib.Values.GValue;
   Val                :      Boxed_Type);
```

Store Val in Value. The latter is properly initialized, and reference counting is handled automatically. You must Unset Value when you are done using it.

```
function Get_Value
  (Value              :      Glib.Values.GValue)
   return Boxed_Type;
```

Get the value stored in Value. Reference counting is automatically handled, and the returned value has been properly referenced. Unset_Value is raised if Value contains no data

6 Package Glib.Glist

This package implements a generic double-linked list. Such lists are used throughout GtkAda to contain lists of widgets (for the children of containers, or for the list of selected widgets in a Gtk_Clist for instance), list of strings (for Gtk_Combo_Box),...

They provide a common interface to traverse these lists.

One useful note: you should only Free the lists that you have allocated yourself, and not the lists that are returned by the subprograms in GtkAda and should be left under GtkAda's control.

See the example below for an example on how to traverse a list.

Instantiating the package Generic_List requires two functions to convert back and forth between your data type and a System.Address which is the type stored at the C level. Note that the lists used in GtkAda already have associated packages, like Gtk.Enums.Gint_List, Gtk.Enums.String_List or Gtk.Widget.Widget_List.

6.1 Types

type Glist **is private**;

This type is both a list and an item in the list. Each item points to its successor.

6.2 Subprograms

```
procedure Alloc
  (List               : out   Glist);
```

Allocate a new item in the list.
This item isn't associated with any data. You probably don't have to use this subprogram, since Append, Insert, Prepend, etc. already handle the allocation for you and give a new value to the item.

```
procedure Append
  (List               : in out Glist;
   Data               :      Gpointer);
```

Add a new item at the end of the list, and stores the new list
directly back in List. The complexity of this operation is O(n)

```
function Concat
  (List1              :      Glist;
   List2              :      Glist)
   return Glist;
```

Concatenate two lists, and return the result.
List2 is added at the end of List1. The complexity is O(n1) (depends on the size of List1).

```
procedure Insert
  (List               : in out Glist;
   Data               :      Gpointer;
   Position           :      Gint);
```

Insert an item in the middle of a list.
If Position is 0, the item is added at the beginning of the list, if it is negative the item is added at the end. The complexity is O(Position).

```
function Find
  (List               :      Glist;
   Data               :      Gpointer)
   return Glist;
```

Find a value in the list, and return the first item that contains it.
Note that this function will not work if the function Convert does not return the same value for two identical values.

```
function First
  (List               :      Glist)
   return Glist;
```

Return the first item in the list.
Note that if List is in fact an item of a larger list, the return value is the first item in the larger list itself.

```
procedure Free
  (List               : in out Glist);
```

Free the list (but does not free the data in each of its elements).
This only frees the memory associated with the list itself. You should only use this function on the lists that you have created yourself, not on the list that are returned by some functions in GtkAda (like Gtk.Clist.Get_Selection). These functions return directly the list managed by the underlying C widget, and you should never free the result yourself.

Note also that the memory might not be actually freed. For efficiency reasons, GtkAda will keep the memory allocated and try to reuse it as much as possible.

```
function Get_Data
  (List               :      Glist)
   return Gpointer;
```

Return the value pointed to by List.
The System.Address container in the C list is converted to a Gpointer through a call to Convert.

```
function Get_Data_Address
  (List               :      Glist)
   return System.Address;
```

Return directly the System.Address contained in the C list.
This is used mainly internally in GtkAda to implement String lists, and you should not have to use this subprogram yourself.

```
function Index
  (List               :      Glist;
   Data               :      Gpointer)
   return Gint;
```

Return the index of the first element in List that contains Data.
Note that this function is irrelevant if Convert does not return the same value for two identical data.

```
function Last
  (List               :      Glist)
   return Glist;
```

Return the last element in the list.

```
function Length
  (List               :      Glist)
   return Guint;
```

Return the number of elements in the list.
The last item's index is Length - 1.

```
procedure List_Reverse
  (List               : in out Glist);
```

Reverse the order of the list (the last item becomes the first, etc.)

```
function Next
  (List               :      Glist)
   return Glist;
```

Returns the Item following List in the global list that contains
both. If there is no such item, return Null_List. This is how you stop iterating over a list.

```
function Nth
  (List               :      Glist;
   N                  :      Guint)
   return Glist;
```

Give the nth item following LIST in the global list that
contains both. If there is no such item, return Null_List.

```
function Nth_Data
  (List               :      Glist;
   N                  :      Guint)
   return Gpointer;
```

Return the Data contained in the N-th item of List.
The result is undefined if there is no such item in the list. The actual result in that case is the result of Convert (System.Null_Address); which might not mean anything.

```
function Position
  (List               :      Glist;
   Link               :      Glist)
   return Gint;
```

Return the position of Link in the List.
If Link is not contained in the list, -1 is returned.

```
procedure Prepend
  (List               : in out Glist;
   Data               :      Gpointer);
```

Add an item at the beginning of the list.
This operation always succeed.

```
function Prev
  (List               :      Glist)
   return Glist;
```

Return the item before List in the global list that contains both.
Return Null_List if there is no such item.

```
procedure Remove
  (List               : in out Glist;
   Data               :      Gpointer);
```

Remove the first item in List that contains Data.
Note that this operation can succeed only if Convert always return the same address for a given value.

```
procedure Remove_Link
  (List               : in out Glist;
   Link               :      Glist);
```

Remove Link from the list to which it belongs.
If that list is not List, no error is returned, but Link is removed anyway.

```
function Is_Created
  (List               :      Glist)
   return Boolean;
```

Return True if there is a C widget associated with List.

6.3 Example

```
with Glib; use Glib;
with Gtk.Enums;
with Ada.Text_IO; use Ada.Text_IO;

procedure Glist_Traverse is
   use Gtk.Enums.Gint_List;
   List : Gtk.Enums.Gint_List.Glist;
   Temp : Gtk.Enums.Gint_List.Glist;
begin

   --  First step: create a new list.

   Prepend (List, 2);                       -- add at the beginning of the list
   Append (List, 3);                        -- add at the end of the list
   Insert (List, Data => 1, Position => 1); -- in the middle of the list

   --  Traverse the list (first way)

   Temp := First (List);
   while Temp /= Null_List loop
      Put_Line (Gint'Image (Get_Data (Temp)));
      Temp := Next (Temp);
   end loop;

   --  Traverse the list (second way)

   for I in 1 .. Length (List) loop
      Put_Line (Gint'Image (Nth_Data (List, I - 1)));
   end loop;

end Glist_Traverse;
```

7 Package Glib.Graphs

General implementation for a graph. This provides a representation for a graph structure, with nodes (vertices) connected by links (edges). It is not intended for huges, highly-connected graphs, since there are several lists provided for efficient access to ancestor and children nodes.

7.1 Types

type Breadth_Vertices_Array **is array** (Natural **range** <>) **of** Breadth_Record;

type Connected_Component **is record**
```
Vertices : Vertices_Array (1 .. Num_Vertices);
Next     : Connected_Component_List;
end record;
```

type Connected_Component_List **is access** Connected_Component;

type Depth_Vertices_Array **is array** (Natural **range** <>) **of** Depth_Record;

type Edge **is abstract tagged private**;

type Edge_Access **is access all** Edge'Class;

General access types to vertices and edges

type Edge_Iterator **is private**;

Iterators other the vertices and edges of the graph

type Edges_Array **is array** (Natural **range** <>) **of** Edge_Access;

type Graph **is private**;

type Reverse_Edge_Callback **is access procedure**
```
(G : Graph; Edge : Edge_Access);
```

Callback called when the two ends of the edge should be reverted, so as to make the graph acyclick

```
type Vertex is abstract tagged private;

type Vertex_Access is access all Vertex'Class;

type Vertex_Iterator is private;

type Vertices_Array is array (Natural range <>) of Vertex_Access;
```

7.2 Subprograms

7.2.1 Modifying a graph

```
procedure Set_Directed
  (G                  : in out Graph;
   Directed           :      Boolean);
```

Indicate whether the graph is oriented.

```
function Is_Directed
  (G                  :      Graph)
   return Boolean;
```

Return True if the graph is oriented

```
procedure Add_Vertex
  (G                  : in out Graph;
   V                  : access Vertex'Class);
```

Add a new vertex to the graph.

```
procedure Add_Edge
  (G                  : in out Graph;
   E                  : access Edge'Class;
   Source, Dest       : access Vertex'Class);
```

Add a new edge to the graph.

```
procedure Destroy
  (E                  : in out Edge);
```

Destroy the memory occupied by the edge. This doesn't remove the edge from the graph. You should override this subprogram for the specific edge type you are using. This subprogram shouldn't (and in fact can't) free E itself.

```
procedure Destroy
  (V                  : in out Vertex);
```

Destroy the memory occupied by the vertex. This doesn't remove the vertex from the graph. This subprogram must be overriden. This subprogram shouldn't (and in fact can't) free V itself.

```
procedure Destroy
  (G                  : in out Graph);
```

Destroy all the nodes and edges of the graph, and then free the memory occupied by the graph itself

```
procedure Clear
  (G                  : in out Graph);
```

Remove all the nodes and edges of the graph.

```
procedure Remove
  (G                  : in out Graph;
   E                  : access Edge'Class);
```

Remove the edge from the graph. The primitive
subprogram Destroy is called for the edge. Any iterator currently pointing to E becomes invalid

```
procedure Remove
  (G                  : in out Graph;
   V                  : access Vertex'Class);
```

Remove the vertex from the graph.
Destroy is called for the vertex. Note that all the edges to or from the vertex are destroyed (see Remove above). Any iterator currently pointing to V becomes invalid

```
function Is_Acyclic
  (G                  :      Graph)
   return Boolean;
```

Return True if G contains no cycle. Note that this requires a
depth-first search, the running time is thus O (edges + vertices). G must be oriented

```
function Get_Src
  (E                  : access Edge)
   return Vertex_Access;
function Get_Dest
  (E                  : access Edge)
   return Vertex_Access;
```

Return the source and destination for a given edge

```
function In_Degree
  (G                  :      Graph;
   V                  : access Vertex'Class)
   return Natural;
function Out_Degree
  (G                  :      Graph;
   V                  : access Vertex'Class)
   return Natural;
```

Return the number of edges ending on V, or starting from V.

```
procedure Move_To_Front
  (G                  : in out Graph;
   V                  : access Vertex'Class);
```

Move V to the front of the list of vertices in the graph, so that the
iterators will return this item first. All iterators become obsolete.

```
procedure Move_To_Back
  (G                  : in out Graph;
   V                  : access Vertex'Class);
```

Move V to the back of the list of vertices in the graph, so that the
iterators will return this item last. All iterators become obsolete.

```
function Get_Index
  (V                  : access Vertex)
   return Natural;
```

Return the uniq index associated with the vertex. Each vertex has a
different index from 0 to Max_Index (Graph)

```
function Max_Index
  (G                  :      Graph)
   return Natural;
```

Return the maximum index used for vertices in the graph.

7.2.2 Breadth First Search

This search algorithm traverse the tree layer after layer (first the@* nodes closer to the specified root, then the grand-children of this root, and so on).

```
function Breadth_First_Search
  (G                  :      Graph;
   Root               : access Vertex'Class)
   return Breadth_Vertices_Array;
```

Traverse the tree Breadth_First, and sort the nodes accordingly.
The returned list is sorted so that all nodes at a distance k from Root are found before the nodes at a distance (k+1). The running time is O(vertices + edges).

7.2.3 Depth First Search

This algorithm traverse the tree in depth, ie all the descendents of the@* first child are found before the second child. This algorithm has several properties: it can indicate whether the graph is cyclic. Moreover, the subgraph formed by all the nodes and the edges between a vertex and its predecessor (see the structure Depth_Record) is a tree. If the graph is acyclic, then the resulting array is sorted topologically: if G contains an edge (u, v), then u appears before v.

The running time for this algorithm is O(vertices + edges)

```
procedure Revert_Edge
  (G                  :      Graph;
   E                  :      Edge_Access);
```

Revert the two ends of Edge. This is meant to be used as a callback for
Depth_First_Search so as to make the graph acyclic.

```
function Depth_First_Search
  (G                  :      Graph)
   return Depth_Vertices_Array;
```

Traverse the tree Depth_First.

```
function Depth_First_Search
  (G                  :      Graph;
   Acyclic            : access Boolean;
   Reverse_Edge_Cb    :      Reverse_Edge_Callback := null)
   return Depth_Vertices_Array;
```

Same as above, but Acyclic is also modified to indicate whether G is
acyclic. If Reverse_Edge_Cb is not null, then it is called to reverse the ends of selected edges, so that the final graph is acyclic. Note that you *must* revert the ends, or there will be an infinite loop. You might also want to mark the edge as reverted somehow, so as to draw the arrows on the correct side, if your application is graphical.

If Reverse_Edge_Cb is null, no edge is reverted, and the graph is unmodified.

7.2.4 Strongly connected components

Strongly connected components in a directed graph are the maximal set of@* vertices such that for every pair {u, v} of vertices in the set, there exist a path from u to v and a path from v to u. Two vertices are in different strongly connected components if there exist at most one of these paths.

```
procedure Free
  (List                : in out Connected_Component_List);
```

Free the list of strongly connected components

```
function Strongly_Connected_Components
  (G                   :      Graph)
   return Connected_Component_List;
```

Return the list of strongly connected components.
This is a linear time algorithm O(vertices + edges).

```
function Strongly_Connected_Components
  (G                   :      Graph;
   DFS                 :      Depth_Vertices_Array)
   return Connected_Component_List;
```

Same as above, but a depth-first search has already been run on G, and
we reuse the result. This is of course more efficient than the previous function.

7.2.5 Minimum spanning trees

A minimum spanning tree is a subset of the edges of G that forms a@* tree (acyclic) and connects all the vertices of G. Note that the number of edges in the resulting tree is always (number of vertices of G) - 1

```
function Kruskal
  (G                   :      Graph)
   return Edges_Array;
```

Return a minimum spanning tree of G using Kruskal's algorithm.
This algorithm runs in O(E * log E), with E = number of edges.

7.2.6 Vertex iterator

```
function First
  (G                   :      Graph)
   return Vertex_Iterator;
```

Return a pointer to the first vertex.

```
procedure Next
  (V                   : in out Vertex_Iterator);
```

Moves V to the next vertex in the graph.

```
function At_End
  (V                   :      Vertex_Iterator)
   return Boolean;
```

Return True if V points after the last vertex

```
function Get
  (V                   :      Vertex_Iterator)
   return Vertex_Access;
```

Get the vertex pointed to by V

7.2.7 Edge iterator

```
function First
  (G                  :      Graph;
   Src, Dest          :      Vertex_Access := null;
   Directed           :      Boolean := True)
   return Edge_Iterator;
```

Return a pointer to the first edge from Src to Dest.
If either Src or Dest is null, then any vertex matches. Thus, if both parameters are nulll, this iterator will traverse the whole graph. Note: there might be duplicates returned by this iterator, especially when the graph is not oriented. Directed can be used to temporarily overrides the setting in the graph: If Directed is True, the setting of G is taken into account. If Directed is False, the setting of G is ignored, and the graph is considered as not directed.

```
procedure Next
  (E                  : in out Edge_Iterator);
```

Moves V to the next edge in the graph.

```
function At_End
  (E                  :      Edge_Iterator)
   return Boolean;
```

Return True if V points after the last edge

```
function Get
  (E                  :      Edge_Iterator)
   return Edge_Access;
```

Get the edge pointed to by E.

```
function Repeat_Count
  (E                  :      Edge_Iterator)
   return Positive;
```

Return the number of similar edges (same ends) that were found before,
and including this one). For instance, if there two edges from A to B, then the first one will have a Repeat_Count of 1, and the second 2.

```
procedure Add
  (List               : in out Edge_List;
   E                  : access Edge'Class);
```

Add a new element to List.
Edges are inserted in the list so that edges with similar ends are next to each other.

```
procedure Remove
  (List               : in out Edge_List;
   E                  : access Edge'Class);
```

Remove an element from List

```
function Length
  (List               :      Edge_List)
   return Natural;
```

Return the number of elements in the list

```
procedure Add
  (List               : in out Vertex_List;
   V                  : access Vertex'Class);
procedure Internal_Remove
  (G                  : in out Graph;
   V                  : access Vertex'Class);
```

8 Package Glib.Main

This package contains low-level subprograms that are used to interact or configure the main loop. This loop is responsible for processing events, monitoring input sources like pipes, sockets,..., and calling callbacks at given time intervals. New event sources can be created.

To allow multiple independent sets of sources to be handled in different threads, each source is associated with a main context. A main context can only be running in a single thread, but sources can be added to it and removed from it from other threads.

Each event source is assigned a priority. The default priority, G_PRIORITY_DEFAULT, is 0. Values less than 0 denote higher priorities. Values greater than 0 denote lower priorities. Events from high priority sources are always processed before events from lower priority sources.

Idle functions can also be added, and assigned a priority. These will be run whenever no events with a higher priority are ready to be processed.

The GMainLoop data type represents a main event loop. A GMainLoop is created with g_main_loop_new(). After adding the initial event sources, g_main_loop_run() is called. This continuously checks for new events from each of the event sources and dispatches them. Finally, the processing of an event from one of the sources leads to a call to g_main_loop_quit() to exit the main loop, and g_main_loop_run() returns.

It is possible to create new instances of GMainLoop recursively. This is often used in GTK+ applications when showing modal dialog boxes. Note that event sources are associated with a particular GMainContext, and will be checked and dispatched for all main loops associated with that GMainContext.

Creating new sources types ==========

One of the unusual features of the GTK+ main loop functionality is that new types of event source can be created and used in addition to the builtin type of event source. A new event source type is used for handling GDK events.

New source types basically interact with with the main context in two ways. Their prepare function in GSourceFuncs can set a timeout to determine the maximum amount of time that the main loop will sleep before checking the source again. In addition, or as well, the source can add file descriptors to the set that the main context checks using g_source_add_poll().

Ada ===

Some of these features duplicate Ada builtin tasking support, but the latter might be more complex to use in the context of a graphical application, since most of the time the windowing system doesn't support multi-threaded applications.

8.1 Types

type Data_Type **is private**;

type Destroy_Notify **is access procedure**
(Data : **in out** Data_Type);

Notify is called just prior to the destruction of Data. It is also called if the idle or timeout is destroyed through a call to Remove (Id);

type G_Main_Context **is new** Glib.C_Proxy;

This type represents a set of sources to handled in the main loop. Basically, this represents a main loop. There might be several main loops running at the same time, although gtk+ itself has only one, identified as the default main context.

type G_Priority **is new** Gint;

type G_Source **is new** Glib.C_Proxy;

This type represents an event source that can be monitored by the main loop. There are various internal types of such sources, that can be configured by setting appropriate callbacks (this is not yet doable in GtkAda). See Idle_Source_New and Timeout_Source_New.

```
type G_Source_Func is access function
          (Data : Data_Type) return Boolean;
```

If the function returns FALSE it is automatically removed from the list of event sources and will not be called again.

```
type G_Source_Func_User_Data is access function
          (User_Data : System.Address) return Gboolean;
```

type G_Source_Id **is new** Guint;

The ID of a source within the context to which it is attached.

type G_Source_Type **is private**;

```
type Source_Check_Func is access function
          (Source : G_Source) return Gboolean;
```

```
type Source_Dispatch_Func is access function
          (Source   : G_Source;
           Callback : G_Source_Func_User_Data;
           Data     : System.Address) return Gboolean;
```

```
type Source_Finalize_Func is access procedure
        (Source : G_Source);
```

```
type Source_Prepare_Func is access function
        (Source : G_Source; Timeout : access Gint) return Gboolean;
```

8.2 Subprograms

8.2.1 G_Main_Context

```
function Main_Context_New      return G_Main_Context;
```

Create a new context

```
procedure Main_Context_Ref
  (Context            :      G_Main_Context);

procedure Main_Context_Unref
  (Context            :      G_Main_Context);
```

Increase or decreate the reference counting for Context. When this reaches 0, the memory is freed.

```
function Main_Context_Default  return G_Main_Context;
```

Returns the default main context. This is the main context used for main loop functions when a main loop is not explicitly specified.

```
procedure Wakeup
  (Context            :      G_Main_Context);
```

If context is currently waiting in a poll(), interrupt the poll(), and continue the iteration process.

```
function Acquire
  (Context            :      G_Main_Context)
   return Boolean;
```

Tries to become the owner of the specified context. If some other thread is the owner of the context, returns FALSE immediately. Ownership is properly recursive: the owner can require ownership again and will release ownership when Release() is called as many times as Acquire(). You must be the owner of a context before you can call Prepare(), Query(), Check(), Dispatch().

```
procedure Release
  (Context            :      G_Main_Context);
```

Releases ownership of a context previously acquired by this thread with Acquire(). If the context was acquired multiple times, the only release ownership when Release() is called as many times as it was acquired.

```
function Is_Owner
  (Context            :      G_Main_Context)
   return Boolean;
```

Determines whether this thread holds the (recursive) ownership of this context. This is useful to know before waiting on another thread that may be blocking to get ownership of context.

```
procedure Dispatch
  (Context            :      G_Main_Context);
```

Dispatches all pending sources.

8.2.2 Main loop

```
function Depth                  return Integer;
```

The main loop recursion level in the current thread. It returns 0 when called from the toplevel.

8.2.3 G_Source

```
function Default_Dispatch
  (Source             :      G_Source;
   Cb                 :      G_Source_Func_User_Data;
   Data               :      System.Address)
   return Gboolean;
function G_Source_Type_New
  (Prepare            :      Source_Prepare_Func;
   Check              :      Source_Check_Func;
   Dispatch           :      Source_Dispatch_Func
                      := Default_Dispatch'Access;
   Finalize           :      Source_Finalize_Func := null)
   return G_Source_Type;
```

Create a new type of sources.
This function is specific to GtkAda. The returned value is never freed. Most of the time, you do not need to create a new source type, or even call Source_New. Most things can be implemented through the careful use of Idle and Timeout callbacks. However, creating a new source type allows for cleaner code, by sharing the common part of the handling.

For idle sources, the prepare and check functions always return TRUE to indicate that the source is always ready to be processed. The prepare function also returns a timeout value of 0 to ensure that the poll() call doesn't block (since that would be time wasted which could have been spent running the idle function).

For timeout sources, the prepare and check functions both return TRUE if the timeout interval has expired. The prepare function also returns a timeout value to ensure that the poll() call doesn't block too long and miss the next timeout.

For file descriptor sources, the prepare function typically returns FALSE, since it must wait until poll() has been called before it knows whether any events need to be processed. It sets the returned timeout to -1 to indicate that it doesn't mind how long the poll() call blocks. In the check function, it tests the results of the poll() call to see if the required condition has been met, and returns TRUE if so.

```
function Source_New
  (Source_Type        :      G_Source_Type;
   User_Data          :      System.Address)
   return G_Source;
```

Creates a new GSource structure.

The source will not initially be associated with any GMainContext and must be added to one with Attach() before it will be executed.

```
function Get_User_Data
  (Source             :      G_Source)
   return System.Address;
```

Return the user data passed to Source_New. This only applies to sources created through that function, and returns undefined results (or even segfaults) otherwise

```
procedure Source_Ref
  (Source             :      G_Source);
procedure Source_Unref
  (Source             :      G_Source);
```

Increase or decrease the reference counting for Source. When this reaches 0, the Source is destroyed

```
procedure Source_Destroy
  (Source             :      G_Source);
```

Removes the source from its context, and mark it as destroyed (the memory is not reclaimed while the reference counting doesn't reach 0). Source cannot be added to another context.

```
function Attach
  (Source             :      G_Source;
   Context            :      G_Main_Context := null)
   return G_Source_Id;
```

Add Source to Context. The Source will be executed within that context. If context is null, the source is added to the default context. Returns the Id of the source within Context.

```
function Remove
  (Id                 :      G_Source_Id)
   return Boolean;
procedure Remove
  (Id                 :      G_Source_Id);
```

Removes the source with the given id from the default main context. The id of. Return True if the source was found and removed

```
procedure Set_Priority
  (Source             :      G_Source;
   Priority           :      G_Priority);
function Get_Priority
  (Source             :      G_Source)
   return G_Priority;
```

Sets the priority of a source. While the main loop is being run, a source will be dispatched if it is ready to be dispatched and no sources at a higher (numerically smaller) priority are ready to be dispatched.

```
procedure Set_Can_Recurse
  (Source             :      G_Source;
   Can_Recurse        :      Boolean);
function Get_Can_Recurse
  (Source             :      G_Source)
   return Boolean;
```

Sets whether a source can be called recursively. If can_recurse is TRUE, then while the source is being dispatched then this source will be processed normally. Otherwise, all processing of this source is blocked until the dispatch function returns.

```
function Get_Id
  (Source             :      G_Source)
   return G_Source_Id;
```

Returns the numeric ID for a particular source. The ID of a source is positive integer which is unique within a particular main loop context. The reverse mapping from ID to source is done by Find_Source_By_Id

```
function Find_Source_By_Id
  (Id                 :      G_Source_Id;
   Context            :      G_Main_Context := null)
   return G_Source;
```

Find a source given a context and its Id.

```
function Get_Context
  (Source             :      G_Source)
   return G_Main_Context;
```

Gets the context with which the source is associated. Calling this function on a destroyed source is an error. The returned value is Null for sources that haven't been attached yet

8.2.4 Idle and timeout

```
function Idle_Source_New       return G_Source;
```

Return a newly allocated idle G_Source. Such a source is polled whenever the main loop is not processing events with a higher priority. This source must be attached to a main context before it will be executed.

```
function Timeout_Source_New
  (Interval           :      Guint)
   return G_Source;
```

Return a newly allocated idle G_Source. Such a source is called at regular intervals. Internval is in milliseconds.

```
function Idle_Add
  (Func               :      G_Source_Func)
   return G_Source_Id;
```

Adds a function to be called whenever there are no higher priority events pending in the default main loop. This function is given the priority Priority_Default_Idle. If the function returns False, it is automatically removed from the list of event sources and will not be called again. This function returns the Id of the event source. See Find_Source_By_Id. This is implemented by using Idle_Source_New internally.

```
function Timeout_Add
  (Interval           :      Guint;
   Func               :      G_Source_Func)
   return G_Source_Id;
```

Create a new function to be called periodically until it returns False.

Note that timeout functions may be delayed, due to the processing of other event sources. Thus they should not be relied on for precise timing. After each call to the timeout function, the time of the next timeout is recalculated based on the current time and the given interval (it does not try to 'catch up' time lost in delays).

```
function Idle_Add
  (Func               :      G_Source_Func;
```

```
   Data               :      Data_Type;
   Priority           :      G_Priority
                      := Priority_Default_Idle;
   Notify             :      Destroy_Notify := null)
   return G_Source_Id;
```

Adds a function to be called whenever there are no higher priority events pending.

```
function Timeout_Add
  (Interval           :      Guint;
   Func               :      G_Source_Func;
   Data               :      Data_Type;
   Priority           :      G_Priority := Priority_Default;
   Notify             :      Destroy_Notify := null)
   return G_Source_Id;
```

Adds a function to be called at regular intervals (in milliseconds).

```
procedure Set_Callback
  (Source             :      G_Source;
   Func               :      G_Source_Func;
   Data               :      Data_Type;
   Notify             :      Destroy_Notify := null);
```

Sets the callback function for a source. The callback for a source is called from the source's dispatch function.

The exact type of func depends on the type of source; ie. you should not count on func being called with data as its first parameter.

Typically, you won't use this function. Instead use functions specific to the type of source you are using.

```
procedure Free_Data
  (D                  :      System.Address);
function General_Cb
  (D                  :      System.Address)
   return Gint;
```

9 Package Glib.Messages

This package provides low level routines for enabling, disabling and modifying the way log messages are handled in glib/gdk/gtk.

9.1 Types

type Log_Function **is access procedure**

type Log_Handler_Id **is new** Guint;

type Log_Level_Flags **is mod** 2 ** 32;

log levels and flags.

9.2 Subprograms

9.2.1 log levels

```
function Log_Set_Handler
  (Log_Domain         :      String;
   Log_Levels         :      Log_Level_Flags;
   Log_Func           :      Log_Function)
   return Log_Handler_Id;
```

Set a log function for the given log levels, and return its id.

```
procedure Log_Remove_Handler
  (Log_Domain         :      String;
   Handler_Id         :      Log_Handler_Id);
```

Unset a given handler.

```
procedure Log_Default_Handler
  (Log_Domain         :      String;
   Log_Levels         :      Log_Level_Flags;
   Message            :      UTF8_String);
```

The default log handler.
Can be called e.g. within a user defined log handler.

```
procedure Log
  (Log_Domain         :      String;
   Log_Levels         :      Log_Level_Flags;
   Message            :      UTF8_String);
```

Log a message through the glib logging facility.

```
function Log_Set_Fatal_Mask
  (Log_Domain         :      String;
   Fatal_Mask         :      Log_Level_Flags)
   return Log_Level_Flags;
```

Set the level at which messages are considered fatal for a given domain.

```
function Log_Set_Always_Fatal
  (Fatal_Mask         :      Log_Level_Flags)
   return Log_Level_Flags;
```

Set the level at which messages are considered fatal for any domain.

10 Package Glib.Module

This package provides wrapper code for dynamic module loading

10.1 Types

type G_Module **is new** C_Proxy;

type Module_Flags **is mod** 2 ** 32;

type Pointer **is private**;

This is typically a pointer to procedure/function.

10.2 Subprograms

```
function Module_Supported      return Boolean;
```

Return True if dynamic module loading is supported

```
function Module_Open
  (File_Name          :      String;
   Flags              :      Module_Flags
                      := Module_Bind_Lazy)
   return G_Module;
```

Open a module 'file_name' and return handle, which is null on error.

```
function Module_Close
  (Module             :      G_Module)
   return Boolean;
```

Close a previously opened module, return True on success.

```
procedure Module_Make_Resident
  (Module             :      G_Module);
```

Make a module resident so Module_Close on it will be ignored

```
function Module_Error          return String;
```

Query the last module error as a string

```
procedure Generic_Module_Symbol
  (Module             :      G_Module;
   Symbol_Name        :      String;
   Symbol             : out  Pointer;
   Success            : out  Boolean);
```

Retrieve a symbol pointer from 'module'.
Success is set to True on success.

```
function Module_Name
  (Module             :      G_Module)
   return String;
```

Retrieve the file name from an existing module

```
function Module_Build_Path
  (Directory           :      String;
   Module_Name         :      String)
   return String;
```

Build the actual file name containing a module.
'directory' is the directory where the module file is supposed to be, or the null string in which case it should either be in the current directory or, on some operating systems, in some standard place, for instance on the PATH. Hence, to be absolutely sure to get the correct module, always pass in a directory. The file name consists of the directory, if supplied, and 'module_name' suitably decorated accoring to the operating system's conventions (for instance lib*.so or *.dll).

No checks are made that the file exists, or is of correct type.

11 Package Glib.Object

This package provides a minimal binding to the GObject type in Glib. See Glib.Properties for information on how to manipulate properties

11.1 Signals

- **"notify"**

  ```
  procedure Handler
  (Object : access GObject_Record'Class; Name : String);
  ```

  Emitted when the property Name has been modified

11.2 Types

type GObject_Class **is new** GType_Class;

type Interface_Vtable **is private**;

The virtual table of an interface (see Glib.Types). This is only useful when doing introspection.

type Signal_Id_Array **is array** (Guint **range** <>) **of** Glib.Signal_Id;

type Signal_Parameter_Types **is array** (Natural **range** <>, Natural **range** <>) **of** GType;

The description of the parameters for each event. These are the parameters that the application must provide when emitting the signal. The user can of course add his own parameters when connecting the signal in his application, through the use of Gtk.Handlers.User_Callback. Each event defined with Initialize_Class_Record below should have an entry in this table. If Gtk_Type_None is found in the table, it is ignored. For instance, a Signal_Parameter_Type like: (1 => (1 => Gdk_Type_Gdk_Event, 2 => GType_None), 2 => (1 => GType_Int, 2 => GType_Int)); defines two signals, the first with a single Gdk_Event parameter, the second with two ints parameters.

type Signal_Query **is private**;

type Weak_Notify **is access procedure**

11.3 Subprograms

```
function Is_Created
  (Object             :      GObject_Record'Class)
   return Boolean;
```

Return True if the associated C object has been created, False if
no C object is associated with Object. This is not the same as testing whether an access type (for instance any of the widgets) is "null", since this relates to the underlying C object.

```
function Get_Type
  (Object             : access GObject_Record)
   return GType;
```

Return the type of Object.
This function is mostly used internally, since in Ada you can simply test whether an object belong to a class with a statement like:

if Object in Gtk_Button_Record'Class then ...

which is easier.

11.3.1 Life cycle

```
procedure G_New
  (Object             : out   GObject);
```

Create a new GObject.
This is only required when you want to create an Ada tagged type to which you can attach new signals. Most of the time, you only need to directly create the appropriate Gtk Widget by calling the correct Gtk_New procedure.

```
procedure Ref
  (Object             : access GObject_Record);
```

Increment the reference counter for Object. See Unref below.
Since an object is not deleted while its reference count is not null, this is a way to keep an object in memory, in particular when you want to temporarily remove a widget from its parent.

```
procedure Unref
  (Object             : access GObject_Record);
```

Decrement the reference counter for Object. When this reaches 0, the
object is effectively destroy, all the callbacks associated with it are disconnected.

```
procedure Weak_Ref
  (Object             : access GObject_Record'Class;
   Notify             :      Weak_Notify;
   Data               :      System.Address
                      := System.Null_Address);
```

This kind of reference doesn't increment the object's reference
counting. However, it can and should be used to monitor the object's life cycle, in particular to detect is destruction. When Object is destroyed, calls Notify

```
procedure Weak_Unref
  (Object             : access GObject_Record'Class;
   Notify             :      Weak_Notify;
   Data               :      System.Address
                      := System.Null_Address);
```

Cancels the settings of Weak_Ref.

```
procedure Deallocate
  (Object             : access GObject_Record);
```

This operation is used to deallocate Object.
The default implementation assumes that the value passed in is an access value created by an allocator of the default pool, i.e. it will assume that an instance of Unchecked_Deallocation (GObject_Record'Class, GObject) can be used to deallocate the designated object. Types derived of GObject_Record can override this operation in order to cope with objects allocated on other pools or even objects allocated on the stack. This design is limited to support only one allocation strategy for each class, as the class tag is used to identify the applicable strategy.

11.3.2 Interfacing with C

The following functions are made public so that one can easily create@* new objects outside the Glib or Gtk package hierarchy. Only experienced users should make use of these functions.

```
function Get_Object
  (Object             : access GObject_Record'Class)
   return System.Address;
```

Access the underlying C pointer.

```
procedure Set_Object
  (Object             : access GObject_Record'Class;
   Value              :      System.Address);
```

Modify the underlying C pointer.

```
function Get_User_Data
  (Obj                :      System.Address;
   Stub               :      GObject_Record'Class)
   return GObject;
```

Return the Ada object matching the C object Obj. If Obj was created
explicitely from GtkAda, this will be the exact same widget. If Obj was created implicitely by gtk+ (buttons in complex windows,...), a new Ada object of type Stub will be created.

```
function Get_User_Data_Fast
  (Obj                :      System.Address;
   Stub               :      GObject_Record'Class)
   return GObject;
```

Same as Get_User_Data, but does not try to guess the type of Obj,
always default to Stub if Obj is unknown to GtkAda.

```
function Unchecked_Cast
  (Obj                : access GObject_Record'Class;
   Stub               :      GObject_Record'Class)
   return GObject;
```

Cast Obj in an object of tag Stub'Class.
Return the resulting object and free the memory pointed by Obj.

11.3.3 Signals

Any child of GObject can be associated with any number of signals. The@* mechanism for signals is fully generic, and any number of arguments can be associated with signals. See the function Initialize_Class_Record for more information on how to create new signals for your own new widgets. The subprograms below are provided for introspection: they make it

possible to query the list of signals defined for a specific widget, as well as their parameters and return types.

```
function Lookup
  (Object              :      Glib.GType;
   Signal              :      String)
   return Glib.Signal_Id;
```

Returns the signal Id associated with a specific Object/Signal pair.
Null_Signal_Id is returned if no such signal exists for Object. You can then use the Query procedure to get more information on the signal.

```
function List_Ids
  (Typ                 :      Glib.GType)
   return Signal_Id_Array;
```

Return the list of signals defined for Typ. You can get more information
on each of this signals by using the Query function below. See also the function Get_Type above to convert from an object instance to its type. Using a GType as the parameter makes it easier to find the signals for a widget and its ancestors (using Glib.Parent).

```
procedure Query
  (Id                  :      Glib.Signal_Id;
   Result              : out   Signal_Query);
```

Return the description associated with the signal Id. You can get the
various fields from Query with one of the functions below. Result is undefined if Id is Invalid_Signal_Id or Null_Signal_Id

```
function Id
  (Q                   :      Signal_Query)
   return Glib.Signal_Id;
```

Return the signal Id. Each Id is specific to a widget/signal name pair.
These Ids can then be used to temporarily block a signal for instance, through the subprograms in Gtk.Handlers.

```
function Signal_Name
  (Q                   :      Signal_Query)
   return Glib.Signal_Name;
```

Return the name of the signal, as should be used in a call to Connect.

```
function Return_Type
  (Q                   :      Signal_Query)
   return Glib.GType;
```

Return the type of object returned by the handlers for this signal.

```
function Params
  (Q                   :      Signal_Query)
   return GType_Array;
```

Return the list of parameters for the handlers for this signal

11.3.4 Creating new widgets

These types and functions are used only when creating new widget types@* directly in Ada. These functions initialize the classes so that they are correctly recognized by gtk+ itself See the GtkAda user's guide for more information on how to create your own widget types in Ada.

```
function Type_From_Class
  (Class_Record        :      GObject_Class)
```

```
   return GType;
```

Return the internal gtk+ type that describes the newly created Class_Record. See the function Glib.Types.Class_Peek for the opposite function converting from a GType to a GObject_Class.

11.3.5 Properties introspection

See glib.ads for more information on properties

```
function Interface_List_Properties
  (Vtable             :      Interface_Vtable)
   return Glib.Param_Spec_Array;
```

Return the list of properties of an interface (see also Glib.Properties) from a Vtable from Default_Interface_Peek). See also Class_List_Properties for a similar function for objects.

```
function Class_List_Properties
  (Class              :      GObject_Class)
   return Glib.Param_Spec_Array;
```

Return the list of all properties of the class.

11.3.6 Signals

??? This section is incomplete.

```
procedure Notify
  (Object             : access GObject_Record;
   Property_Name      :      String);
```

Emits the "notify" signal, to signal every listener that the property has been changed.

11.3.7 Lists

```
function Convert
  (W                  :      GObject)
   return System.Address;
function Convert
  (W                  :      System.Address)
   return GObject;
```

12 Package Glib.Properties

Properties are a fully general way to modify the appareance or behavior of widgets. Most of the time, there exists a faster way to modify the widget in the same fashion (for instance a direct call to a primitive subprogram). However, the properties provide a general scheme to modify these attributes. For instance, they can be used to provide introspection on the widget (to automatically retrieve the attributes that can be modified), or if you need to implement a tool like a GUI-Builder that is able to manipulate any widget, even those that didn't exist when the tool was written.

Two functions are provided for each type of property: Set_Property and Get_Property, which allow easy modification of specific widget properties. For instance, you could do the following: declare Button : Gtk_Button; begin Gtk_New (Button, "old label"); Set_Property (Button, Label_Property, "new label"); end; to modify the label of a button.

Likewise, you can retrieve the current label with: Current : String := Get_Property (Button, Label_Property);

Dispatching is used ensure type-safety while using properties. The appropriate Set_Property/Get_Property functions are called depending on the type of the property you are trying to use. This is checked statically by the compiler, which provides additional type-safety compared to the C library.

Note that some properties are read-only, and thus do not have the Set_Property subprogram defined.

When a property is modified, the signal "notify::<property>" is emitted, for instance, "notify::label" for a gtk_button. This is a standard gtk+ signal to which you can connect with the subprograms in gtk-handlers.ads

12.1 Types

type Property_Address **is new** Glib.Property;

type Property_Boolean **is new** Glib.Property;

type Property_C_Proxy **is new** Glib.Property;

type Property_Char **is new** Char_Properties.Property;

type Property_Char_RO **is new** Char_Properties.Property_RO;

type Property_Double **is new** Glib.Property;

```
type Property_Float is new Glib.Property;

type Property_Int is new Int_Properties.Property;

type Property_Long is new Long_Properties.Property;

type Property_Long_RO is new Long_Properties.Property_RO;

type Property_Object is new Glib.Property;

type Property_Object_WO is new Glib.Property;

type Property_String is new Glib.Property;

type Property_String_RO is new Glib.Property;

type Property_String_WO is new Glib.Property;

type Property_Uchar is new Uchar_Properties.Property;

type Property_Uchar_RO is new Uchar_Properties.Property_RO;

type Property_Uint is new Uint_Properties.Property;

type Property_Uint_RO is new Uint_Properties.Property_RO;

type Property_Ulong is new Ulong_Properties.Property;
```

type Property_Ulong_RO **is new** Ulong_Properties.Property_RO;

type Property_Unichar **is new** Unichar_Properties.Property;

12.2 Subprograms

```
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      String;
   Value              : in out Glib.Values.GValue);
procedure Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      String;
   Value              : in out Glib.Values.GValue);
```

Get the property. Value must have been initialized first with the expected type for the property, as in: Value : GValue; Init (Value, Value_Type (Pspec)); Get_Property (Object, Pspec_Name (Pspec), Value); If you do not have a Param_Spec, this can be replaced with: Init (Value, GType_Int); Get_Property (Object, Property_Name (Property), Value); Value must be Unset by the caller to free memory

```
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_String;
   Value              :      String);
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_String_WO;
   Value              :      String);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_String)
   return String;
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_String_RO)
   return String;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Boolean;
   Value              :      Boolean);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Boolean)
   return Boolean;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Object;
   Value              : access Glib.Object.GObject_Record'Class);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
```

```
   Name               :      Property_Object)
   return Glib.Object.GObject;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Object_WO;
   Value              : access Glib.Object.GObject_Record'Class);
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Address;
   Value              :      System.Address);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Address)
   return System.Address;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Float;
   Value              :      Gfloat);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Float)
   return Gfloat;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Double;
   Value              :      Gdouble);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_Double)
   return Gdouble;
procedure Set_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_C_Proxy;
   Value              :      C_Proxy);
function Get_Property
  (Object             : access Glib.Object.GObject_Record'Class;
   Name               :      Property_C_Proxy)
   return C_Proxy;
```

13 Package Glib.Properties.Creation

This package provides all the required subprograms to create and manipulate new properties associated with new widget types.

You do not have to be familiar with this package in order to use properties. See Glib.Object instead, that provides the minimal subprograms to get and set properties.

This package is only intended for writers of new widgets. You will need this function to create new properties.

Each object in gtk+ has a set of so-called properties. These are attributes that can be accessed, and possibly modified, by names. They provide introspection, that is an object can specify which properties it knows about, which can be modified,..., and thus provide a huge support for special applications like GUI-Builders that need to act on any kind of widgets, even those it doesn't know about yet.

However, for efficiency reasons, the properties are only names, and are not the only way to modify attributes of objects. It is often more efficient to use the alternate method, as documented in the GtkAda documentation for each property.

Another interesting feature of properties is that every time a property is modified, a signal "property_changed" or "notify" is emitted, and it is thus easy to keep track of attributes in objects.

13.1 Types

type Enum_Class **is new** Glib.C_Proxy;

type Enum_Value **is new** Glib.C_Proxy;

type Flags_Class **is new** Glib.C_Proxy;

type Flags_Int_Value **is mod** Glib.Gint'Last;

type Flags_Value **is new** Glib.C_Proxy;

type Get_Property_Handler **is access procedure**

type Param_Spec_Boolean **is new** Param_Spec;

type Param_Spec_Boxed **is new** Param_Spec;

type Param_Spec_Char **is new** Param_Spec;

type Param_Spec_Double **is new** Param_Spec;

type Param_Spec_Enum **is new** Param_Spec;

type Param_Spec_Flags **is new** Param_Spec;

type Param_Spec_Float **is new** Param_Spec;

type Param_Spec_Int **is new** Param_Spec;

type Param_Spec_Long **is new** Param_Spec;

type Param_Spec_Object **is new** Param_Spec;

type Param_Spec_Param **is new** Param_Spec;

type Param_Spec_Pointer **is new** Param_Spec;

type Param_Spec_String **is new** Param_Spec;

type Param_Spec_Uchar **is new** Param_Spec;

type Param_Spec_Uint **is new** Param_Spec;

type Param_Spec_Ulong **is new** Param_Spec;

type Param_Spec_Unichar **is new** Param_Spec;

type Property_Id **is new** Guint;

type Set_Property_Handler **is access procedure**

13.2 Subprograms

```
procedure Unref
  (Param              :      Param_Spec);
```

Decrement the reference counter. If it reaches 0, the memory is freed.

13.2.1 Enum classes

gtk+, a C library, has a whole system to describe its enumeration types,@* similar to what is available from the start in Ada ('Image and 'Value for instance). All enumerations are represented internally as Enum_Classes. However, there is no easy conversion between such an enum class and a GtkAda enumeration type. Most of the time, this has no impact on your work, since you know what type you need to use when calling an Ada function. However, you will need to manipulate these enumeration classes when interfacing with ParamSpecs and dealing with properties.

```
function Get_Value
  (Klass              :      Enum_Class;
   Value              :      Glib.Gint)
   return Enum_Value;
```

Return the value in Klass that is Value (equivalent of 'Val in Ada)

```
function Nth_Value
  (Klass              :      Enum_Class;
   Nth                :      Glib.Guint)
   return Enum_Value;
```

Return the Nth-th value in Klass, or null if there is no such value.

```
function Value
  (Val                :      Enum_Value)
   return Glib.Gint;
```

Return the numeric value for a specific enumeration. Use the matching Ada type and 'Val to convert it to a valid Ada enumeration

```
function Name
  (Val                :      Enum_Value)
   return String;
```

Return the name of Val. This is the equivalent of 'Image in Ada.

```
function Nick
  (Val                 :      Enum_Value)
   return String;
```

Return a displayable string for Val.

```
function Register_Static_Enum
  (Name                :      String;
   Values              :      Interfaces.C.Strings.chars_ptr_array)
   return Glib.GType;
```

Create a new enumeration class from a list of valid values.
Values must be freed by the caller.

```
function Enum_Class_From_Type
  (Typ                 :      Glib.GType)
   return Enum_Class;
```

Return the enumeration class corresponding to a type

13.2.2 Flags classes

These are very similar to Enum Classes. However, the actual value@* of an instance of this type is a combination of a set of flags, rather than one single enumeration value. For instance, a Gdk_Event_Mask is a Flags_Class

```
function Nth_Value
  (Klass               :      Flags_Class;
   Nth                 :      Glib.Guint)
   return Flags_Value;
```

Return the Nth-th value in Klass, or null if there is no such value.

```
function Value
  (Val                 :      Flags_Value)
   return Flags_Int_Value;
```

Return the numeric value for a specific enumeration. Use the matching
Ada type and 'Val to convert it to a valid Ada enumeration

```
function Name
  (Val                 :      Flags_Value)
   return String;
```

Return the name of Val. This is the equivalent of 'Image in Ada.

```
function Nick
  (Val                 :      Flags_Value)
   return String;
```

Return a displayable string for Val.

13.2.3 ParamSpec

```
function Pspec_Name
  (Param               :      Param_Spec)
   return String;
```

Return the name of the property.
This is the internal string representing the property. It Should probably not be displayed on

```
function Nick_Name
  (Param               :      Param_Spec)
   return String;
```

Return the nickname of the property. This is a string
that can be displayed to represent the property, and is more user-friendly than the result of Name.

```
function Flags
  (Param              :      Param_Spec)
   return Param_Flags;
```

Return the flags for the property

```
function Owner_Type
  (Param              :      Param_Spec)
   return Glib.GType;
```

The type that defined Param. If you look for instance at all properties
provides by a type, they will also include properties provided by the parents of the type. This function can be used to find those declared with that type only

```
function Description
  (Param              :      Param_Spec)
   return String;
```

Return the description (ie the help string) for Param

```
function Value_Type
  (Param              :      Param_Spec)
   return Glib.GType;
```

Return the type of param

```
procedure Set_Value_Type
  (Param              :      Param_Spec;
   Typ                :      Glib.GType);
```

Override the type of param. You should only use this function when
creating new Param_Spec types based on existing types. You should not change the type if you haven't created param yourself.

```
function Get_Qdata
  (Param              :      Param_Spec;
   Quark              :      GQuark)
   return Glib.C_Proxy;
```

Return the user data set for Param

```
procedure Set_Qdata
  (Param              :      Param_Spec;
   Quark              :      GQuark;
   Data               :      Glib.C_Proxy;
   Destroy            :      G_Destroy_Notify := null);
```

Associate some named data with Param. Destroy is called when Param is
destroyed.

```
function Minimum
  (Param              :      Param_Spec_Char)
   return Glib.Gint8;

function Maximum
  (Param              :      Param_Spec_Char)
   return Glib.Gint8;

function Default
  (Param              :      Param_Spec_Char)
   return Glib.Gint8;
```

```
function Gnew_Char
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Gint8;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Uchar)
   return Glib.Guint8;
function Maximum
  (Param              :      Param_Spec_Uchar)
   return Glib.Guint8;
function Default
  (Param              :      Param_Spec_Uchar)
   return Glib.Guint8;
function Gnew_Uchar
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Guint8;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Default
  (Param              :      Param_Spec_Boolean)
   return Boolean;
function Gnew_Boolean
  (Name, Nick, Blurb  :      String;
   Default            :      Boolean;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Int)
   return Glib.Gint;
function Maximum
  (Param              :      Param_Spec_Int)
   return Glib.Gint;
function Default
  (Param              :      Param_Spec_Int)
   return Glib.Gint;
function Gnew_Int
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Gint;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Uint)
   return Glib.Guint;
function Maximum
  (Param              :      Param_Spec_Uint)
   return Glib.Guint;
function Default
  (Param              :      Param_Spec_Uint)
   return Glib.Guint;
```

```
function Gnew_Uint
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Guint;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Long)
   return Glib.Glong;
function Maximum
  (Param              :      Param_Spec_Long)
   return Glib.Glong;
function Default
  (Param              :      Param_Spec_Long)
   return Glib.Glong;
function Gnew_Long
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Glong;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Ulong)
   return Glib.Gulong;
function Maximum
  (Param              :      Param_Spec_Ulong)
   return Glib.Gulong;
function Default
  (Param              :      Param_Spec_Ulong)
   return Glib.Gulong;
function Gnew_Ulong
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Gulong;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Default
  (Param              :      Param_Spec_Unichar)
   return Gunichar;
function Gnew_Unichar
  (Name, Nick, Blurb  :      String;
   Default            :      Gunichar;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Enumeration
  (Param              :      Param_Spec_Enum)
   return Enum_Class;
function Default
  (Param              :      Param_Spec_Enum)
   return Glib.Gint;
function Gnew_Enum
  (Name, Nick, Blurb  :      String;
   Enum_Type          :      GType;
   Default            :      Gint := 0;
```

```
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
```

See Glib.Properties.Creation.Register_Static_Enum on how to create Enum_Type

```
function Flags_Enumeration
  (Param              :      Param_Spec_Flags)
   return Flags_Class;
function Default
  (Param              :      Param_Spec_Flags)
   return Glong;
function Gnew_Flags
  (Name, Nick, Blurb  :      String;
   Flags_Type         :      Glib.GType;
   Default            :      Guint;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Float)
   return Gfloat;
function Maximum
  (Param              :      Param_Spec_Float)
   return Gfloat;
function Default
  (Param              :      Param_Spec_Float)
   return Gfloat;
function Epsilon
  (Param              :      Param_Spec_Float)
   return Gfloat;
function Gnew_Float
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Gfloat;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
function Minimum
  (Param              :      Param_Spec_Double)
   return Gdouble;
function Maximum
  (Param              :      Param_Spec_Double)
   return Gdouble;
function Default
  (Param              :      Param_Spec_Double)
   return Gdouble;
function Epsilon
  (Param              :      Param_Spec_Double)
   return Gdouble;
function Gnew_Double
  (Name, Nick, Blurb  :      String;
   Minimum, Maximum, Default  :      Glib.Gdouble;
   Flags              :      Param_Flags
                       := Param_Readable or Param_Writable)
   return Param_Spec;
```

```
function Default
  (Param              :      Param_Spec_String)
   return String;
function Cset_First
  (Param              :      Param_Spec_String)
   return String;
function Cset_Nth
  (Param              :      Param_Spec_String)
   return String;
function Substitutor
  (Param              :      Param_Spec_String)
   return Character;
function Ensure_Non_Null
  (Param              :      Param_Spec_String)
   return Boolean;
function Gnew_String
  (Name, Nick, Blurb  :      String;
   Default            :      String;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
function Gnew_Param
  (Name, Nick, Blurb  :      String;
   Param_Type         :      Glib.GType;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
function Gnew_Boxed
  (Name, Nick, Blurb  :      String;
   Boxed_Type         :      Glib.GType;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
function Gnew_Pointer
  (Name, Nick, Blurb  :      String;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
function Gnew_Object
  (Name, Nick, Blurb  :      String;
   Object_Type        :      Glib.GType;
   Flags              :      Param_Flags
                      := Param_Readable or Param_Writable)
   return Param_Spec;
```

13.2.4 Creating new properties

There are several things that need to be done when creating a property.@* For one thing, you need to create the string that represents the property. This is the only item that needs to go in the specifications of your page. You then need to describe the type of the property, and the values it allows. This is very simple for simple types, and a generic packages is provided to handle the more complex enumeration-based properties.

Your widget needs to define two handlers, Set_Property_Handler and Get_Property_Handler, that are called every time the user accesses the value of a property through a call to Glib.Object.Set_Property or Glib.Object.Get_Property.

For efficiency reasons, a property is also associated with an integer value, that you must provide when creating the property. This value is completely free, and is passed to the two handlers described above.

The two handlers manipulate Glib.Values.GValue values, so that they can get and return various types.

```
procedure Set_Properties_Handlers
  (Class_Record       :      Glib.Object.GObject_Class;
   Set_Property       :      Set_Property_Handler;
   Get_Property       :      Get_Property_Handler);
```

Set the two functions used to set and retrieve properties. You
should never call this function on the class record of the standard gtk+ widgets, since this will break their behavior. You should first create a new class record through Initialize_Class_Record, and then use the returned Class_Record as a parameter to this subprogram.

You cannot pass null to either of the two parameters, or you won't be able to install new properties afterwards

```
procedure Install_Property
  (Class_Record       :      Glib.Object.GObject_Class;
   Prop_Id            :      Property_Id;
   Property_Spec      :      Param_Spec);
```

Adds a new property to Class_Record. You should use this function only
on class records you have created yourself, not on one of the standard widgets. Prop_Id is the internal representation for properties, that will be passed to the Set_Property and Get_Property_Handlers (see above) to set and retrieve the value of a property. Property_Spec should be the result of one of the GNew_* subprograms for Param_Spec, and this defines the type of the property.

14 Package Glib.Type_Conversion_Hooks

This package provides an implementation for hooks used in Gtk.Type_Conversion. These hooks should be used when you import a new C GObject, so that GtkAda can recreate the Ada structure from the underlying C structure. Note that when you create a GObject directly in Ada, you do not need to provide any hook.

Implementation note: This is a separate package from Gtk.Type_Conversion so that adding a hook does not necessarily mean the user has to 'with' Gtk.Type_Conversion, and thus all the packages from GtkAda.

Note that this package is not thread safe. You should call the function Add_Hook from the elaboration part of your packages.

14.1 Types

type Handled_Type **is new** GObject_Record **with private**;

The type we want to convert to.

14.2 Subprograms

```
function Creator
  (Expected_Object    :      GObject_Record'Class)
   return GObject;
```

This function will create an Ada type corresponding to Handled_Type.
In case Expected_Object is a child type of Handled_Type, an Ada object of type Expected_Object is returned instead.

This allows convertion of types we know are expected, but don't have registered conversion hook functions.

15 Package Glib.Types

This package provides an interface to the type system in Glib. These types provide an object-oriented framework (through inheritance and interfaces), as well as reference-counting, signals and properties on these types.

Most of the time, you DO NOT need to use this package, only when you are working with the introspection capabilities of glib.

See the other glib packages for more subprograms to manipulate these types. In particular, Glib.Properties describes the properties system, that provide the base for dynamic introspection. See also Glib itself, which contains several general subprograms, and Glib.Object that provides the root object for any type hierarchy based on glib.

15.1 Types

type GType_Interface **is private**;

15.2 Subprograms

```
function Class_Peek
  (T                  :      GType)
   return Glib.GType_Class;
```

```
function Class_Ref
  (T                  :      GType)
   return Glib.GType_Class;
```

Return the class structure encapsulated in T.
Class_Ref will create the class on-demand if it doesn't exist yet, but Class_Peek might return null if the class hasn't been referenced before. Class_Ref also increments the reference counting for the returned value

```
procedure Class_Unref
  (T                  :      GType);
```

Decrement the reference counting on the associated class. When it reaches 0, the class may be finalized by the type system.

```
function Depth
  (T                  :      GType)
   return Guint;
```

Returns the length of the ancestry of the passed in type. This includes the type itself, so that e.g. a fundamental type has depth 1.

```
function Is_A
  (T                  :      GType;
   Is_A_Type          :      GType)
   return Boolean;
```

If Is_A_Type is a derivable type, check whether type is a descendant of Is_A_Type. If Is_A_Type is an interface, check whether type conforms to it.

15.2.1 Interfaces

Interfaces are similar, in concept, to those found in Ada 2005 or in@* Java. They define a set of subprograms that any type implementing the interface must also define. They are different from standard inheritance since no implementation of these subprograms can be provided in the interface itself.

Whereas an object can only derive from one other object, it can implement any number of interfaces.

Some of the standard gtk+ objects implement interfaces. In this case, their Ada package contains one or more functions to convert from the object itself to the interface, for instance:

package Implements_Cell_Layout is new Glib.Types.Implements (...); function "+" (...) renames Implements_Cell_Layout.To_Interface; function "-" (...) renames Implements_Cell_Layout.To_Object;

The two unary operators "+" and "-" can be used to convert to and from the interface, for instance calling: View : Gtk_Cell_View; Gtk.Cell_Layout.Pack_Start (+View, Cell, Expand);

```
function To_Object
  (Interf             :      GType_Interface)
   return Glib.Object.GObject;
```

Return the object that the interface represents. This is slightly
different from using Implements.To_Object, in the case when the object wasn't created through Ada. In such a case, GtkAda needs to create an Ada wrapper around the object, and will choose a different tagged type:

- Implements.To_Object creates a tagged type of type Object_Type.
- This function creates a GObject, which you cannot cast easily to some other tagged type afterward. Both behave the same when the object was created from Ada.

```
function Interfaces
  (T                  :      GType)
   return GType_Array;
```

Return the list of interfaces implemented by objects of a given type.

```
function Is_Interface
  (T                  :      GType)
   return Boolean;
```

Whether T represents an interface type description

```
function Default_Interface_Peek
  (T                  :      GType)
   return Glib.Object.Interface_Vtable;
function Default_Interface_Ref
  (T                  :      GType)
   return Glib.Object.Interface_Vtable;
```

If the interface type T is currently in use, returns its default
interface vtable. Default_Interface_Ref will create the default vtable for the type if the type is not currently in use. This is useful when you want to make sure that signals and properties for an interface have been installed.

16 Package Glib.Unicode

This package provides functions for handling of unicode characters and utf8 strings. See also Glib.Convert.

16.1 Types

```
type G_Unicode_Type is
          (Unicode_Control,
           Unicode_Format,
           Unicode_Unassigned,
           Unicode_Private_Use,
           Unicode_Surrogate,
           Unicode_Lowercase_Letter,
           Unicode_Modifier_Letter,
           Unicode_Other_Letter,
           Unicode_Titlecase_Letter,
           Unicode_Uppercase_Letter,
           Unicode_Combining_Mark,
           Unicode_Enclosing_Mark,
           Unicode_Non_Spacing_Mark,
           Unicode_Decimal_Number,
           Unicode_Letter_Number,
           Unicode_Other_Number,
           Unicode_Connect_Punctuation,
           Unicode_Dash_Punctuation,
           Unicode_Close_Punctuation,
           Unicode_Final_Punctuation,
           Unicode_Initial_Punctuation,
           Unicode_Other_Punctuation,
           Unicode_Open_Punctuation,
           Unicode_Currency_Symbol,
           Unicode_Modifier_Symbol,
           Unicode_Math_Symbol,
           Unicode_Other_Symbol,
           Unicode_Line_Separator,
           Unicode_Paragraph_Separator,
           Unicode_Space_Separator);
```

The possible character classifications. See http://www.unicode.org/Public/UNIDATA/UCD.html

16.2 Subprograms

```
procedure UTF8_Validate
  (Str                :      UTF8_String;
   Valid              : out   Boolean;
   Invalid_Pos        : out   Natural);
```

Validate a UTF8 string.
Set Valid to True if valid, set Invalid_Pos to first invalid byte.

16.2.1 Character classes

```
function Is_Space
```

```
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is a space character

```
function Is_Alnum
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is an alphabetical or numerical character

```
function Is_Alpha
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is an alphabetical character

```
function Is_Digit
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is a digit

```
function Is_Lower
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is a lower-case character

```
function Is_Upper
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is an upper-case character

```
function Is_Punct
   (Char                :      Gunichar)
    return Boolean;
```

True if Char is a punctuation character

```
function Unichar_Type
   (Char                :      Gunichar)
    return G_Unicode_Type;
```

Return the unicode character type of a given character

16.2.2 Case handling

```
function To_Lower
   (Char                :      Gunichar)
    return Gunichar;
```

Convert Char to lower cases

```
function To_Upper
   (Char                :      Gunichar)
    return Gunichar;
```

Convert Char to upper cases

```
function UTF8_Strdown
   (Str                 :      ICS.chars_ptr;
    Len                 :      Integer)
    return ICS.chars_ptr;
```

```
function UTF8_Strdown
   (Str                 :      UTF8_String)
    return UTF8_String;
```

Convert Str to lower cases

```
function UTF8_Strup
  (Str                :      ICS.chars_ptr;
   Len                :      Integer)
   return ICS.chars_ptr;
function UTF8_Strup
  (Str                :      UTF8_String)
   return UTF8_String;
```

Convert Str to upper cases

16.2.3 Manipulating strings

```
function UTF8_Strlen
  (Str                :      ICS.chars_ptr;
   Max                :      Integer := -1)
   return Glong;
function UTF8_Strlen
  (Str                :      UTF8_String)
   return Glong;
```

Return the number of characters in Str

```
function UTF8_Find_Next_Char
  (Str                :      ICS.chars_ptr;
   Str_End            :      ICS.chars_ptr := ICS.Null_Ptr)
   return ICS.chars_ptr;
function UTF8_Find_Next_Char
  (Str                :      UTF8_String;
   Index              :      Natural)
   return Natural;
function UTF8_Next_Char
  (Str                :      UTF8_String;
   Index              :      Natural)
   return Natural;
function UTF8_Find_Prev_Char
  (Str_Start          :      ICS.chars_ptr;
   Str                :      ICS.chars_ptr)
   return ICS.chars_ptr;
function UTF8_Find_Prev_Char
  (Str                :      UTF8_String;
   Index              :      Natural)
   return Natural;
```

Find the start of the previous UTF8 character after the Index-th byte.
Index doesn't need to be on the start of a character. Index is set to a value smaller than Str'First if there is no previous character.

16.2.4 Conversions

```
function Unichar_To_UTF8
  (C                  :      Gunichar;
   Buffer             :      ICS.chars_ptr := ICS.Null_Ptr)
   return Natural;
procedure Unichar_To_UTF8
  (C                  :      Gunichar;
   Buffer             : out  UTF8_String;
   Last               : out  Natural);
```

Encode C into Buffer. Buffer must have at least 6 bytes free.
Return the index of the last byte written in Buffer.

```
function UTF8_Get_Char
  (Str                 :      UTF8_String)
   return Gunichar;
```

Converts a sequence of bytes encoded as UTF8 to a unicode character.
If Str doesn't point to a valid UTF8 encoded character, the result is undefined.

```
function UTF8_Get_Char_Validated
  (Str                 :      UTF8_String)
   return Gunichar;
```

Same as above. However, if the sequence if an incomplete start of a possibly valid character, it returns -2. If the sequence is invalid, returns -1.

17 Package Glib.Values

This package provides an interface to generic values as used in the Glib object model.

The main type in this package is GValues, which is the equivalent of the C's (GValue*) array, i.e an array of unions. This package provides functions to extract the values from this type.

18 Package Glib.XML

This package provides a simple minded XML parser to be used with Gate.

18.1 Types

```
type Free_Specific_Data is access procedure
          (Data : in out XML_Specific_Data);
```

```
type Node is record
          Tag   : String_Ptr;
            -- The name of this node. This is utf8-encoded

          Attributes   : String_Ptr;
            -- The attributes of this node. This is utf8-encoded

          Value : String_Ptr;
            -- The value, or null is not relevant. This is utf8-encoded

          Parent : Node_Ptr;
            -- The parent of this Node.

          Child : Node_Ptr;
            -- The first Child of this Node. The next child is Child.Next

          Next  : Node_Ptr;
            -- Next sibling node.

          Specific_Data : XML_Specific_Data;
            -- Use to store data specific to each implementation (e.g a boolean
            -- indicating whether this node has been accessed)
          end record;
```

A node of the XML tree. Each time a tag is found in the XML file, a new node is created, that points to its parent, its children and its siblings (nodes at the same level in the tree and with the same parent).

```
type Node_Ptr is access all Node;
```

Pointer to a node of the XML tree.

```
type XML_Specific_Data is private;
```

The type of the extra data that can be attached to each node of the XML tree. See for instance the package Glib.Glade.

18.2 Subprograms

```
function Parse
  (File               :      String)
   return Node_Ptr;
```

Parse File and return the first node representing the XML file.

```
function Parse_Buffer
  (Buffer             :      UTF8_String)
   return Node_Ptr;
```

Parse a given Buffer in memory and return the first node representing the XML contents.

```
procedure Print
  (N                  :      Node_Ptr;
   File_Name          :      String := "");
```

Write the tree starting with N into a file File_Name. The generated file is valid XML, and can be parsed with the Parse function. If File_Name is the empty string, then the tree is printed on the standard output

```
procedure Print
  (N                  :      Node_Ptr;
   File_Name          :      String;
   Success            : out  Boolean);
```

Same as above, with Success reporting the success of the operation.

```
function Protect
  (S                  :      String)
   return String;
```

Return a copy of S modified so that it is a valid XML value

```
function Find_Tag
  (N                  :      Node_Ptr;
   Tag                :      UTF8_String)
   return Node_Ptr;
```

Find a tag Tag in N and its brothers.

```
function Get_Field
  (N                  :      Node_Ptr;
   Field              :      UTF8_String)
   return String_Ptr;
```

Return the value of the field 'Field' if present in the children of N. Return null otherwise. Do not free the returned value.

```
function Is_Equal
  (Node1, Node2       :      Node_Ptr)
   return Boolean;
```

Compare two XML nodes recursively, and returns True if they are equal. Casing in attributes is relevant. Order of attributes is also relevant.

```
procedure Add_Child
  (N                  :      Node_Ptr;
   Child              :      Node_Ptr;
   Append             :      Boolean := False);
```

Add a new child to a node. If Append is true, the child is added at the end of the current list of children.

```
function Deep_Copy
  (N                  :      Node_Ptr)
   return Node_Ptr;
```

Return a deep copy of the tree starting with N. N can then be freed without affecting the copy.

```
procedure Free
  (N                  : in out Node_Ptr;
   Free_Data          :      Free_Specific_Data := null);
```

Free the memory allocated for a node and its children.
It also disconnects N from its parent. If Free_Data is not null, it is used to free the memory occupied by the Specific_Data for each node.

```
function Get_Attribute
  (N                  : in    Node_Ptr;
   Attribute_Name     : in    UTF8_String;
   Default            : in    UTF8_String := "")
   return UTF8_String;
```

Return the value of the attribute 'Attribute_Name' if present.
Special XML characters have already been interpreted in the result string. Return Default otherwise.

```
procedure Set_Attribute
  (N                  :      Node_Ptr;
   Attribute_Name, Attribute_Value  :      UTF8_String);
```

Create a new attribute, or replace an existing one. The attribute value
is automatically protected for special XML characters

```
function Find_Tag_With_Attribute
  (N                  :      Node_Ptr;
   Tag                :      UTF8_String;
   Key                :      UTF8_String;
   Value              :      UTF8_String := "")
   return Node_Ptr;
```

Find a tag Tag in N that has a given key (and value if given).

19 Package Gtkada

This is the parent package for the GtkAda specific units.

20 Package Gtkada.Canvas

This package provides an interactive canvas, on which the user can put items, move them with the mouse, etc. The items can be connected together, and the connections remain active while the items are moved.

It also supports scrolling if put in a Gtk_Scrolled_Window. The canvas will be scrolled (and the selected items moved) if an item is selected and the mouse is dragged on a small area on the side of the canvas or even directly outside of the canvas. Scrolling will continue until the mouse is either released or moved back inside the canvas.

The scrolling speed will slightly increase over time if the mouse is kept outside of the canvas. This makes the canvas much more comfortable to use for the user.

All items put in this canvas must inherit from the type Canvas_Item_Record. However, it is your responsability, as a programmer, to provide drawing routines. In fact, all these items should draw in a pixmap, which is then copied automatically to the screen whenever the canvas needs to redraw itself.

The items can also react to mouse events: mouse clicks are transmitted to the item if the mouse did not move more than a given amount of pixels. To decide what their reaction should be, you should override the On_Button_Click subprogram.

This canvas is not intended for cases where you want to put hundreds of items on the screen. For instance, it does not provide any smart double-buffering other than the one provided by gtk+ itself, and thus you would get some flicker if there are too many items.

There are three coordinate systems used by widget. All the subprograms expect a specific coordinate system as input or output. Here are the three systems: @itemize @bullet @item World coordinates The position of an item is reported in pixels, as if the canvas currently had a zoom level of 100%. This is fully independent, at any time, from the current zoom level of the canvas. Since the canvas is considered to expand ad infinitum, the top-left corner doesn't have any specific fixed coordinates. It can be known by checking the current lower value of the adjustments (aka scrollbars).

@item Canvas coordinates This is similar to world coordinates, except these depend on the current zoom level of the canvas. This also affect the width and height of the objects in the canvas. The subprograms To_Canvas_Coordinates and To_World_Coordinates can be used to convert lengths from world to canvas coordinates. The same behavior as world coordinates applies for the top-left corner. All drawing to the screen, in particular for Draw_Background, must be done using this coordinate systems

@item Item coordinates The position of a point is relative to the top-left corner of the current item. This corner therefore has coordinates (0, 0). This coordinate systems assumes a zoom-level of 100%

@end itemize Items are selected automatically when they are clicked. If Control is pressed at the same time, multiple items can be selected. If the background is clicked (and control is not pressed), then all items are unselected. Pressing and dragging the mouse in the backgroudn draws a virtual box on the screen. All the items fully included in this box when it is released will be selected (this will replace the current selection if Control was not pressed).

20.1 Signals

- **"background_click"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class;
  Event  : Gdk.Event.Gdk_Event);
  ```

  Called every time the user clicks in the background (ie not on an item, or On_Button_Click would be called). This is called both on Button_Release and Button_Press events. The coordinates (X, Y) in the Event are relative to the top-left corner of Canvas.

- **"item_moved"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class;
  Item   : Canvas_Item);
  ```

  Emitted when Item has been moved. New coordinates have been assigned to Item. However, the canvas hasn't been refreshed yet. This signal might be called multiple time when the user finishes a drag action, in case there were several selected items.

- **"item_selected"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class;
  Item   : Canvas_Item);
  ```

  Emitted when the user has clicked on an item to select it, ie before any drag even has occured. This is a good time to add other items to the selection if you need. At thee same time, the primitive operation Selected is called for the item.

- **"item_unselected"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class;
  Item   : Canvas_Item);
  ```

  Emitted when the Item was unselected. At the same time, the primitive operation Selected is called for the item.

- **"set_scroll_adjustments"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class);
  ```

  Emitted when the canvas has scrolled.

- **"zoomed"**

  ```
  procedure Handler (Canvas : access Interactive_Canvas_Record'Class);
  ```

  Emitted when the canvas has been zoomed in or out. You do not need to redraw the items yourself, since this will be handled by calls to Draw

20.2 Types

```
type Arrow_Type is
        (No_Arrow,
           -- the link does not have an arrow

         Start_Arrow,
           -- the link has an arrow at its beginning

         End_Arrow,
           -- the link has an arrow at the end

         Both_Arrow
           -- the link has an arrow on both sides
         );
```

Indicate whether the links have an arrow or not.

type Item_Iterator **is private**;

type Item_Processor **is access function**
(Canvas : **access** Interactive_Canvas_Record'Class;

type Item_Side **is**
(East, West, North, South);

Each side of an item, along its rectangle bounding box

type Layout_Algorithm **is access procedure**

type Link_Processor **is access function**
(Canvas : **access** Interactive_Canvas_Record'Class;

20.3 Subprograms

20.3.1 Creating a canvas

```
procedure Gtk_New
  (Canvas             : out   Interactive_Canvas;
   Auto_Layout        :       Boolean := True);
```

Create a new empty Canvas.
If Auto_Layout is True, then the items are automatically positioned as they are put in the canvas, if no coordinates are specified.

```
procedure Configure
  (Canvas             : access Interactive_Canvas_Record;
   Grid_Size          :       Glib.Guint
                       := Default_Grid_Size;
   Annotation_Font    :       Pango.Font.Pango_Font_Description
                       := Pango.Font.From_String (Default_Annotation_Font);
function Get_Vadj
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the vertical adjustment associated with Canvas

```
function Get_Hadj
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the horizontal adjustment associated with Canvas

```
procedure Draw_Area
  (Canvas             : access Interactive_Canvas_Record'Class;
   Rect               :      Gdk.Rectangle.Gdk_Rectangle);
```

Draw in Canvas the specified area.

```
procedure Draw_Background
  (Canvas             : access Interactive_Canvas_Record;
   Screen_Rect        :      Gdk.Rectangle.Gdk_Rectangle);
```

Draw the background of the canvas. This procedure should be overriden if you want to draw something else on the background. It must first clear the area on the screen.

Screen_Rect is the rectangle on the screen that needs to be refreshed. These are canvas coordinates, therefore you must take into account the current zoom level while drawing.

The default implementation draws a grid.

An example implementation that draws a background image is shown at the end of this file.

```
procedure Draw_Grid
  (Canvas             : access Interactive_Canvas_Record;
   GC                 :      Gdk.GC.Gdk_GC;
   Screen_Rect        :      Gdk.Rectangle.Gdk_Rectangle);
```

Helper function that can be called from Draw_Background. It cannot be used directly as Draw_Background, since it doesn't clear the area first.

```
procedure Set_Orthogonal_Links
  (Canvas             : access Interactive_Canvas_Record;
   Orthogonal         :      Boolean);
```

If Orthogonal is True, then all the links will be drawn only with vertical and horizontal lines. This is not applied for the second or more link between two items.

```
function Get_Orthogonal_Links
  (Canvas             : access Interactive_Canvas_Record)
   return Boolean;
```

Return True if the links are only drawn horizontally and vertically.

```
procedure Align_On_Grid
  (Canvas             : access Interactive_Canvas_Record;
   Align              :      Boolean := True);
```

Choose whether the items should be aligned on the grid when moved. Existing items are not moved even if you set this parameter to True, this will only take effect the next time the items are moved.

```
function Get_Align_On_Grid
  (Canvas             : access Interactive_Canvas_Record)
   return Boolean;
```

Return True if items are currently aligned on grid.

```
procedure Move_To
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class;
   X, Y               :      Glib.Gint := Glib.Gint'First);
```

Move the item in the canvas, to world coordinates (X, Y). Item is assumed to be already in the canvas. If you leave both coordinates X and Y to their

default value, then the item's location will be automatically computed when you layout the canvas (it is your responsability to call Layout).

```
procedure Set_Items
  (Canvas             : access Interactive_Canvas_Record;
   Items              :      Glib.Graphs.Graph);
```

Set the items and links to display in the canvas from Items.
All items previously in the canvas are removed, and replaced by the vertices in Items. Note that the vertices in Items must be in Canvas_Item_Record'Class, and the links must be in Canvas_Link_Record'Class. If you do not have an automatic layout set up in Canvas, you need to set the coordinates of all the vertices by calling Move_To separately.

You mustn't destroy items yourself, this is done automatically when the canvas is destroyed.

```
procedure Put
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class;
   X, Y               :      Glib.Gint := Glib.Gint'First);
```

Add a new item to the canvas, at world coordinates (X, Y).
The item is added at a specific location. If you leave both X and Y to their default value, the item's location will be computed automatically when you call Layout on the canvas, unless Auto_Layout has been set, in which case the position will be computed immediately.

```
function Item_At_Coordinates
  (Canvas             : access Interactive_Canvas_Record;
   X, Y               :      Glib.Gint)
   return Canvas_Item;
```

Return the item at world coordinates (X, Y) which is on top of all
others. null is returned if there is no such item.

```
function Item_At_Coordinates
  (Canvas             : access Interactive_Canvas_Record;
   Event              :      Gdk.Event.Gdk_Event)
   return Canvas_Item;
```

Same as above, but using the canvas coordinates of the event, taking
into account the current zoom level and current scrolling

```
procedure Clear
  (Canvas             : access Interactive_Canvas_Record);
```

Remove all items from the canvas

```
procedure Remove
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Remove an item and all the links to and from it from the canvas.
The item itself is not freed, but the links are. Nothing is done if the item is not part of the canvas.

```
procedure Item_Updated
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

This should be called when Item has changed the contents of its
pixmap, and thus the Canvas should be updated.

```
procedure Refresh_Canvas
  (Canvas             : access Interactive_Canvas_Record);
```

Redraw the whole canvas (both in the double buffer and on the screen).

```
procedure Raise_Item
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Raise the item so that it is displayed on top of all the others
The canvas is refreshed as needed to reflect the change. Nothing happens if Item is not part of the canvas.

```
procedure Lower_Item
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Lower the item so that it is displayed below all the others.
The canvas is refreshed as needed to reflect the change. Nothing happens if Item is not part of the canvas.

```
function Is_On_Top
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class)
   return Boolean;
```

Return True if Item is displayed on top of all the others in the canvas.

```
procedure Show_Item
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Scroll the canvas so that Item is visible. Nothing is done if the item
is already visible

```
procedure Align_Item
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class;
   X_Align            :      Float := 0.5;
   Y_Align            :      Float := 0.5);
```

Scroll the canvas so that the Item appears at the given location in the
canvas. If X_Align is 0.0, the item is align on the left. With 0.5, it is centered horizontally. If 1.0, it is aligned on the right.

```
function Get_Arrow_Angle
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Float;
```

Return the angle of arrows in the canvas.

```
function Get_Arrow_Length
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Glib.Gint;
```

Return the length of arrows in the canvas.

20.3.2 Iterating over items

```
procedure For_Each_Item
  (Canvas             : access Interactive_Canvas_Record;
   Execute            :      Item_Processor;
   Linked_From_Or_To  :      Canvas_Item := null);
```

Execute an action on each of the items contained in the canvas.
If Execute returns False, we stop traversing the list of children. It is safe to remove the items in Item_Processor.

If Linked_From_Or_To is not null, then only the items linked to this one will be processed. It is possible that a given item will be returned twice, if it is both linked to and from the item.

```
function Start
  (Canvas             : access Interactive_Canvas_Record;
   Linked_From_Or_To  :      Canvas_Item := null;
   Selected_Only      :      Boolean := False)
   return Item_Iterator;
```

Return the first item in the canvas.
The same restriction as above applies if Linked_From_Or_To is not null.

```
procedure Next
  (Iter               : in out Item_Iterator);
function Next
  (Iter               :      Item_Iterator)
   return Item_Iterator;
```

Move the iterator to the next item.
All items will eventually be returned if you do not add new items during the iteration and none are removed. However, it is safe to remove items at any time, except the current item

```
function Get
  (Iter               :      Item_Iterator)
   return Canvas_Item;
```

Return the item pointed to by the iterator.
null is returned when there are no more item in the canvas.

```
function Is_Linked_From
  (Iter               :      Item_Iterator)
   return Boolean;
```

Return True if there is a link from:
Get (Iter) -> Linked_From_Or_To Linked_From_Or_To is the item passed to Start. False is returned if this item was null.

20.3.3 Zooming

```
procedure Zoom
  (Canvas             : access Interactive_Canvas_Record;
   Percent            :      Glib.Guint := 100;
   Steps              :      Glib.Guint := 1);
```

Zoom in or out in the canvas.

Steps is the number of successive zooms that will be done to provide smooth scrolling.

Note that one possible use for this function is to refresh the canvas and emit the "zoomed" signal, which might redraw all the items. This can be accomplished by keeping the default 100 value for Percent.

```
function Get_Zoom
  (Canvas             : access Interactive_Canvas_Record)
   return Glib.Guint;
```

Return the current zoom level

```
function To_Canvas_Coordinates
  (Canvas             : access Interactive_Canvas_Record'Class;
   X                  :      Glib.Gint)
   return Glib.Gint;
```

Scale the scalar X depending on the zoom level (map from world
lengths to canvas lengths). Substract the coordinates of the top-left corner if you are converting coordinates instead of lengths.

```
function Top_World_Coordinates
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Glib.Gint;
```

Return the world coordinates for the y=0 canvas coordinates (ie for the upper-left corner of the screen).

```
function Left_World_Coordinates
  (Canvas             : access Interactive_Canvas_Record'Class)
   return Glib.Gint;
```

Return the world coordinates for the x=0 canvas coordinates (ie for the upper-left corner of the screen).

```
function To_World_Coordinates
  (Canvas             : access Interactive_Canvas_Record'Class;
   X                  :      Glib.Gint)
   return Glib.Gint;
```

Scale the scalar X depending by the zoom level (map from canvas coordinates to world coordinates)

```
procedure Get_World_Coordinates
  (Canvas             : access Interactive_Canvas_Record'Class;
   X, Y               : out   Glib.Gint;
   Width              : out   Glib.Gint;
   Height             : out   Glib.Gint);
```

Return the world coordinates of Canvas.

20.3.4 Layout of items

```
procedure Set_Layout_Algorithm
  (Canvas             : access Interactive_Canvas_Record;
   Algorithm          :      Layout_Algorithm);
```

Set the layout algorithm to use to compute the position of the items. Algorithm mustn't be null.

```
procedure Default_Layout_Algorithm
  (Canvas             : access Interactive_Canvas_Record'Class;
   Graph              :      Glib.Graphs.Graph;
   Force              :      Boolean;
   Vertical_Layout    :      Boolean);
```

The default algorithm used in the canvas.
Basically, items are put next to each other, unless there is a link between two items. In that case, the second item is put below the first, as space allows.

```
procedure Set_Auto_Layout
  (Canvas             : access Interactive_Canvas_Record;
   Auto_Layout        :      Boolean);
```

If Auto_Layout is true, then every time an item is inserted in the canvas, the layout algorithm is called. If set to False, it is the responsability of the caller to call Layout below to force a recomputation of the layout, preferably after inserting a number of items.

```
procedure Set_Layout_Orientation
  (Canvas             : access Interactive_Canvas_Record;
   Vertical_Layout    :      Boolean := False);
```

Specify the layout orientation to use for this canvas. The setting is passed as a parameter to the layout algorithm

```
procedure Layout
  (Canvas             : access Interactive_Canvas_Record;
   Force              :      Boolean := False);
```

Recompute the layout of the canvas.
Force can be used to control the layout algorithm, as described above for Layout_Algorithm.

20.3.5 Links

```
procedure Configure
  (Link               : access Canvas_Link_Record;
   Arrow              : in    Arrow_Type := End_Arrow;
   Descr              : in    Glib.UTF8_String := "");
```

Configure a link.
The link is an oriented bound between two items on the canvas. If Descr is not the empty string, it will be displayed in the middle of the link, and should indicate what the link means. Arrow indicates whether some arrows should be printed as well.

```
function Get_Descr
  (Link               : access Canvas_Link_Record)
   return Glib.UTF8_String;
```

Return the description for the link, or "" if there is none

```
function Get_Arrow_Type
  (Link               : access Canvas_Link_Record)
   return Arrow_Type;
```

Return the location of the arrows on Link

```
procedure Set_Src_Pos
  (Link               : access Canvas_Link_Record;
   X_Pos, Y_Pos       :      Glib.Gfloat := 0.5);
```

Set the position of the link's attachment in its source item.
X_Pos and Y_Pos should be given between 0.0 and 1.0 (from left to right or top to bottom).. By default, all links are considered to be attached to the center of items. However, in some cases it is more convenient to attach it to a specific part of the item. For instance, you can force a link to always start from the top of the item by setting Y_Pos to 0.0.

```
procedure Set_Dest_Pos
  (Link               : access Canvas_Link_Record;
   X_Pos, Y_Pos       :      Glib.Gfloat := 0.5);
```

Same as Set_Src_Pos for the destination item

```
procedure Get_Src_Pos
  (Link               : access Canvas_Link_Record;
   X, Y               : out   Glib.Gfloat);
```

Return the attachment position of the link along its source item

```
procedure Get_Dest_Pos
  (Link               : access Canvas_Link_Record;
   X, Y               : out   Glib.Gfloat);
```

Return the attachment position of the link along its destination item

```
function Has_Link
  (Canvas             : access Interactive_Canvas_Record;
   From, To           : access Canvas_Item_Record'Class;
   Name               :      Glib.UTF8_String := "")
   return Boolean;
```

Test whether there is a link from From to To, with the same name.
If Name is the empty string "", then no check is done on the name, and True if returned if there is any link between the two items.

```
procedure Add_Link
  (Canvas             : access Interactive_Canvas_Record;
   Link               : access Canvas_Link_Record'Class;
   Src                : access Canvas_Item_Record'Class;
   Dest               : access Canvas_Item_Record'Class;
   Arrow              : in    Arrow_Type := End_Arrow;
   Descr              : in    Glib.UTF8_String := "");
```

Add Link in the canvas. This connects the two items Src and Dest.
Simpler procedure to add a standard link. This takes care of memory allocation, as well as adding the link to the canvas.

```
procedure Remove_Link
  (Canvas             : access Interactive_Canvas_Record;
   Link               : access Canvas_Link_Record'Class);
```

Remove a link from the canvas.
It also destroys the link itself, and free the memory allocated to it. Nothing is done if Link does not belong to canvas.

```
procedure For_Each_Link
  (Canvas             : access Interactive_Canvas_Record;
   Execute            :      Link_Processor;
   From, To           :      Canvas_Item := null);
```

Execute an action on each of the links contained in the canvas.
If Execute returns False, we stop traversing the list of links. It is safe to remove the link from the list in Link_Processor.

(From, To) can be used to limit what links are looked for.

??? Would be nicer to give direct access to the Graph iterators

```
procedure Destroy
  (Link               : in out Canvas_Link_Record);
```

Method called every time a link is destroyed. You should override this
if you define your own link types. Note that the link might already have been removed from the canvas when this subprogram is called. This shouldn't free the link itself, only its fields.

20.3.6 Drawing links

Drawing of links can be controlled at several levels:@* @itemize @bullet @item Redefining Update_Links gives control at the canvas level. This can be used to implement routing algorithms for the links where the routes must be computed before any link is actually drawn (otherwise it is better to redefine Draw_Link). It can also be used to control in what order the links should be drawn. @item Redefining Draw_Link gives the opportunity to draw links any way you need (several bends, ...). It can be used to control the routing of this specific link, for routing algorithms that only rely on the items layout and not on other links. Otherwise see Update_Links. @item Redefining Draw_Straight_Line if slightly lower-level. This is called by the default Draw_Link procedure, once the ends of the links have been computed. @end itemize

```
procedure Update_Links
  (Canvas             : access Interactive_Canvas_Record;
   GC                 :      Gdk.GC.Gdk_GC;
```

```
   Invert_Mode        :      Boolean;
   From_Selection     :      Boolean);
```

Redraw all the links in the canvas, after the items have been laid out.
GC is a default graphic context that can be used for drawing. However, any other graphic context will do. If Invert_Mode is true, this graphic context must draw in xor mode. If From_Selection is true, then only the links to or from one of the selected items need to be drawn.

```
procedure Draw_Link
  (Canvas             : access Interactive_Canvas_Record'Class;
   Link               : access Canvas_Link_Record;
   Invert_Mode        :      Boolean;
   GC                 :      Gdk.GC.Gdk_GC;
   Edge_Number        :      Glib.Gint;
   Show_Annotation    :      Boolean := True);
```

Redraw the link on the canvas.
Note that this is a primitive procedure of Link, not of Canvas, and thus can easily be overrided for specific links. The default version draws either straight or arc links (the latter when there are multiple links between two given items). This function shouldn't be called if one of the two ends of the link is invisible.

The link should be drawn directly in Get_Window (Canvas).

GC is a possible graphic context that could be used to draw the link. You shouldn't destroy it or modify its attributes. However, you can use any other graphic context specific to your application, for instance if you want to draw the link in various colors or shapes. The graphic context you use must be in Invert mode (see Gdk.GC.Set_Function) if and only if Invert_Mode is true, so that when items are moved on the canvas, the links properly follow the items they are attached to. This graphic context is only used to draw links, so you don't need to restore it on exit if your Draw_Link function always sets it at the beginning.

Edge_Number indicates the index of link in the list of links that join the same source to the same destination. It should be used so that two links do not overlap (for instance, the default is to draw the first link straight, and the others as arcs).

```
procedure Clip_Line
  (Src                : access Canvas_Item_Record;
   Canvas             : access Interactive_Canvas_Record'Class;
   To_X               :      Glib.Gint;
   To_Y               :      Glib.Gint;
   X_Pos              :      Glib.Gfloat;
   Y_Pos              :      Glib.Gfloat;
   Side               : out  Item_Side;
   X_Out              : out  Glib.Gint;
   Y_Out              : out  Glib.Gint);
```

Clip the line that goes from Src at pos (X_Pos, Y_Pos) to (To_X, To_Y)
in world coordinates. The intersection between that line and the border of Rect is returned in (X_Out, Y_Out). The result should be in world coordinates. X_Pos and Y_Pos have the same meaning as Src_X_Pos and Src_Y_Pos in the link record. This procedure is called when computing the position for the links within the default Draw_Link procedure. The default implementation only works with rectangular items. The computed coordinates are then passed on directly to Draw_Straight_Line.

```
procedure Draw_Straight_Line
  (Link               : access Canvas_Link_Record;
   Window             :      Gdk.Window.Gdk_Window;
   GC                 :      Gdk.GC.Gdk_GC;
   Src_Side           :      Item_Side;
   X1, Y1             :      Glib.Gint;
   Dest_Side          :      Item_Side;
   X2, Y2             :      Glib.Gint);
```

Draw a straight link between two points. This could be overriden if you need to draw an something along the link. The links goes from (Src, X1, Y1) to (Dest, X2, Y2), in canvas coordinates. The coordinates have already been clipped so that they do not override the item.

20.3.7 Selection

```
procedure Clear_Selection
  (Canvas             : access Interactive_Canvas_Record);
```

Clear the list of currently selected items.

```
procedure Add_To_Selection
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Add Item to the selection. This is only meaningful during a drag operation (ie during a button press and the matching button release). Item will be moved at the same time that the selection is moved. Item is not added again if it is already in the selection. This function can be called from the Button_Click subprogram to force moving items. This emits the "item_selected" signal.

```
procedure Remove_From_Selection
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class);
```

Remove Item from the selection.
This emits the "item_unselected" signal.

```
procedure Select_All
  (Canvas             : access Interactive_Canvas_Record);
```

Select all the Item in the canvas.

```
function Is_Selected
  (Canvas             : access Interactive_Canvas_Record;
   Item               : access Canvas_Item_Record'Class)
   return Boolean;
```

Return True if the item is currently selected

20.3.8 Items manipulation

```
procedure Selected
  (Item               : access Canvas_Item_Record;
   Canvas             : access Interactive_Canvas_Record'Class;
   Is_Selected        :      Boolean);
```

Called when the item is selected or unselected.
The default is to do nothing.

```
function Point_In_Item
  (Item               : access Canvas_Item_Record;
   X, Y               :      Glib.Gint)
   return Boolean;
```

This function should return True if (X, Y) is inside the item. X and Y
are in world coordinates. This function is meant to be overriden for non-rectangular items, since the default behavior works for rectangular items. This function is never called for invisible items

```
procedure Set_Screen_Size
  (Item                : access Canvas_Item_Record;
   Width               :      Glib.Gint;
   Height              :      Glib.Gint);
```

Set the size of bounding box for the item in world coordinates.
The item itself needn't occupy the whole area of this bounding box, see Point_In_Item. You need to redraw the item, and call Item_Updated to force the canvas to refresh the screen.

```
procedure Draw
  (Item                : access Canvas_Item_Record;
   Canvas              : access Interactive_Canvas_Record'Class;
   GC                  :      Gdk.GC.Gdk_GC;
   Xdest, Ydest        :      Glib.Gint);
```

This subprogram, that must be overridden, should draw the item on
Get_Pixmap (Canvas), at the specific location (Xdest, Ydest). The item must also be drawn at the appropriate zoom level. To do so, all lengths must be multiplied by the current zoom level. If you need to change the contents of the item, you should call Item_Updated after having done the drawing.

```
procedure Destroy
  (Item                : in out Canvas_Item_Record);
```

Free the memory occupied by the item (not the item itself). You should
override this function if you define your own widget type, but always call the parent's Destroy subprogram.

```
procedure On_Button_Click
  (Item                : access Canvas_Item_Record;
   Event               :      Gdk.Event.Gdk_Event_Button);
```

Function called whenever the item was clicked on.
Note that this function is not called when the item is moved, and thus is only called when the click was short. The coordinates (X, Y) in the Event are relative to the top-left corner of Item.

```
function Get_Coord
  (Item                : access Canvas_Item_Record)
   return Gdk.Rectangle.Gdk_Rectangle;
```

Return the coordinates and size of the bounding box for item, in world
coordinates. If the item has never been resized, it initially has a width and height of 1.

```
procedure Set_Visibility
  (Item                : access Canvas_Item_Record;
   Visible             :      Boolean);
```

Set the visibility status of the item. An invisible item will not be
visible on the screen, and will not take part in the computation of the the scrollbars for the canvas. The canvas is not refreshed (this is your responsibility to do it after you have finished doing all the modifications).

```
function Is_Visible
  (Item                : access Canvas_Item_Record)
   return Boolean;
```

Return True if the item is currently visible

```
function Is_From_Auto_Layout
  (Item               : access Canvas_Item_Record)
   return Boolean;
```

Return True if the current location of the item is the result from the auto layout algorithm. False is returned if the item was moved manually by the user.

20.3.9 Buffered items

```
function Pixmap
  (Item               : access Buffered_Item_Record)
   return Gdk.Pixmap.Gdk_Pixmap;
```

Return the double-buffer.
All the drawing on this pixmap must be done at zoom level 100%.

20.3.10 Signals

```
procedure Set_Screen_Size
  (Item               : access Buffered_Item_Record;
   Width, Height      :      Glib.Gint);
```

See documentation from inherited subprogram

```
procedure Draw
  (Item               : access Buffered_Item_Record;
   Canvas             : access Interactive_Canvas_Record'Class;
   GC                 :      Gdk.GC.Gdk_GC;
   Xdest, Ydest       :      Glib.Gint);
```

Draw the item's double-buffer onto Dest.

```
procedure Destroy
  (Item               : in out Buffered_Item_Record);
```

Free the double-buffer allocated for the item

20.4 Example

```
--  The following example shows a possible Draw_Background procedure,
--  that draws a background image on the canvas's background. It fully
--  handles zooming and tiling of the image. Note that drawing a large
--  image will dramatically slow down the performances.

Background : Gdk.Pixbuf.Gdk_Pixbuf := ...;

procedure Draw_Background
  (Canvas         : access Image_Canvas_Record;
   Screen_Rect    : Gdk.Rectangle.Gdk_Rectangle)
is
   X_Left : constant Glib.Gint := Left_World_Coordinates (Canvas);
   Y_Top  : constant Glib.Gint := Top_World_Coordinates (Canvas);
   X, Y, W, H, Ys : Gint;
   Xs : Gint := Screen_Rect.X;
   Bw : constant Gint := Get_Width (Background)
     * Gint (Get_Zoom (Canvas)) / 100;
   Bh : constant Gint := Get_Height (Background)
```

```
      * Gint (Get_Zoom (Canvas)) / 100;
   Scaled : Gdk_Pixbuf := Background;
begin
   if Get_Zoom (Canvas) /= 100 then
      Scaled := Scale_Simple (Background, Bw, Bh);
   end if;

   while Xs < Screen_Rect.X + Screen_Rect.Width loop
      Ys := Screen_Rect.Y;
      X := (X_Left + Xs) mod Bw;
      W := Gint'Min (Screen_Rect.Width + Screen_Rect.X- Xs, Bw - X);

      while Ys < Screen_Rect.Y + Screen_Rect.Height loop
         Y := (Y_Top  + Ys) mod Bh;
         H := Gint'Min
           (Screen_Rect.Height + Screen_Rect.Y - Ys, Bh - Y);
         Render_To_Drawable
           (Pixbuf       => Scaled,
            Drawable     => Get_Window (Canvas),
            Gc           => Get_Black_GC (Get_Style (Canvas)),
            Src_X        => X,
            Src_Y        => Y,
            Dest_X       => Xs,
            Dest_Y       => Ys,
            Width        => W,
            Height       => H);
         Ys := Ys + H;
      end loop;
      Xs := Xs + W;
   end loop;

   if Get_Zoom (Canvas) /= 100 then
      Unref (Scaled);
   end if;
end Draw_Background;
```

21 Package Gtkada.Dialogs

This package provides a ready to use high level dialog capability.

21.1 Types

type Button_Range **is range** 0 .. 8;

The range of valid buttons.

type Message_Dialog_Buttons **is mod** 2 ** 32;

Define the set of values a button in a message dialog box can have.

```
type Message_Dialog_Type is
          (Warning,
              -- Message box with a yellow exclamation point.

           Error,
              -- Message box with a red stop sign.

           Information,
              -- Message box with a blue "i".

           Confirmation,
              -- Message box with a blue question mark.

           Custom
              -- Message box with no pixmap. The caption of the box should be set by
              -- the user.
           );
```

Define the values describing the type of message box. Used by the Message_Dialog function.

21.2 Subprograms

```
function Message_Dialog
  (Msg                :      Glib.UTF8_String;
   Dialog_Type        :      Message_Dialog_Type
                      := Information;
   Buttons            :      Message_Dialog_Buttons
                      := Button_OK or Button_Help;
   Default_Button     :      Message_Dialog_Buttons
                      := Button_OK;
   Help_Msg           :      Glib.UTF8_String := "";
   Title              :      Glib.UTF8_String := "";
   Justification      :      Gtk_Justification
                      := Justify_Center;
   Parent             :      Gtk.Window.Gtk_Window := null)
   return Message_Dialog_Buttons;
```

Display a message dialog box centered on the mouse.
This will create a dialog box containing the specified message. Dialog_Type indicates the purpose of the dialog. Buttons indicates which buttons should appear in the dialog. Help_Msg is the message displayed in a separate dialog box when the help button is pressed

while the dialog is displayed. If Help_Msg is null, a dialog containing the message "No help available" will be displayed. In both cases, the dialog displayed will only have a OK button. If Title is null, a default title will be chosen depending on the value of Dialog_Type. The dialog will be centered with regards to Parent

This function will return only after the user pressed one of the buttons or deleted the dialog, by running an additional level of main loop. One of the following values will be returned:

- Button_None
- Button_Abort
- Button_Yes
- Button_Ok
- Button_Retry
- Button_No
- Button_Cancel
- Button_Ignore
- Button_All

```
function Create_Gtk_Dialog
  (Msg                :      Glib.UTF8_String;
   Dialog_Type        :      Message_Dialog_Type
                      := Information;
   Title              :      Glib.UTF8_String := "";
   Justification      :      Gtk_Justification
                      := Justify_Center;
   Parent             :      Gtk.Window.Gtk_Window := null)
   return Gtk.Dialog.Gtk_Dialog;
```

Convenience function to create a new dialog.
This function was introduced in GtkAda 2.0 to provide a compatibility with Message_Dialog, while using the standard Gtk.Dialog. You should add the buttons yourself, through Gtk.Dialog.Gtk_Dialog, and then display the dialog on the screen through Gtk.Dialog.Run. As opposed to Message_Dialog, you can provide your own custom buttons if needed.

22 Package Gtkada.File_Selection

This package provides a high level support for creating file selection dialogs by handling the signals internally.

22.1 Subprograms

```
function File_Selection_Dialog
  (Title              :      Glib.UTF8_String
                      := "Select File";
   Default_Dir        :      String := "";
   Dir_Only           :      Boolean := False;
   Must_Exist         :      Boolean := False)
   return String;
```

Open a file selection dialog and make it modal.
Return when either the Cancel button is clicked or when a file is selected. Default_Dir is the directory to display in dialog initially. Note that it must end with a directory separator ('/' or '\', depending on your system). You can use GNAT.Os_Lib.Directory_Separator to get the correct value for your system. If Must_Exist is True, then the file (or directory if Dir_Only is True) must exist. If Dir_Only is True, then the dialog is modified so that the user can only choose a directory name, but not a file name. The value returned is the name of the file selected, or "" if none.

23 Package Gtkada.Handlers

This package provides the most commonly used instantiations of Gtk.Handlers
Gate takes advantage of these pre-instantiated packages.

24 Package Gtkada.Intl

This package provides support for string internationalization using the libintl library.

Developer setup ==========

To provide internationalization in your application, you must install a number of files along with your application, and modify your code to highlight the strings to translate. This translation is based on the gettext() library. Reading its documentation is recommanded since it explains best practices for handling translations.

Preparing your code ==========

Gettext needs to information to locate the translation files: a language, as setup by the user (see User Setup below), and a domain, hard-coded in the application. The domain is the name of your application. Given these two informations, the translation file will be found in: $prefix/<lang>/LC_MESSAGES/<domain>.mo

Where $prefix is either one of the standard search paths, or specified through a call to Bind_Text_Domain.

Although the user can simply specify which language to use by setting one environment variable, they are in fact several other setup to be done, so that the C library properly handles date format for instance. This is done through a call to Setlocale.

An application can be associated with several domains, although it is generally recommanded to have one default domain, specify through a call to Text_Domain. Each string can then be translated through a call to Gettext, without specifying the domain every time. A convenient shortcut is provided in the form of the "-" operator.

As a result, typical code would look like: begin Setlocale; Text_Domain ("application"); Bind_Text_Domain ("application", "/usr/local/share/locale"); ... Put_Line (-"Internalized string"); end;

Preparing and installing the translation files ==========

The Gtkada distribution comes with a convenient script named build_skeleton.pl, which you can run on your application to extract all the strings that should be translated. See the "po/" directory in GtkAda, as well as the Makefile in this directory.

Running "make refresh" will reparse all the source files in your application, and create (or update if they already exist) one file .po for each language registered in the Makefile.

You would then translate each of the string indicated my "msgid", by modifying the lines starting with "msgstr".

Once this is done, running the msgfmt tool through "make install" will generate a <lang>.mo binary file, which should be copied in the directory $prefix/<lang>/LC_MESSAGES/<domain>.mo

The translation files can also be created fully by hand. Here is a sample translation file that can be used as an input for msgfmt:

gtkada-fr.po msgid "Help" msgstr "Aide"

msgid "Yes" msgstr "Oui"

$ msgfmt gtkada-fr.po -o gtkada-fr.gmo $ cp gtkada-fr.gmo /usr/share/locale/fr/LC_MESSAGES/gtkada.mo

If your program uses GtkAda, there are also a number of strings that need to be translated in that library. The recommanded approach is to merge the .po files found in the

GtkAda distribution in the "po/" directory, and use the tool msgmerge to merge these into your applications' translation file.

User setup ==========

To change the current locale setting, use the environment variables "LANG". For example, to switch to the french locale using bash:

$ export LANG=fr_FR

Depending on the specific implementation of gettext, the following environment variables may be set to change the default settings of locale parameters:

@itemize @bullet @item LANG Specifies locale name.

@item LC_MESSAGES Specifies messaging locale, and if present overrides LANG for messages.

@item TEXTDOMAIN Specifies the text domain name, which is identical to the message object filename without .mo suffix.

@item TEXTDOMAINDIR Specifies the pathname to the message database, and if present replaces the default (e.g /usr/lib/locale on Solaris, /usr/share/locale on Linux).

@end itemize See the gettext documentation of your specific OS for more details.

24.1 Subprograms

```
function Gettext
  (Msg                 :      Glib.UTF8_String)
   return Glib.UTF8_String;
```

Look up Msg in the current default message catalog.
Use the current locale as specified by LC_MESSAGES. If not found, return Msg itself (the default text).

```
function Dgettext
  (Domain              :      String;
   Msg                 :      Glib.UTF8_String)
   return Glib.UTF8_String;
```

Look up Msg in the Domain message catalog for the current locale.

```
function "-"
  (Msg                 :      Glib.UTF8_String)
   return Glib.UTF8_String;
```

Shortcut for Dgettext ("GtkAda", Msg)

```
function Dcgettext
  (Domain              :      String;
   Msg                 :      Glib.UTF8_String;
   Category            :      Integer)
   return Glib.UTF8_String;
```

Look up Msg in the Domain message catalog for the Category locale.

```
function Default_Text_Domain   return String;
```

Return the current default message catalog.

```
procedure Text_Domain
  (Domain              :      String := "");
```

Set the current default message catalog to Domain.
If Domain is "", reset to the default of "messages".

```
procedure Bind_Text_Domain
  (Domain             :      String;
   Dirname            :      String);
```

Specify that the Domain message catalog will be found in Dirname.
This overrides the default system locale data base. Dirname will generally be the installation prefix for your application.

```
procedure Setlocale;
```

This procedure must be called before any other subprogram in this
package. It will initialize internal variables based on the environment variables.

25 Package Gtkada.MDI

This widget organizes its children into resizable panes. Within each pane, multiple children can be put, and they will be accessible through a notebook.

25.1 Signals

- **"child_added"**

  ```
  procedure Handler
  (MDI : access MDI_Window_Record'Class; Child : System.Address);
  ```

  Emitted when a new child is added. You cannot use the "add" signal since in fact the children are added to notebooks that are part of the MDI, and thus "add" is only emitted when a new notebook is created.

- **"child_icon_changed"**

  ```
  procedure Handler
  (MDI : access MDI_Window_Record'Class; Child : System.Address);
  ```

  Emitted when the icon for Child has changed

- **"child_removed"**

  ```
  procedure Handler
  (MDI : access MDI_Window_Record'Class; Child : System.Address);
  ```

  Emitted when a new child is removed. You cannot use the "remove" signal since in fact the children are removed from notebooks that are part of the MDI, and thus "remove" is only emitted when a new notebook is destroyed. When this signal is emitted, Child no longer contains a widget, and is no longer part of the children, although you can still access its titles.

- **"child_selected"**

  ```
  procedure Handler
  (MDI : access MDI_Window_Record'Class; Child : System.Address);
  ```

  This signal is emitted when a new child has gained the focus. Convert Child to a MDI_Child by calling Gtk.Arguments.To_Object. This can be used to change some global information at the MDI level. You should connect to "selected" (see below) instead if you want to change some information at the child level. Child might be null if no child has the focus anymore

- **"child_title_changed"**

  ```
  procedure Handler
  (MDI : access MDI_Window_Record'Class; Child : System.Address);
  ```

  Emitted when the title of a child is changed. This signal is not emitted if Set_Title is called for a child that hasn't been put in the MDI yet.

- **"children_reorganized"**

  ```
  procedure Handler (MDI : access MDI_Window_Record'Class);
  ```

  Emitted when the children have been reorganized: either a split occurred, or a window was dropped into another position

  The following new signals are defined for the MDI_Child_Record object:

- **"delete_event"**

```
function Handler (Child : access Gtk_Widget_Record'Class)
return Boolean;
```

This signal is emitted for each item in the MDI window before it is actually deleted. The child is destroyed only if the handler returns False. Note that the Child passed in argument is exactly the one you passed to Put to insert it in the MDI window. Note that this is also the signal to use to prevent top level Gtk_Window from being destroyed.

- **"float_child"**

```
procedure Handler (Child : access MDI_Child_Record'Class);
```

Emitted when a child is set as floating

- **"selected"**

```
procedure Handler (Child : access MDI_Child_Record'Class);
```

This is emitted when the child is selected, ie gains the MDI focus. You should probably also connect to the "grab_focus" signal to be informed when the child gets the keyboard focus. This can be used to transfer the focus to some specific part of the widget. Connecting to "grab_focus" should be done with the After parameter set to True.

- **"unfloat_child"**

```
procedure Handler (Child : access MDI_Child_Record'Class);
```

Emitted when a child is put back in the main MDI window

25.2 Types

```
type Child_Flags is mod 2 ** 5;
```

```
type Child_Group is new Positive;
```

```
type Child_Iterator is private;
```

```
type Child_Position is
        (Position_Automatic,
         Position_Bottom,
         Position_Top,
         Position_Left,
         Position_Right);
```

```
type Load_Desktop_Function is access function
        (MDI : MDI_Window; Node : Glib.Xml_Int.Node_Ptr; User : User_Data)
```

type MDI_Child_Array **is array** (Natural **range** <>) **of** MDI_Child;

type Save_Desktop_Function **is access function**
(Widget : **access** Gtk.Widget.Gtk_Widget_Record'Class;

type Show_Tabs_Policy_Enum **is**
(Always, Never, Automatic);

subtype Side_Position **is** Child_Position **range** Position_Bottom .. Position_Right;

The initial position of windows within the MDI. In all cases, the initial location for a window is computed with the following algorithm. This algorithm is designed with the notion of groups of windows in mind, so that some windows (typically editors) have a special status. - If another window with the same Group is already in the MDI, the new window is put on top of it. - Otherwise, if Position_Automatic, if an empty area exists within the MDI, the new window is put in that area. - Else if the Position is Bottom .. Right, the new window is put below all others (resp. to the top, left or right) - Else the window is put on top of the currently selected window

type State_Type **is**
(Normal, Floating);

This type indicates the state of an item in the MDI: - Normal: the item can be manipulated (moved and resized) by the user. It is found either in the middle notebook (maximized items), or in the layout. - Floating: the item has its own toplevel window, and is thus managed by the window manager.

type User_Data **is private**;

Generic type of parameter that is passed to all the children's save and restore functions.

25.3 Subprograms

```
procedure Gtk_New
  (MDI                : out   MDI_Window;
   Group              : access Gtk.Accel_Group.Gtk_Accel_Group_Record'Class);
```

Create a new MDI window.
Note that it is recommended that you modify the style (Set_Background in State_Normal) to have a different color. You should call Setup_Toplevel_Window once you have added the MDI to a toplevel widget, so that focus is correctly handled when the toplevel window gains the focus

```
procedure Setup_Toplevel_Window
  (MDI                : access MDI_Window_Record;
```

```
   Parent             : access Gtk.Window.Gtk_Window_Record'Class);
```

Setup Parent to properly handle focus when the window manager changes
the window that currently has the focus. Parent must be the toplevel window that contains the MDI.

```
procedure Configure
  (MDI                : access MDI_Window_Record;
   Opaque_Resize      :      Boolean := False;
   Close_Floating_Is_Unfloat  :      Boolean := True;
   Title_Font         :      Pango.Font.Pango_Font_Description
                      := null;
   Background_Color   :      Gdk.Color.Gdk_Color
                      := Gdk.Color.Null_Color;
   Title_Bar_Color    :      Gdk.Color.Gdk_Color
                      := Gdk.Color.Null_Color;
   Focus_Title_Color  :      Gdk.Color.Gdk_Color
                      := Gdk.Color.Null_Color;
   Draw_Title_Bars    :      Boolean := True;
   Tabs_Position      :      Gtk.Enums.Gtk_Position_Type
                      := Gtk.Enums.Pos_Bottom;
   Show_Tabs_Policy   :      Show_Tabs_Policy_Enum
                      := Automatic);
```

Change the setup of the MDI.
Close_Floating_Is_Unfloat, if True, means that closing a floating child will put it back in the MDI instead of destroying it (unless its flag Always_Destroy_Float is set). Title_Font is the font used in the title bars (if null, "sans 8" is used). The colors, when Null_Color, will not change the current setup. If Draw_Title_Bars is False, then no extra title bar will be displayed for the MDI children when they are maximized. This saves space on the screen. However, the notebook tabs will be highlighted with Title_Bar_Color in exchange. Tabs_Position indicates where the notebook tabs should be put. Show_Tabs_Policy indicates when the notebook tabs should be displayed.

25.3.1 Windows

```
procedure Gtk_New
  (Child              : out   MDI_Child;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Flags              :      Child_Flags := All_Buttons;
   Group              :      Child_Group := Group_Default;
   Focus_Widget       :      Gtk.Widget.Gtk_Widget := null);
```

Create a new MDI child that contains widget.
Widget mustn't be of type Gtk_Window.

You shouldn't access Widget directly afterwards, but should manipulate Child only. However, as a special exception, you can still pass Widget as a parameter to the subprograms in this package to manipulate it (e.g. in Raise_Child, ...)

Note: You might have to call Set_Size_Request on Widget to set its initial size. This won't prevent it from being resized by the user.

If Focus_Widget is not null, this is the widget that gets the keyboard focus when the child is selected.

```
procedure Put
  (MDI                : access MDI_Window_Record;
   Child              : access MDI_Child_Record'Class;
```

```
   Initial_Position   :      Child_Position
                       := Position_Automatic);
```

Add a new child to the MDI window, and return its embedding widget.
Calling Put does not give the focus to the newly inserted widget. To do that, you should call Set_Focus_Child.

```
procedure Set_Size
  (MDI                : access MDI_Window_Record;
   Child              : access MDI_Child_Record'Class;
   Width              :      Glib.Gint;
   Height             :      Glib.Gint;
   Fixed_Size         :      Boolean := False);
```

Forces a new size for a child. If Width or Height is left to -1, the
matching size will be computed from the child's requisition. If they are left to 0, the corresponding length is left to its current value. If Fixed_Size is True, then the widget will not be resized when the MDI itself is resized (unless the user has first moved one of the handles to manually resize it). Otherwise, it will grow proportionally with the rest of the MDI.

```
procedure Close
  (MDI                : access MDI_Window_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Force              :      Boolean := False);
```

Close the child that contains Child, and remove its window from the
MDI. See also Close_Child if you need to close a MDI_Child itself. This first checks through a delete_event callback whether the child accepts to be closed. "delete_event" is not sent, and the child is automatically closed, if Force is set to True.

```
procedure Set_Title
  (Child              : access MDI_Child_Record;
   Title              :      UTF8_String;
   Short_Title        :      UTF8_String := "");
```

Set the title for a child. Title is the title put in titlebar of
the children, whereas Short_Title is the name of the notebook tab when children are maximized. By default, it is the same as Title.

The default title is the empty string. This title will be the one used for the window when the child is set to floating state.

```
function Get_Title
  (Child              : access MDI_Child_Record)
   return UTF8_String;
```

Return the title for a specific child

```
function Get_Short_Title
  (Child              : access MDI_Child_Record)
   return UTF8_String;
```

Return the name of the notebook tab used when children are maximized.

```
function Get_State
  (Child              : access MDI_Child_Record)
   return State_Type;
```

Return the current state of the child

```
procedure Set_Icon
  (Child              : access MDI_Child_Record;
   Icon               :      Gdk.Pixbuf.Gdk_Pixbuf);
```

Associate an icon with Child. This icon is visible in the title bar, the notebook tabs, the Window menu and the interactive selection dialog. The icon is updated dynamically on the screen.

```
function Get_Icon
  (Child              : access MDI_Child_Record)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Returns the icon associated with Child

25.3.2 Drag and Drop support

```
function Dnd_Data
  (Child              : access MDI_Child_Record;
   Copy               :      Boolean)
   return MDI_Child;
```

When a drag-and-drop operation took place to move a child from one position to the next, this function is called to know what child should be moved. As a result, the implementor can choose whether a copy of the child should be returned (creating a new view for an editor for instance), or if the child itself should be moved (the default). The returned MDI_Child must have been added to the MDI before it is returned. Copy is set to true if a copy operation was requested, to False if a simple move operation was requested. It can be ignored if Child doesn't know how to create a copy of itself for instance.

```
procedure Child_Drag_Begin
  (Child              : access MDI_Child_Record'Class;
   Event              :      Gdk.Event.Gdk_Event);
```

Starts a drag-and-drop operation for the child, so that it can be put in some other place on the desktop. This should only be called when a handler for the "button_press_event" signal, passing the event itself in parameter. The Child is immediately raised and gains the focus.

```
procedure Cancel_Child_Drag
  (Child              : access MDI_Child_Record'Class);
```

Cancel a drag operation started by Child_Drag_Begin. It doesn't call Child_Drag_Finished.

```
procedure Child_Drag_Finished
  (Child              : access MDI_Child_Record);
```

Called when a drag operation is either aborted or completed. It should be overriden if special cleanup should be done.

25.3.3 Menus

```
function Create_Menu
  (MDI                : access MDI_Window_Record;
   Accel_Path_Prefix  :      String := "<gtkada>")
   return Gtk.Menu.Gtk_Menu;
```

Create a dynamic menu that can then be inserted into a menu bar. This menu is dynamic, ie its content will changed based on the focus child. If this function is called several times, the same menu is returned every time. Accel_Path_Prefix must be the same for every call. Accel_Path_Prefix is used so that the key shortcuts associated with these menu items can be changed dynamically by the user (see gtk-accel_map.ads). The prefix must start with "<" and end with ">".

25.3.4 Selecting children

```
procedure Highlight_Child
  (Child              : access MDI_Child_Record;
   Highlight          :      Boolean := True);
```

Highlight the child until it is selected by the user.
The color of its menu label and of the text in the notebook tabs is changed. Nothing is done if the child is already fully visible (either in the active page in one of the notebooks, or the child that has the selection in the layout). This is meant to be used as a graphical note to the user that the child has been updated and the user should look at it.

```
function Get_Focus_Child
  (MDI                : access MDI_Window_Record)
   return MDI_Child;
```

Return the child that currently has the MDI focus.
null is returned if no child has the focus.

```
procedure Set_Focus_Child
  (MDI                : access MDI_Window_Record;
   Containing         : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Give the focus to the child containing Containing. This will not
Grab_Focus for the child in all cases, since you might want to give the focus to some specific part of your widget (an entry field,...) in some cases.

```
procedure Set_Focus_Child
  (Child              : access MDI_Child_Record);
```

Make Child the active widget, and raise it at the top.

```
procedure Check_Interactive_Selection_Dialog
  (MDI                : access MDI_Window_Record;
   Event              :      Gdk.Event.Gdk_Event;
   Move_To_Next       :      Boolean;
   Only_Group         :      Child_Group := Group_Any);
```

Open the interactive dialog for selecting windows.
This dialog should be open as a result of a key press event. Move_To_Next indicates whether we want to select the next child (True) or the previous child (False). This dialog will be closed only when the key that opened it is fully released. For instance, if the dialog was opened as a result of pressing Ctrl-Tab, the dialog will only be closed when Ctrl itself is released. You can call this procedure even if a dialog is currently open. This simply forces a move to the next or previous child. In fact, it is your responsability to call this procedure when the user presses the keys to move between children.

If Event is null, then no dialog is displayed. Instead, the next or previous visible child is immediately selected. In such a mode, windows that are not on top of their respective notebook are ignored. This can be used to emulate Emacs's behavior for goto-other-window.

If Only_Group is specified, then only the windows from that group will be shown in the dialog.

25.3.5 MDI_Child and encapsulated children

```
function Get_Widget
  (Child              : access MDI_Child_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget that Child encapsulates. This is the widget you
initially Put() in MDI.

```
function Find_MDI_Child
  (MDI                : access MDI_Window_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return MDI_Child;
```

Return the MDI_Child that encapsulates Widget.
Widget must be the exact same one you gave in argument to Put.

```
function Find_MDI_Child_From_Widget
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return MDI_Child;
```

Return the MDI child that encapsulate the parent of Widget.
As opposed to Find_MDI_Child, Widget can be anywhere within the widget tree. This function properly handles floating children

```
function Find_MDI_Child_By_Tag
  (MDI                : access MDI_Window_Record;
   Tag                :      Ada.Tags.Tag)
   return MDI_Child;
```

Return the first child matching Tag

```
function Find_MDI_Child_By_Name
  (MDI                : access MDI_Window_Record;
   Name               :      String)
   return MDI_Child;
```

Return the first child matching Name.

```
function First_Child
  (MDI                : access MDI_Window_Record;
   Group_By_Notebook  :      Boolean := False)
   return Child_Iterator;
```

Return an access to the first child of the MDI.

If Group_By_Notebook is True, then the children are reported one after the other, but all the widget from the same notebook are reported in the same order as the notebook pages. Floating children do not belong to a notebook, and are also reported together. To find out to which notebook a child belongs, use Get_Notebook below.

If Group_By_Notebook is False, it is garanteed that the first child is the one that currently has the focus in the MDI. The children are returned in the order in which they last had the focus.

```
procedure Next
  (Iterator           : in out Child_Iterator);
```

Move to the next child in the MDI

```
function Get_Notebook
  (Iterator           :      Child_Iterator)
   return Gtk.Notebook.Gtk_Notebook;
```

Return the notebook to which the current child belongs. null is returned
for floating children

```
function Get
  (Iterator           :      Child_Iterator)
   return MDI_Child;
```

Return the child pointed to by Iterator.
If Iterator is no longer valid, null is returned.

25.3.6 Floating and closing children

```
procedure Float_Child
  (Child              : access MDI_Child_Record'Class;
   Float              :      Boolean);
```

Change the floating state of a child

```
function Is_Floating
  (Child              : access MDI_Child_Record'Class)
   return Boolean;
```

Return True if Child is currently in a separate window

```
procedure Close_Child
  (Child              : access MDI_Child_Record'Class;
   Force              :      Boolean := False);
```

Same as Close, but applies directly to a MDI_Child.

```
procedure Set_All_Floating_Mode
  (MDI                : access MDI_Window_Record;
   All_Floating       :      Boolean);
```

If All_Floating is set to true, the MDI will have a size of 0x0, and all children are set to floating. This can be used if you wish to let the window manager handle the windows. If All_Floating is True, children can no longer be maximized.

```
procedure Use_Short_Titles_For_Floats
  (MDI                : access MDI_Window_Record;
   Short_Titles       :      Boolean);
```

If Short_Titles is set to true, all floating children will use their short titles.

25.3.7 Reorganizing children

```
procedure Raise_Child
  (Child              : access MDI_Child_Record'Class;
   Give_Focus         :      Boolean := True);
```

Put Child in the foreground.
Note that this does not give the focus to this child, unless Give_Focus is set to True. If Child and the current focus child are in the same notebook, Child will always gain the focus, so that the focus is not left on an invisible window.

```
function Is_Raised
  (Child              : access MDI_Child_Record'Class)
   return Boolean;
```

Whether the child is currently raised, ie fully visible to the user

```
procedure Lower_Child
  (Child              : access MDI_Child_Record'Class);
```

Put Child in the background.
If the children are maximized, this selected the next page from the notebook.

```
procedure Split
  (MDI                : access MDI_Window_Record;
   Orientation        :      Gtk.Enums.Gtk_Orientation;
   Reuse_If_Possible  :      Boolean := False;
   After              :      Boolean := False;
   Width, Height      :      Glib.Gint := 0);
```

Split the central area. The split starting from either the currently selected child or the last child that had the focus in that area. If Reuse_If_Possible is True,

and the current child is already splitted in the right directory, we reuse that area. If After is true, then the currently selected child is put below or to the right in the splitted area, otherwise it is left on the top or left of that area). Width and Height indicate the desired geometry for the splitted area, 0 indicate a 50/50 split.

25.3.8 Desktop Handling

The MDI provides a way to save desktops, i.e the list of children@* currently open in the MDI and their location. It can then restore the desktop at some later point.

Desktops require support from the widgets that are put in the MDI. They need to register a function to save them and a function to recreate them. Using Ada streams for this didn't prove workable since some children might need extra parameters not available to them through streams. This is why the following subprograms are in a generic package, so that you can pass whatever parameter(s) is needed in your application.

Desktops are saved and restored in XML trees.

If you need your application to load a "default desktop" when the user hasn't defined one, it is recommended that you distribute an actual file containing this desktop. You could also create the XML tree in memory yourself, and thus hard-code the default desktop if need be.

```
procedure Register_Desktop_Functions
  (Save               :      Save_Desktop_Function;
   Load               :      Load_Desktop_Function);
```

Register a set of functions to save and load desktops for some
specific widget types. Neither Save nor Load can be null.

```
function Restore_Desktop
  (MDI                : access MDI_Window_Record'Class;
   From_Tree          :      Glib.Xml_Int.Node_Ptr;
   User               :      User_Data)
   return Boolean;
```

Restore the contents of the MDI from its saved XML tree.
User is passed as a parameter to all of the Load_Desktop_Function registered by the widgets. Return False if the desktop couldn't be loaded It also restores the size and position of the toplevel window that contains the MDI

```
function Save_Desktop
  (MDI                : access MDI_Window_Record'Class;
   User               :      User_Data)
   return Glib.Xml_Int.Node_Ptr;
```

Return an XML tree that describes the current contents of the MDI.
This function calls each of the registered function for the children of the MDI. It also saves the size and position of the toplevel window that contains the MDI

```
procedure Free_Registered_Desktop_Functions;
```

Free the memory allocated for the registered functions.

```
function Desktop_Was_Loaded
  (MDI                : access MDI_Window_Record)
   return Boolean;
```

Return True if a desktop was loaded, False if the MDI is only the result
of calls to Gtk_New and Put.

26 Package Gtkada.Multi_Paned

This widget implements a multi-paned widget, similar to the standard Gtk_Paned widget, but which can contain several children side to side. This widget can mix vertical and horizontal splits

26.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Fixed        (Package Gtk.Fixed)
            \___ Gtkada_Multi_Paned (Package Gtkada_Multi_Paned)
```

26.2 Types

type Child_Iterator **is private**;

type Pane **is private**;

An area of the window, which can is splitted either horizontally or vertically. It can contain one or several children, next to each other, or on top of one another.

26.3 Subprograms

```
procedure Gtk_New
  (Win                : out   Gtkada_Multi_Paned);
procedure Set_Opaque_Resizing
  (Win                : access Gtkada_Multi_Paned_Record;
   Opaque             :      Boolean);
```

Whether resizing of the widgets should be opaque or not. The default is not to do opaque resizing for efficiency reasons

```
procedure Add_Child
  (Win                : access Gtkada_Multi_Paned_Record;
   New_Child          : access Gtk.Widget.Gtk_Widget_Record'Class;
   Orientation        :      Gtk.Enums.Gtk_Orientation
                      := Gtk.Enums.Orientation_Horizontal;
   Fixed_Size         :      Boolean := False;
   Width, Height      :      Glib.Gint := 0;
   After              :      Boolean := True);
```

Add new child, splitting as needed.
This should be used when there is no child yet The window is splitted in two by default. However, if Width and Height are specified (or left to -1 for automatic computation), the window is splitted so that amount of screen space is left to the widget (leaving some minimum amount of space to other children as needed). If Fixed_Size is true, then the size of the dock will not change when Win is resized. Otherwise, it will keep its relative size (x% of the total size of Win). This Fixed_Size setting will be reset to False as soon as the user has resized a pane with the mouse.

```
procedure Split
  (Win                : access Gtkada_Multi_Paned_Record;
   Ref_Widget         : access Gtk.Widget.Gtk_Widget_Record'Class;
   New_Child          : access Gtk.Widget.Gtk_Widget_Record'Class;
   Orientation        :      Gtk.Enums.Gtk_Orientation;
   Fixed_Size         :      Boolean := False;
   Width, Height      :      Glib.Gint := 0;
   After              :      Boolean := True);
```

Split the pane containing Ref_Widget, and add New_Child
in the new pane (on the right or at the bottom if After is True, on the left or at the top if After is False).

```
procedure Set_Size
  (Win                : access Gtkada_Multi_Paned_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Width, Height      :      Glib.Gint := -1;
   Fixed_Size         :      Boolean := False);
```

Force a specific size for Widget

```
function Splitted_Area
  (Win                : access Gtkada_Multi_Paned_Record;
   Ref_Widget         : access Gtk.Widget.Gtk_Widget_Record'Class;
   Orientation        :      Gtk.Enums.Gtk_Orientation;
   After              :      Boolean := True)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget in the splitted area next to Ref_Widget if any exist.
Orientation and After define which splitted area we are looking at. null is returned if there are no such splitted area.

```
function Get_Pane
  (Win                : access Gtkada_Multi_Paned_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Pane;
function Get_Pane
  (Current_Pane       :      Pane)
   return Pane;
```

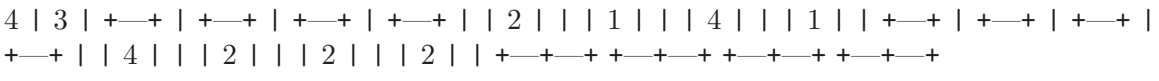
Return the pane that contains the widget. See comment for Split below.

```
procedure Split
  (Win                : access Gtkada_Multi_Paned_Record;
   Ref_Pane           :      Pane;
   New_Child          : access Gtk.Widget.Gtk_Widget_Record'Class;
   Orientation        :      Gtk.Enums.Gtk_Orientation;
   Fixed_Size         :      Boolean := False;
   Width, Height      :      Glib.Gint := 0;
   After              :      Boolean := True);
```

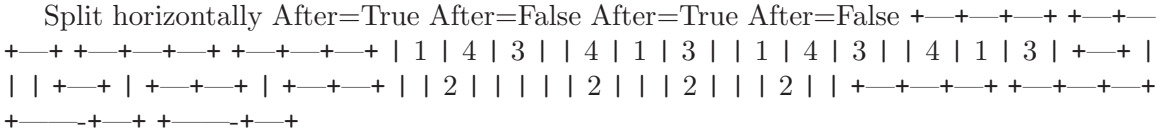
Split Ref_Pane to display New_Child to one of its sides.
See the comments for Root_Pane above. The examples below assume that you are using one of the two split procedures, either with a Ref_Pane or a Ref_Widget. In the former case, the pane is obtained with a call to Get_Pane(Ref_Widget). As you will see, the results are different (although they might appear similar sometimes on this simple example. In all these examples, we split either vertically or horizontally, and add a new widget "4".

Given the following setup: +—+—+ | 1 | | +—+ 3 | | 2 | | +—+—+

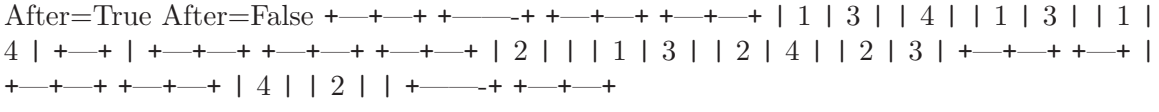
Ref_Pane = Get_Pane ("1") Ref_Widget = "1" Split vertically After=True After=False After=True After=False +—+—+ +—+—+ +—+—+ +—+—+ | 1 | 3 | | 4 | 3 | | 1 | 3 | |



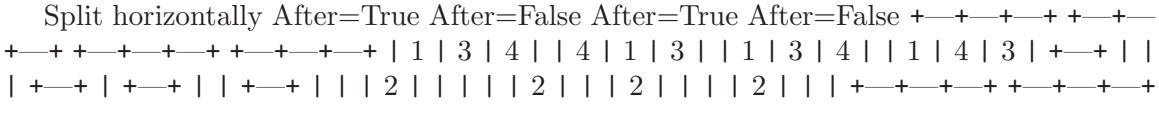
Split horizontally After=True After=False After=True After=False



Ref_Pane = Get_Pane ("3") Ref_Widget = "3" Split vertically After=True After=False After=True After=False



Split horizontally After=True After=False After=True After=False



```
procedure Freeze
  (Win                 : access Gtkada_Multi_Paned_Record);
```

Freeze the window, ie when a child is inserted, no computation of its size is done, and will not generate immediate resizing. You only need to call this procedure when restoring Win to a previously state saved, and never if you are using the GtkAda.MDI which takes care of it on its own.

```
procedure Thaw
  (Win                 : access Gtkada_Multi_Paned_Record);
```

Opposite of Freeze. You should call Size_Allocate on Win afterward to force a recomputation of the size

26.3.1 Iterators

```
function Start
  (Win                 : access Gtkada_Multi_Paned_Record)
   return Child_Iterator;
```

Return an iterator to the first child of the window. This also returns children which are not widget, but are used to organize the window into horizontal and vertical panes

```
function At_End
  (Iter                :      Child_Iterator)
   return Boolean;
```

True if there is no more child to be returned

```
procedure Next
  (Iter                : in out Child_Iterator);
```

Move to the next child of Iterator

```
function Get_Widget
  (Iter                :      Child_Iterator)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget embedded in the current child. This returns null if the current child is only used as a pane separator (horizontal or vertical). You mustn't remove the widget from the paned widget, or the iterator becomes invalid.

```
function Get_Orientation
  (Iter               :      Child_Iterator)
   return Gtk.Enums.Gtk_Orientation;
```

Return the orientation of the current child. This is only relevant if the child doesn't contain a widget (and therefore Get_Widget has returned null).

```
function Get_Depth
  (Iter               :      Child_Iterator)
   return Natural;
```

Return the depth of the current child (0 means the child is at the toplevel, 1 that this is a child directly underneath,...). This can be used to detect when the Iter has finished traversing one of the panes.

```
procedure Dump
  (Split              : access Gtkada_Multi_Paned_Record'Class);
```

Dump the configuration of Split to stdout. This is only intended for testing purposes. If you want to save and restore this configuration, you should look at Gtkada.MDI instead, which contains all the subprograms needed to handle desktops.

27 Package Gtkada.Pixmaps

This package provides a collection of "standard" pixmaps

27.1 Subprograms

```
function "+"
  (Str                : in    String)
   return Gtkada.Types.Chars_Ptr;
```

28 Package Gtkada.Properties

This package provides a dialog, that you can use a development help in your own application. This dialog allows you to select any widget from your application, and see its properties, or even change them dynamically. This helps in analyzing the effect of properties.

28.1 Subprograms

```
procedure Popup_Properties_Editor
  (Object             : access Glib.Object.GObject_Record'Class);
```

Popup a dialog to view and edit the properties of Object. If such a dialog is already displayed for Object, it is made visible.

```
function Widget_At
  (Top                : access Gtk.Widget.Gtk_Widget_Record'Class;
   X, Y               :      Glib.Gint)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget at the given coordinates within Top

```
function Widget_At_Pointer     return Gtk.Widget.Gtk_Widget;
```

Return the widget below the mouse pointer
??? See Gdk.Display.Get_Window_At_Pointer

29 Package Gtkada.Types

This package provides GtkAda specific types and their associated functions.

29.1 Types

subtype Chars_Ptr **is** Interfaces.C.Strings.chars_ptr;

subtype Chars_Ptr_Array **is** Interfaces.C.Strings.chars_ptr_array;

29.2 Subprograms

```
procedure g_free
  (Mem                 :      Chars_Ptr);
```

Free a C string returned from Gtk

```
function Null_Array             return Chars_Ptr_Array;
```

Return a null array.

29.2.1 Handling of arrays of Strings

The following functions provide a very convenient way to create@* C arrays of null terminated strings in Ada.

You can either create such a String on the fly, or declare a variable:

Signals : Chars_Ptr_Array := "clicked" + "missed" + "new signal";

which corresponds to the C declaration:

char *signals[] = @{"clicked", "missed", "new signal"@};

Note that you still need to manually call Free (Signals) if you want to release the memory dynamically allocated by the "+" functions.

```
function "+"
  (S1, S2              :      String)
   return Chars_Ptr_Array;
```

Create an array containing S1 and S2.
Note that this function allocates memory to store S1 and S2 as null terminated Strings. The user is responsible for calling Free on the resulting array.

```
function "+"
  (S1                  :      Chars_Ptr_Array;
   S2                  :      String)
   return Chars_Ptr_Array;
```

Append S2 to S1.
Note that this function allocates memory to store S2 as a null terminated Strings. The user is responsible for calling Free on the resulting array.

```
function "+"
  (S1                  :      Chars_Ptr_Array;
   S2                  :      Chars_Ptr)
   return Chars_Ptr_Array;
```

Append S2 to S1.
Note that this function allocates memory to store S2 as a null terminated Strings. The user is responsible for calling Free on the resulting array.

```
function "+"
  (S1                  :      Chars_Ptr;
   S2                  :      String)
   return Chars_Ptr_Array;
```

Create an array containing S1 and S2.
Note that this function allocates memory to store S2 as a null terminated string. The user is responsible for calling Free on the resulting array.

```
procedure Free
  (A                   : in out Chars_Ptr_Array);
```

Free all the strings in A.

30 Package Gtkada_Multi_Paned

31 Package Bonobo

This is the root of the Bonobo hierarchy.

32 Package Canvas_Link

33 Package GObject

34 Package Gdk

This is the top level package of the Gdk hierarchy. It provides the type definitions used to access underlying C structures.

34.1 Types

subtype C_Proxy **is** Glib.C_Proxy;

subtype Gdk_Bitmap **is** Gdk_Drawable;

type Gdk_Colormap **is new** C_Proxy;

type Gdk_Drawable **is new** C_Proxy;

type Gdk_Font **is new** C_Proxy;

type Gdk_GC **is new** C_Proxy;

type Gdk_Image **is new** C_Proxy;

subtype Gdk_Pixmap **is** Gdk_Drawable;

type Gdk_Region **is new** C_Proxy;

type Gdk_Screen **is new** C_Proxy;

type Gdk_Visual **is new** C_Proxy;

subtype Gdk_Window **is** Gdk_Drawable;

type Gdk_Window_Attr **is new** C_Proxy;

35 Package Gdk.Bitmap

Pixmaps are off-screen drawables. They can be drawn upon with the standard drawing primitives, then copied to another drawable (such as a Gdk_Window) with Gdk.Drawable.Draw_Drawable. The depth of a pixmap is the number of bits per pixels. Bitmaps are simply pixmaps with a depth of 1. (That is, they are monochrome bitmaps - each pixel can be either on or off). See Gdk.Pixmap for more details on pixmap handling.

35.1 Types

subtype Gdk_Bitmap **is** Gdk.Gdk_Bitmap;

A black and white image. This type is mainly used as a mask when drawing other colored images. Each pixel can have two values, 0 or 1.

35.2 Subprograms

```
procedure Gdk_New
  (Bitmap              : out   Gdk_Bitmap;
   Window              :       Gdk.Window.Gdk_Window;
   Width               :       Gint;
   Height              :       Gint);
```

Create a new bitmap with a given size.
Window is used to determine default values for the new bitmap. Can be eventually null in which case the root window is used. Width is the width of the new bitmap in pixels. Height is the height of the new bitmap in pixels.

```
procedure Ref
  (Bitmap              :       Gdk_Bitmap);
```

Add a reference to a bitmap.

```
procedure Unref
  (Bitmap              :       Gdk_Bitmap);
```

This is the usual way to destroy a bitmap. The memory is freed when there is no more reference

```
procedure Create_From_Data
  (Bitmap              : out   Gdk_Bitmap;
   Window              :       Gdk.Window.Gdk_Window;
   Data                :       String;
   Width               :       Gint;
   Height              :       Gint);
```

Create a bitmap from data in XBM format.
Window is used to determine default values for the new bitmap, can be null in which case the root window is used. Data is the XBM data. Width is the width of the new bitmap in pixels. Height is the height of the new bitmap in pixels.

36 Package Gdk.Color

This package provides an interface to the color handling facilities in gtk+. It is able to handle any kind of visual (monochrome, greyscale, color with different depths, ...), but provides a common and easy interface for all of them. Some of these functions expect a Colormap. There are two ways you can get such a colormap, either a system default colormap or a per-widget colormap. It is recommended, unless you are writing your own new widget, to always use the system default Colormap. All the functions to get these colormaps are found in Gtk.Widget.

Getting the Red/Green/Blue components can be done through Parse, and is actually recommended, since the exact color generally depends on the visual your application is running on.

Note for users transitioning from gtk+ 1.2: the Get_System call is now obsolete, and you should use Gtk.Widget.Get_Default_Colormap instead.

36.1 Types

type Gdk_Color **is private**;

A color to be displayed on the screen. Currently, GtkAda only supports the RGB standard, ie each color is set by its red, green and blue components. An extra field (Pixel) is the internal representation of the color, which is set once the color has been allocated.

type Gdk_Color_Array **is array** (Natural **range** <>) **of** Gdk_Color;

An array of colors.

subtype Gdk_Colormap **is** Gdk.Gdk_Colormap;

The set of colors the can be displayed on the screen. When the screen is not a true-color screen (ie there is only a limited number of possible colors, like 256), the colors are in fact indexes into a colormap, which gives the components of the color. This is the same concept as a palette.

36.2 Subprograms

```
function Gdk_Color_Type        return Glib.GType;
```

Return the internal gtk+ types associated with a color

```
function Gdk_Colormap_Type     return Glib.GType;
```

Return the internal gtk+ types associated with a colormap

36.2.1 Setting/Getting the fields of Gdk_Color

```
procedure Set_Rgb
  (Color              : out   Gdk_Color;
   Red, Green, Blue   :      Guint16);
```

Modify the fields of the color.
You then have to allocate the color with one of the Alloc* functions above.

```
procedure Set_Pixel
  (Color              : in out Gdk_Color;
   Pixel              :      Guint32);
```

This function should almost never be used. Instead, use Alloc_Color.

```
function Red
  (Color              :      Gdk_Color)
   return Guint16;
```

Return the Red field of Color.

```
function Green
  (Color              :      Gdk_Color)
   return Guint16;
```

Return the Green field of Color.

```
function Blue
  (Color              :      Gdk_Color)
   return Guint16;
```

Return the Blue field of Color.

```
function Pixel
  (Color              :      Gdk_Color)
   return Guint32;
```

Return the Pixel field of Color.

36.2.2 Creating and Destroying colors

```
procedure Gdk_New
  (Colormap           : out   Gdk_Colormap;
   Visual             :      Gdk.Visual.Gdk_Visual;
   Private_Cmap       :      Boolean);
```

Create a new colormap for the visual.
If Private_Cmap is true, then the colormap won't be modifiable outside this scope. This might result in some strange colors on the display...

```
procedure Ref
  (Colormap           :      Gdk_Colormap);
```

Increment the ref-count for the color.

```
procedure Unref
  (Colormap           :      Gdk_Colormap);
```

Unref is the only way to destroy a colormap once you no longer need it.
Note that because gtk+ uses reference counts, the colormap will not be actually destroyed while at least one object is using it.

```
procedure Change
  (Colormap           :      Gdk_Colormap;
   Ncolors            :      Gint);
```

Change the first Ncolors defined in Colormap.

```
procedure Alloc_Colors
  (Colormap           :      Gdk_Colormap;
   Colors             : in out Gdk_Color_Array;
   Writeable          :      Boolean := False;
   Best_Match         :      Boolean := True;
   Success            : out   Boolean_Array;
   Result             : out   Gint);
```

Allocate a set of colors.
The parameters are the same as for Alloc_Color Result is the number of colors not successfully allocated.

The size of the Boolean_Array is equal to the length of the Colors_Array. Usage of an array of a different size will probably lead to a Constraint_Error.

```
procedure Alloc_Color
  (Colormap           :      Gdk_Colormap;
   Color              : in out Gdk_Color;
   Writeable          :      Boolean := False;
   Best_Match         :      Boolean := True;
   Success            : out   Boolean);
```

Allocate a new color.
The fields RGB should have been set before calling this function. If Writeable is True, the color will be allocated read/write, that can be changed at any time. Not all visuals support this. On modern systems this usage has become less useful than before, since redrawing the screen with a new color is about as fast. If Best_Match is True, and the exact color can not be allocated, GtkAda will find the closest possible match, and modify the fields Red, Green and Blue of Color. Note that the allocation has more chances to succeed if Writeable is False and Best_Match is True. When you no longer use a color, you should call Free.

```
procedure Free_Colors
  (Colormap           :      Gdk_Colormap;
   Colors             :      Gdk_Color_Array);
```

Free Colors, assuming they are allocated in Colormap.

```
procedure Get_Visual
  (Colormap           :      Gdk_Colormap;
   Visual             : out   Gdk.Visual.Gdk_Visual);
```

Get the visual associated with a colormap.
The main information you can get from there is the depth of the display.

```
procedure Copy
  (Source             :      Gdk_Color;
   Destination        : out   Gdk_Color);
```

Copy the Source color to Destination.

```
function Parse
  (Spec               :      String)
   return Gdk_Color;
```

Parse the string Spec, and get its Red/Green/Blue components.
The color is not allocated, and you need to call Alloc_Color. If the string could not be parsed to an existing color, Wrong_Color is raised. The string can be one of :

- "RGB:FF/FF/FF" where the "FF" substrings are respectively the value of the red, green and blue components. Some other prefixes than RGB are defined in the X11 definition, please see some X11 documentation (or the man page XParseColor on unix systems).
- "color_name" which can be any color name defined in the file rgb.txt of the user's system. You should always check that Wrong_Color was not raised, in case the color was not known on the user's system. This string is case insensitive. Color names are not supported on Windows systems.

```
function Equal
  (Colora, Colorb     :      Gdk_Color)
```

```
return Boolean;
```

True if the Red, Green and Blue components of both colors are equal.

36.3 Example

```
--  Here is an example how you can allocate a new color, when you know
--  its red/green/blue components: Note that we allocate white in fact
--  since the maximal value for color components is 65535.
   Color   : Gdk_Color;
   Success : Boolean;
   Set_Rbg (Color, 65535, 65535, 65535);
   Alloc_Color (Colormap   => Gtk.Widget.Get_Default_Colormap,
                Color      => Color,
                Writeable  => False,
                Best_Match => True,
                Success    => Success);
   if not Success then
       ...;  --  allocation failed
   end if;
```

37 Package Gdk.Cursor

This package provides the capability to create predefined mouse cursors as well as user defined ones.

37.1 Types

type Gdk_Cursor **is new** Gdk.C_Proxy;

```
type Gdk_Cursor_Type is
           (X_Cursor,
            Arrow,
            Based_Arrow_Down,
            Based_Arrow_Up,
            Boat,
            Bogosity,
            Bottom_Left_Corner,
            Bottom_Right_Corner,
            Bottom_Side,
            Bottom_Tee,
            Box_Spiral,
            Center_Ptr,
            Circle,
            Clock,
            Coffee_Mug,
            Cross,
            Cross_Reverse,
            Crosshair,
            Diamond_Cross,
            Dot,
            Dotbox,
            Double_Arrow,
            Draft_Large,
            Draft_Small,
            Draped_Box,
            Exchange,
            Fleur,
            Gobbler,
            Gumby,
            Hand1,
            Hand2,
            Heart,
            Icon,
            Iron_Cross,
            Left_Ptr,
            Left_Side,
            Left_Tee,
            Leftbutton,
            Ll_Angle,
            Lr_Angle,
            Man,
            Middlebutton,
            Mouse,
            Pencil,
            Pirate,
```

```
   Plus,
   Question_Arrow,
   Right_Ptr,
   Right_Side,
   Right_Tee,
   Rightbutton,
   Rtl_Logo,
   Sailboat,
   Sb_Down_Arrow,
   Sb_H_Double_Arrow,
   Sb_Left_Arrow,
   Sb_Right_Arrow,
   Sb_Up_Arrow,
   Sb_V_Double_Arrow,
   Shuttle,
   Sizing,
   Spider,
   Spraycan,
   Star,
   Target,
   Tcross,
   Top_Left_Arrow,
   Top_Left_Corner,
   Top_Right_Corner,
   Top_Side,
   Top_Tee,
   Trek,
   Ul_Angle,
   Umbrella,
   Ur_Angle,
   Watch,
   Xterm);
```

37.2 Subprograms

```
procedure Gdk_New
  (Widget             : out   Gdk_Cursor;
   Cursor_Type        :       Gdk_Cursor_Type);
```

Create a new standard cursor.

```
procedure Gdk_New
  (Widget             : out   Gdk_Cursor;
   Source             :       Gdk.Gdk_Pixmap;
   Mask               :       Gdk.Gdk_Pixmap;
   Fg                 :       Gdk.Color.Gdk_Color;
   Bg                 :       Gdk.Color.Gdk_Color;
   X                  :       Glib.Gint;
   Y                  :       Glib.Gint);
```

Create a new cursor from a given pixmap and mask.
See also Gdk.Pixbuf.Gdk_New_From_Pixbuf. Both the pixmap and mask must have a depth of 1 (i.e. each pixel has only 2 values - on or off). The standard cursor size is 16 by 16 pixels.

- Source is the pixmap specifying the cursor.

- Mask is the pixmap specifying the mask, which must be the same size as source.
- Fg is the foreground color, used for the bits in the source which are enabled. The color does not have to be allocated first.
- Bg is the background color, used for the bits in the source which are disabled. The color does not have to be allocated first.
- X is the horizontal offset of the 'hotspot' of the cursor.
- Y is the vertical offset of the 'hotspot' of the cursor.

```
procedure Gdk_New
  (Cursor             : out   Gdk_Cursor;
   Name               :       String);
```

Create a cursor from a name

```
procedure Destroy
  (Cursor             :       Gdk_Cursor);
procedure Ref
  (Cursor             :       Gdk_Cursor);
```

Increment the reference counting for the cursor.

```
procedure Unref
  (Cursor             :       Gdk_Cursor);
```

Decrement the reference counting for the cursor.
When this reaches 0, the cursor is destroyed.

38 Package Gdk.Display

Gdk_Display objects purpose are two fold: @itemize @bullet @item To grab/ungrab keyboard focus and mouse pointer @item To manage and provide information about the Gdk_Screen(s) available for this Gdk_Display Gdk_Display objects are the GDK representation of the X Display which can be described as a workstation consisting of a keyboard a pointing device (such as a mouse) and one or more screens. It is used to open and keep track of various Gdk_Screen objects currently instanciated by the application. It is also used to grab and release the keyboard and the mouse pointer.

@end itemize

38.1 Signals

- **"closed"**

```
procedure Handler
(Display  : access Gdk_Display_Record'Class;
Is_Error : Boolean);
```

The ::closed signal is emitted when the connection to the windowing system for display is closed. Is_Error is set to true if the connection is closed due to an error.

38.2 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal type associated with a Gdk_Display

```
function Open
  (Display_Name       :      String)
   return Gdk_Display;
```

Open a new display. Display_Name follows the unix convention, and should have the format host:screen, for instance "foo.com:0".

```
function Get_Default           return Gdk_Display;
```

Gets the default Gdk_Display

```
function Get_Name
  (Display            : access Gdk_Display_Record)
   return String;
```

Return the name of the screen

```
function Get_N_Screens
  (Display            : access Gdk_Display_Record)
   return Glib.Gint;
```

Return the number of screens managed by the display. See the function Gdk.Screen.Get_Screen or Gdk.Screen.Get_Default_Screen

```
procedure Pointer_Ungrab
  (Display            : access Gdk_Display_Record;
   Time               :      Glib.Guint32
                       := Gdk.Types.Current_Time);
```

Release any pointer grab.

```
procedure Keyboard_Ungrab
  (Display            : access Gdk_Display_Record;
   Time               :      Glib.Guint32
                       := Gdk.Types.Current_Time);
```

Release any keyboard grab

```
function Pointer_Is_Grabbed
  (Display            : access Gdk_Display_Record)
   return Boolean;
```

Test if the pointer is grabbed.

```
procedure Beep
  (Display            : access Gdk_Display_Record);
```

Emits a short beep on display

```
procedure Sync
  (Display            : access Gdk_Display_Record);
```

Flushes any requests queued for the windowing system and waits until all requests have been handled. This is often used for making sure that the display is synchronized with the current state of the program. Calling Sync before Gdk.Error.Trap_Pop makes sure that any errors generated from earlier requests are handled before the error trap is removed.

This is most useful for X11. On windowing systems where requests are handled synchronously, this function will do nothing.

```
procedure Flush
  (Display            : access Gdk_Display_Record);
```

Flushes any requests queued for the windowing system; this happens automatically when the main loop blocks waiting for new events, but if your application is drawing without returning control to the main loop, you may need to call this function explicitely. A common case where this function needs to be called is when an application is executing drawing commands from a thread other than the thread where the main loop is running.

This is most useful for X11. On windowing systems where requests are handled synchronously, this function will do nothing.

```
procedure Close
  (Display            : access Gdk_Display_Record);
```

Closes the connection to the windowing system for the given display, and cleans up associated resources.

```
function Get_Event
  (Display            : access Gdk_Display_Record)
   return Gdk.Event.Gdk_Event;
```

Gets the next Gdk_Event to be processed for Display, fetching events from the windowing system if necessary. null is returned if no events are pending. The returned Gdk_Event must be freed with Gdk.Event.Free

```
function Peek_Event
  (Display            : access Gdk_Display_Record)
   return Gdk.Event.Gdk_Event;
```

Gets a copy of the first Gdk_Event in the Display's event queue, without removing the event from the queue. (Note that this function will not get more events from the windowing system. It only checks the events that have already been moved to the GDK event queue.) null is returned if there are no events in the queue. The returned event should be freed with Gdk.Event.Free.

```
procedure Put_Event
  (Display            : access Gdk_Display_Record;
   Event              :      Gdk.Event.Gdk_Event);
```

Appends a copy of the given event onto the front of the event queue for Display.

```
procedure Set_Double_Click_Time
  (Display            : access Gdk_Display_Record;
   Msec               :      Glib.Guint);
```

Sets the double click time (two clicks within this time interval count as a double click and result in a GDK_2BUTTON_PRESS event). Applications should not set this, it is a global user-configured setting.

```
procedure Set_Double_Click_Distance
  (Display            : access Gdk_Display_Record;
   Distance           :      Glib.Guint);
```

Sets the double click distance (two clicks within this distance count as a double click and result in a GDK_2BUTTON_PRESS event). See also Set_Double_Click_Time. Applications should not set this, it is a global user-configured setting.

```
procedure Get_Window_At_Pointer
  (Display            : access Gdk_Display_Record;
   Win_X              : out   Glib.Gint;
   Win_Y              : out   Glib.Gint;
   Win                : out   Gdk.Gdk_Window);
```

Obtains the window underneath the mouse pointer, returning the location of that window in Win_X, Win_Y. Returns nullif the window under the mouse pointer is not known to GDK (for example, belongs to another application). (Win_X, Win_Y) are relative to the origin of the window under the pointer.

```
function Supports_Cursor_Color
  (Display            : access Gdk_Display_Record)
   return Boolean;
```

Returns TRUE if multicolored cursors are supported on display. Otherwise, cursors have only a forground and a background color.

```
function Supports_Cursor_Alpha
  (Display            : access Gdk_Display_Record)
   return Boolean;
```

Returns TRUE if cursors can use an 8bit alpha channel on display. Otherwise, cursors are restricted to bilevel alpha (i.e. a mask).

```
function Get_Default_Cursor_Size
  (Display            : access Gdk_Display_Record)
   return Glib.Guint;
```

Returns the default size to use for cursors on display.

```
procedure Get_Maximal_Cursor_Size
  (Display            : access Gdk_Display_Record;
   Width              : out   Glib.Guint;
   Height             : out   Glib.Guint);
```

Gets the maximal size to use for cursors on display

```
function Get_Default_Group
  (Display            : access Gdk_Display_Record)
   return Gdk.Gdk_Window;
```

Returns the default group leader window for all toplevel windows on display. This window is implicitly created by GDK. See Gdk.Window.Set_Group.

```
function Supports_Selection_Notification
  (Display            : access Gdk_Display_Record)
   return Boolean;
```

Returns whether Gdk.Event.Owner_Change events will be sent when the owner of a selection changes.

```
function Request_Selection_Notification
  (Display            : access Gdk_Display_Record;
   Selection          :      Gdk.Types.Gdk_Atom)
   return Boolean;
```

Request Gdk.Event.Owner_Change events for ownership changes of the selection named by the given atom.

```
function Supports_Clipboard_Persistence
  (Display            : access Gdk_Display_Record)
   return Boolean;
```

Returns whether the specifed display supports clipboard persistance; i.e. if it's possible to store the clipboard data after an application has quit. On X11 this checks if a clipboard daemon is running.

```
procedure Store_Clipboard
  (Display            : access Gdk_Display_Record;
   Clipboard_Window   :      Gdk.Gdk_Window;
   Time               :      Glib.Guint32;
   Targets            :      Gdk.Types.Gdk_Atom_Array);
```

Issues a request to the clipboard manager to store the clipboard data. On X11, this is a special program that works according to the freedesktop clipboard specification, available at http://www.freedesktop.org/Standards/clipboard-manager-spec. See also Gtk.Clipboard.Store.

39 Package Gdk.Drawable

This package provides support for drawing points, lines, arcs and text onto what are called 'drawables'. Drawables, as the name suggests, are things which support drawing onto them, and are either Gdk_Window or Gdk_Pixmap objects.

Many of the drawing operations take a Gdk_GC argument, which represents a graphics context. This Gdk_GC contains a number of drawing attributes such as foreground color, background color and line width, and is used to reduce the number of arguments needed for each drawing operation. See Gdk.GC for more information.

39.1 Types

subtype Gdk_Drawable **is** Gdk.Gdk_Drawable;

A screen area that can be drawn upon.

39.2 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gdk_Drawable.

```
procedure Get_Size
  (Drawable           :      Gdk_Drawable;
   Width              : out  Gint;
   Height             : out  Gint);
```

Return the width and height of a given drawable.

```
procedure Set_Colormap
  (Drawable           :      Gdk_Drawable;
   Colormap           :      Gdk.Gdk_Colormap);
function Get_Colormap
  (Drawable           :      Gdk_Drawable)
   return Gdk.Gdk_Colormap;
function Get_Visual
  (Drawable           :      Gdk_Drawable)
   return Gdk.Gdk_Visual;
function Get_Depth
  (Drawable           :      Gdk_Drawable)
   return Gint;
procedure Ref
  (Drawable           :      Gdk_Drawable);
procedure Unref
  (Drawable           :      Gdk_Drawable);
procedure Draw_Point
  (Drawable           :      Gdk_Drawable;
   GC                 :      Gdk.Gdk_GC;
   X                  :      Gint;
   Y                  :      Gint);
```

Draw a point, using the foreground color and other attributes of the Gc.

```
procedure Draw_Line
  (Drawable           :      Gdk_Drawable;
   GC                 :      Gdk.Gdk_GC;
```

```
   X1                  :      Gint;
   Y1                  :      Gint;
   X2                  :      Gint;
   Y2                  :      Gint);
```

Draw a line, using the foreground color and other attributes of the Gc.
(X1, Y1) is coordinate of the start point. (X2, Y2) is coordinate of the end point.

```
procedure Draw_Rectangle
  (Drawable            :      Gdk_Drawable;
   GC                  :      Gdk.Gdk_GC;
   Filled              :      Boolean := False;
   X                   :      Gint;
   Y                   :      Gint;
   Width               :      Gint;
   Height              :      Gint);
```

Draw a rectangular outline or filled rectangle.
Note that a rectangle drawn filled is 1 pixel smaller in both dimensions than a rectangle outlined. Calling Draw_Rectangle (Window, Gc, True, 0, 0, 20, 20) results in a filled rectangle 20 pixels wide and 20 pixels high. Calling Draw_Rectangle (Window, Gc, False, 0, 0, 20, 20) results in an outlined rectangle with corners at (0, 0), (0, 20), (20, 20), and (20, 0), which makes it 21 pixels wide and 21 pixels high.

(X, Y) represents the coordinate of the top-left edge of the rectangle.

```
procedure Draw_Arc
  (Drawable            :      Gdk_Drawable;
   GC                  :      Gdk.Gdk_GC;
   Filled              :      Boolean := False;
   X                   :      Gint;
   Y                   :      Gint;
   Width               :      Gint;
   Height              :      Gint;
   Angle1              :      Gint;
   Angle2              :      Gint);
```

Draws an arc or a filled 'pie slice'.
The arc is defined by the bounding rectangle of the entire ellipse, and the start and end angles of the part of the ellipse to be drawn. Filled is True if the arc should be filled, producing a 'pie slice'. (X, Y) represent the coordinate of the top-left edge of the bounding rectangle. Angle1 is the start angle of the arc, relative to the 3 o'clock position, counter-clockwise, in 1/64ths of a degree. Angle2 is the end angle of the arc, relative to angle1, in 1/64ths of a degree.

```
procedure Draw_Polygon
  (Drawable            :      Gdk_Drawable;
   GC                  :      Gdk.Gdk_GC;
   Filled              :      Boolean;
   Points              :      Gdk.Types.Gdk_Points_Array);
```

Draw an outlined or filled polygon.
Filled is True if the polygon should be filled. The polygon is closed automatically, connecting the last point to the first point if necessary. Points is an array of Gdk_Point specifying the points making up the polygon.

```
procedure Draw_Text
  (Drawable            :      Gdk_Drawable;
   Font                :      Gdk.Gdk_Font;
   GC                  :      Gdk.Gdk_GC;
```

```
   X                 :      Gint;
   Y                 :      Gint;
   Text              :      UTF8_String);
```

Draw a string in the given font or fontset.
X is the x coordinate of the left edge of the text. Y is the y coordinate of the baseline of the text.

You should use Gtk.Widget.Create_Pango_Layout instead to handle internationalization.

```
procedure Draw_Text
  (Drawable          :      Gdk_Drawable;
   Font              :      Gdk.Gdk_Font;
   GC                :      Gdk.Gdk_GC;
   X                 :      Gint;
   Y                 :      Gint;
   Wide_Text         :      Gdk.Types.Gdk_WString);
```

Draw a wide string in the given font of fontset.
If the font is a 1-byte font, the string is converted into 1-byte characters (discarding the high bytes) before output.

```
procedure Draw_Drawable
  (Drawable          :      Gdk_Drawable;
   GC                :      Gdk.Gdk_GC;
   Src               :      Gdk_Drawable;
   Xsrc              :      Gint;
   Ysrc              :      Gint;
   Xdest             :      Gint;
   Ydest             :      Gint;
   Width             :      Gint := -1;
   Height            :      Gint := -1);
```

Draw a pixmap, or a part of a pixmap, onto another drawable.
Src is the source Gdk_Drawable to draw. Xsrc is the left edge of the source rectangle within Src. Ysrc is the top of the source rectangle within Src. Xdest is the x coordinate of the destination within Src. Ydest is the y coordinate of the destination within Src. Width is the width of the area to be copied, or -1 to make the area extend to the right edge of the source pixmap. Height is the height of the area to be copied, or -1 to make the area extend to the bottom edge of the source pixmap.

```
procedure Draw_Layout
  (Drawable          :      Gdk_Drawable;
   GC                :      Gdk.Gdk_GC;
   X                 :      Gint;
   Y                 :      Gint;
   Layout            :      Pango.Layout.Pango_Layout);
```

Display the layout and its text in Drawable. This method should be preferred over Draw_Text.

```
procedure Draw_Pixmap
  (Drawable          :      Gdk_Drawable;
   GC                :      Gdk.Gdk_GC;
   Src               :      Gdk_Drawable;
   Xsrc              :      Gint;
   Ysrc              :      Gint;
   Xdest             :      Gint;
   Ydest             :      Gint;
   Width             :      Gint := -1;
   Height            :      Gint := -1);
```

Deprecated, use Draw_Drawable instead.

```
procedure Draw_Image
  (Drawable           :      Gdk_Drawable;
   GC                 :      Gdk.Gdk_GC;
   Image              :      Gdk.Gdk_Image;
   Xsrc               :      Gint;
   Ysrc               :      Gint;
   Xdest              :      Gint;
   Ydest              :      Gint;
   Width              :      Gint := -1;
   Height             :      Gint := -1);
```

Draw a Gdk_Image onto a Drawable.
The depth of the Gdk_Image must match the depth of the Gdk_Drawable. Image is the Gdk_Image to draw. Xsrc is the left edge of the source rectangle within Image. Ysrc is the top of the source rectangle within Image. Xdest is the x coordinate of the destination within Drawable. Ydest is the y coordinate of the destination within Drawable. Width is the width of the area to be copied, or -1 to make the area extend to the right edge of image. Height is the height of the area to be copied, or -1 to make the area extend to the bottom edge of image.

```
procedure Draw_Points
  (Drawable           :      Gdk_Drawable;
   GC                 :      Gdk.Gdk_GC;
   Points             :      Gdk.Types.Gdk_Points_Array);
```

Draw a number of points.
Use the foreground color and other attributes of the Gc.

```
procedure Draw_Segments
  (Drawable           : in   Gdk_Drawable;
   GC                 : in   Gdk.Gdk_GC;
   Segs               : in   Gdk.Types.Gdk_Segments_Array);
```

Draw a number of unconnected lines.

```
procedure Draw_Lines
  (Drawable           :      Gdk_Drawable;
   GC                 :      Gdk.Gdk_GC;
   Points             :      Gdk.Types.Gdk_Points_Array);
```

Draw a series of lines connecting the given points.
The way in which joins between lines are drawn is determined by the Cap_Style value in the Gdk_GC. This can be set with Gdk.Gc.Set_Line_Attributes.

```
function Get_Image
  (Drawable           :      Gdk_Drawable;
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint)
   return Gdk_Image;

function Get_Clip_Region
  (Drawable           :      Gdk_Drawable)
   return Gdk.Gdk_Region;

function Get_Visible_Region
  (Drawable           :      Gdk_Drawable)
   return Gdk.Gdk_Region;
```

39.3 Example

```
with Glib;
with Gdk.Window;
with Gdk.Drawable;
with Gdk.GC;
with Gdk.Font;
with Gtk.Drawing_Area;

procedure Draw (Drawing : in out Gtk.Drawing_Area.Gtk_Drawing_Area) is
   Gdkw : Gdk.Window.Gdk_Window;
   GC   : Gdk.GC.Gdk_GC;
   Font : Gdk.Font.Gdk_Font;
   use type Glib.Gint;

begin
   -- Get the Gdk window

   Gdkw := Gtk.Drawing_Area.Get_Window (Drawing);

   -- Clear the window

   Gdk.Window.Clear (Gdkw);

   -- Create a graphic context associated with this window

   Gdk.GC.Gdk_New (GC, Gdkw);

   -- Draw a line in this window

   Gdk.Drawable.Draw_Line
     (Drawable => Gdkw,
      GC => GC,
      X1 =>   0, Y1 =>   0,
      X2 => 100, Y2 => 100);

   -- Draw an arc

   Gdk.Drawable.Draw_Arc
     (Drawable => Gdkw,
      Gc       => GC,
      Filled   => True,
      X        => 100,
      Y        => 100,
      Width    => 200,
      Height   => 100,
```

```
         Angle1   => 0 * 64,
         Angle2   => 270 * 64);

      -- Ask for a given font

      Gdk.Font.Load (Font,
                     "-adobe-courier-medium-i-*-*-15-*-*-*-*-*-*-*");
      Gdk.Drawable.Draw_Text
        (Drawable    => Gdkw,
         Font        => Font,
         Gc          => GC,
         X           => 50,
         Y           => 50,
         Text        => "Hello World");
      Gdk.GC.Destroy (GC);
   end Draw;
```

40 Package Gdk.Event

This package provides functions dealing with events from the window system. In GtkAda applications, the events are handled automatically in Gtk.Main.Do_Event, and passed on to the appropriate widgets, so these functions are rarely needed.

!! Warning !! This is one of the only package that requires manual memory management in some cases. If you use the function Allocate, you have to use the function Free too...

40.1 Types

type Event_Handler_Func **is access procedure**

type Gdk_Crossing_Mode **is**
```
          (Crossing_Normal, Crossing_Grab, Crossing_Ungrab);
```

type Gdk_Device_Id **is new** Guint32;

This type is specific to GtkAda. In Gdk, guint32 is used instead.

type Gdk_Event **is new** Gdk.C_Proxy;

subtype Gdk_Event_Any **is** Gdk_Event;

Change from GtkAda1.2.3: There is no longer a tagged type hierarchy, only one type. However there are now a few runtime tests for each of the function, to check whether a given field is available or not. Fields common to all events: Window, Send_Event, Event_Type

subtype Gdk_Event_Button **is** Gdk_Event;

A button was pressed or released. Relevant fields: Time, X, Y, Axes, State, Button, Device_Id, X_Root, Y_Root, Window. Type: Button_Press, Gdk_2Button_Press, Gdk_3Button_Press or Button_Release.

subtype Gdk_Event_Client **is** Gdk_Event;

This is an event used to send arbitrary data from one X application to another. This event too is almost never used, and is not documented here. Please consult an X11 documentation for more information. Relevant fields: Message_Type, Data Type: Client_Event

type Gdk_Event_Client_Data_Format **is**
```
          (Char_Array, Short_Array, Long_Array);
```

subtype Gdk_Event_Configure **is** Gdk_Event;

The window configuration has changed: either it was remapped, resized, moved, ... Note that you usually don't have to redraw your window when you receive such an event, since it is followed by an Gdk_Event_Expose. Relevant fields: X, Y, Width, Height Type: Configure

subtype Gdk_Event_Crossing **is** Gdk_Event;

The mouse has been moved in or out of the window Relevant fields: SubWindow, Time, X, Y, X_Root, Y_Root, Mode, Detail, Focus, State Type: Enter_Notify, Leave_Notify

subtype Gdk_Event_DND **is** Gdk_Event;

??? Relevant fields: Context, Time, X_Root, Y_Root Type: Drag_Enter, Drag_Leave, Drag_Motion, Drag_Status, Drop_Start, Drop_Finished

subtype Gdk_Event_Expose **is** Gdk_Event;

The window needs to be redrawn. For efficiency, gtk gives you the smallest area that you need to redraw. Relevant fields: Area, Region, Count Type: Expose

subtype Gdk_Event_Focus **is** Gdk_Event;

The focus has changed for a window. Relevant fields: in Type: Focus_Change

subtype Gdk_Event_Key **is** Gdk_Event;

A keyboard key was pressed Relevant fields: Time, State, Key_Val, String, Hardware_Keycode, Group Type: Key_Press, Key_Release

type Gdk_Event_Mask **is mod** 2 ** 32;

Note that you need to change the event mask of a widget if you want to be able to get some events. To change this mask, the widget must first be Unrealized.

subtype Gdk_Event_Motion **is** Gdk_Event;

The mouse has moved Relevant fields: Time, X, Y, Axes, State, Is_Hint, Device_Id, X_Root, Y_Root Type: Motion_Notify

subtype Gdk_Event_No_Expose **is** Gdk_Event;

Indicate that the source region was completely available when parts of a drawable were copied. This is also emitted when a gc whose "exposures" attribute is set to False in a call to Copy_Area or Draw_Pixmap. See the documentation for Gdk.GC.Set_Exposures. No Relevent fields except the common ones Type: No_Expose

subtype Gdk_Event_Property **is** Gdk_Event;

Some property of the window was modified. GtkAda provides a higher level interface, and you almost never need to use this event. Relevent fields: Atom, Time, Property_State Type: Property_Notify

subtype Gdk_Event_Proximity **is** Gdk_Event;

This event type will be used pretty rarely. It only is important for XInput aware programs that are drawing their own cursor. This is only used with non standard input devices, like graphic tablets. Relevant fields: Time, Device_Id Type: Proximity_In, Proximity_Out

subtype Gdk_Event_Scroll **is** Gdk_Event;

A button was pressed or released. Relevant fields: Time, X, Y, State, Direction, Device_Id, X_Root, Y_Root Type: Scroll

subtype Gdk_Event_Selection **is** Gdk_Event;

This is how X11 implements a simple cut-and-paste mechanism. However, GtkAda provides a higher level interface to the selection mechanism, so this event will almost never be used. Relevant fields: Selection, Target, Property, Time, Requestor Type: Selection_Clear, Selection_Request, Selection_Notify

subtype Gdk_Event_Setting **is** Gdk_Event;

??? Relevant fields: Action, Name. Type: ???

```
type Gdk_Event_Type is
          (Nothing,
              -- No event occurred.

           Delete,
              -- A window delete event was sent by the window manager. The specified
              -- window should be deleted.

           Destroy,
              -- A window has been destroyed.

           Expose,
              -- Part of a window has been uncovered.

           Motion_Notify,
           Button_Press,
              -- A mouse button was pressed.

           Gdk_2button_Press,
              -- Double-click

           Gdk_3button_Press,
              -- Triple-click

           Button_Release,
              -- A mouse button was released.

           Key_Press,
              -- A key was pressed.

           Key_Release,
              -- A key was released.

           Enter_Notify,
              -- A window was entered.
```

```
Leave_Notify,
   -- A window was exited.

Focus_Change,
   -- The focus window has changed. (The focus window gets keyboard events)
```

subtype Gdk_Event_Visibility **is** Gdk_Event;

The visibility state of the window (partially visibly, fully visible, hidden). This event almost never need to be used, since other events are generated at the same time, like expose_events Relevant fields: Visibility_State type: Visibility_Notify

subtype Gdk_Event_Window_State **is** Gdk_Event;

??? Relevant fields: Changed_Mask, New_Window_State. Type: Delete, Destroy, Map, Unmap ???

```
type Gdk_Notify_Type is
          (Notify_Ancestor,
           Notify_Virtual,
           Notify_Inferior,
           Notify_Non_Linear,
           Notify_Non_Linear_Virtual,
           Notify_Unknown);
```

```
type Gdk_Property_State is
          (Property_New_Value, Property_Delete);
```

```
type Gdk_Scroll_Direction is
          (Scroll_Up,
           Scroll_Down,
           Scroll_Left,
           Scroll_Right);
```

```
type Gdk_Setting_Action is
          (Setting_Action_New,
           Setting_Action_Changed,
           Setting_Action_Deleted);
```

```
type Gdk_Visibility_State is
          (Visibility_Unobscured,
```

```
Visibility_Partial,
Visibility_Fully_Obscured);
```

type Gdk_Window_State **is mod** 2 ** 32;

State of a Window. Default is 0.

type Property_Gdk_Event_Mask **is new** Event_Mask_Properties.Property;

40.2 Subprograms

40.2.1 Access to fields of the event

The following functions can be used to retrieve some specific fields@* from an event. Some of these fields do not exist for all the types of events (see the description of each event for a list of the relevant fields). Note also that you can not pass a null event to them. The parameter must be a correct event, or the result is undefined.

```
function Get_Event_Type
  (Event              :      Gdk_Event)
   return Gdk_Event_Type;
```

The type of the event.

```
function Get_Send_Event
  (Event              :      Gdk_Event)
   return Boolean;
```

Set to true if the event was generated by the application, False
if generated by the X server/Win32.

```
function Get_Window
  (Event              :      Gdk_Event)
   return Gdk.Gdk_Window;
```

The window the event occured on.
See the function Gdk.Window.Get_User_Data to get the actual widget.

```
function Get_Time
  (Event              :      Gdk_Event)
   return Guint32;
```

Time when the event occured.

```
function Get_X
  (Event              :      Gdk_Event)
   return Gdouble;
```

Horizontal coordinate of the mouse when the event occured.
The coordinates are relative to the parent window.

```
function Get_Y
  (Event              :      Gdk_Event)
   return Gdouble;
```

Vertical coordinate of the mouse when the event occured.
The coordinates are relative to the parent window.

```
function Get_X_Root
  (Event              :      Gdk_Event)
   return Gdouble;
```

Horizontal coordinate of the mouse when the event occured.
Relative to the root window.

```
function Get_Y_Root
  (Event              :      Gdk_Event)
   return Gdouble;
```

Vertical coordinate of the mouse when the event occured.
Relative to the root window.

```
function Get_Button
  (Event              :      Gdk_Event)
   return Guint;
```

Number of the button that was pressed.

```
function Get_State
  (Event              :      Gdk_Event)
   return Gdk.Types.Gdk_Modifier_Type;
```

State of the mouse buttons and keyboard keys just prior to the event.

```
function Get_Subwindow
  (Event              :      Gdk_Event)
   return Gdk.Gdk_Window;
```

Child window for the event.
For an Enter_Notify_Event, this is set to the initial window for the pointer; for an Leave_Notify_Event this is set to the window occupied by the pointer in its last position.

```
function Get_Mode
  (Event              :      Gdk_Event)
   return Gdk_Crossing_Mode;
```

Return the mode of an Event.
Set to indicate whether the events are normal events, pseudo-motion events when a grab activates or pseudo-motion events when a grab deativates.

```
function Get_Detail
  (Event              :      Gdk_Event)
   return Gdk_Notify_Type;
```

Set to indicate the notify details.
Most applications can ignore events with a Notify Virtual or a Notify_Non_Linear_Virtual detail.

```
function Get_Focus
  (Event              :      Gdk_Event)
   return Boolean;
```

Set to true if the window for the event is the focus window.

```
function Get_Width
  (Event              :      Gdk_Event)
   return Gint;
```

Get the width in a configure event.

```
function Get_Height
  (Event              :      Gdk_Event)
   return Gint;
```

Get the height in a configure event.

```
function Get_Direction
  (Event               :      Gdk_Event)
   return Gdk_Scroll_Direction;
```

Get the direction in a scroll event.

```
function Get_Device_Id
  (Event               :      Gdk_Event)
   return Gdk_Device_Id;
```

Set to a constant for now in the gtk+ source... Probably useless.
Since multiple input devices can be used at the same time, like a mouse and a graphic tablet, this indicates which one generated the event.

```
function Get_Area
  (Event               :      Gdk_Event)
   return Rectangle.Gdk_Rectangle;
```

The minimal area on which the event applies.
For Expose_Events, this is the minimal area to redraw.

```
function Get_Region
  (Event               :      Gdk_Event)
   return Gdk.Region.Gdk_Region;
```

Return the region to which the event applies.
Do not free the returned value

```
function Get_Count
  (Event               :      Gdk_Event)
   return Gint;
```

Number of Expose_Events that are to follow this one.
Most applications can ignore the event if Count is not 0, which also allows for optimizations.

```
function Get_In
  (Event               :      Gdk_Event)
   return Boolean;
```

True if the window has gained the focus, False otherwise.

```
function Get_Is_Hint
  (Event               :      Gdk_Event)
   return Boolean;
```

???

```
function Get_Key_Val
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Key_Type;
```

Code of the key that was pressed (and that generated the event).

```
function Get_Group
  (Event               :      Gdk_Event)
   return Guint8;
```

Group of the key that was pressed;

```
function Get_Hardware_Keycode
  (Event               :      Gdk_Event)
   return Guint16;
```

Hardware key code of the key that was pressed.

```
function Get_String
  (Event               :      Gdk_Event)
   return String;
```

Symbol of the key that was pressed, as a string.

```
function Get_Atom
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Atom;
```

Indicate which property has changed.
??? Atom should not be a Guint

```
function Get_Property_State
  (Event               :      Gdk_Event)
   return Guint;
```

??? The return type should be changed.

```
function Get_Visibility_State
  (Event               :      Gdk_Event)
   return Gdk_Visibility_State;
```

Return the new visibility state for the window.

```
function Get_Selection
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Atom;
```

What was selected in the window...

```
function Get_Target
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Atom;
```

???

```
function Get_Property
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Atom;
```

???

```
function Get_Requestor
  (Event               :      Gdk_Event)
   return Guint32;
```

???

```
function Get_Message_Type
  (Event               :      Gdk_Event)
   return Gdk.Types.Gdk_Atom;
```

???

```
function Get_Data
  (Event               :      Gdk_Event)
   return Gdk_Event_Client_Data;
```

???

40.2.2 Modifying the fields of an event

```
procedure Set_Window
  (Event               :      Gdk_Event;
   Win                 :      Gdk.Gdk_Window);
```

Set the Window field of an event.

```
procedure Set_X
  (Event               :      Gdk_Event;
   X                   :      Gdouble);
```

Set the X field of an event.

```
procedure Set_Y
  (Event               :      Gdk_Event;
   Y                   :      Gdouble);
```

Set the Y field of an event.

```
procedure Set_Xroot
  (Event               :      Gdk_Event;
   Xroot               :      Gdouble);
```

Set the Xroot field of an event.

```
procedure Set_Yroot
  (Event               :      Gdk_Event;
   Yroot               :      Gdouble);
```

Set the Yroot field of an event.

```
procedure Set_Width
  (Event               :      Gdk_Event;
   Width               :      Gint);
```

Set the Width field of an event.

```
procedure Set_Height
  (Event               :      Gdk_Event;
   Height              :      Gint);
```

Set the Height field of an event.

```
procedure Set_Button
  (Event               :      Gdk_Event;
   Button              :      Guint);
```

Set the Button field of an event.

```
procedure Set_Time
  (Event               :      Gdk_Event;
   Time                :      Guint32);
```

Set the time for the event.
If Time is 0, then it is set to the current time.

```
procedure Set_State
  (Event               :      Gdk_Event;
   State               :      Gdk.Types.Gdk_Modifier_Type);
```

Set the State field of an event.

```
procedure Set_Subwindow
  (Event               :      Gdk_Event;
   Window              :      Gdk.Gdk_Window);
```

Set the Subwindow field of an event.

```
procedure Set_Mode
  (Event               :      Gdk_Event;
   Mode                :      Gdk_Crossing_Mode);
```

Set the Mode field of an event.

```
procedure Set_Detail
  (Event               :      Gdk_Event;
   Detail              :      Gdk_Notify_Type);
```

Set the Detail field of an event.

```
procedure Set_Focus
  (Event               :      Gdk_Event;
   Has_Focus           :      Boolean);
```

Set the Focus field of an event.

```
procedure Set_Area
  (Event              :      Gdk_Event;
   Area               :      Rectangle.Gdk_Rectangle);
```

Set the Area field of an event.

```
procedure Set_In
  (Event              :      Gdk_Event;
   Focus_In           :      Boolean);
```

Set the In field of an event.

```
procedure Set_Is_Hint
  (Event              :      Gdk_Event;
   Is_Hint            :      Boolean);
```

Set the Is_Hint field of an event.

```
procedure Set_Key_Val
  (Event              :      Gdk_Event;
   Key                :      Gdk.Types.Gdk_Key_Type);
```

Set the Key_Val field of an event.

```
procedure Set_Group
  (Event              :      Gdk_Event;
   Group              :      Guint8);
```

Set the group field of a key event.

```
procedure Set_Hardware_Keycode
  (Event              :      Gdk_Event;
   Keycode            :      Guint16);
```

Set the hardware key code field of a key event.

```
procedure Set_Direction
  (Event              :      Gdk_Event;
   Direction          :      Gdk_Scroll_Direction);
```

Set the direction field of a scroll event.

```
procedure Set_Atom
  (Event              :      Gdk_Event;
   Atom               :      Gdk.Types.Gdk_Atom);
```

Set the Atom field of an event.

```
procedure Set_Property_State
  (Event              :      Gdk_Event;
   State              :      Guint);
```

Set the Property_State field of an event.

```
procedure Set_Visibility_State
  (Event              :      Gdk_Event;
   State              :      Gdk_Visibility_State);
```

Set the Visibility_State field of an event.

```
procedure Set_Selection
  (Event              :      Gdk_Event;
   Selection          :      Gdk.Types.Gdk_Atom);
```

Set the Selection field of an event.

```
procedure Set_Target
  (Event              :      Gdk_Event;
   Target             :      Gdk.Types.Gdk_Atom);
```

Set the Target field of an event.

```
procedure Set_Property
  (Event              :      Gdk_Event;
   Property           :      Gdk.Types.Gdk_Atom);
```

Set the Property field of an event.

```
procedure Set_Requestor
  (Event              :      Gdk_Event;
   Requestor          :      Guint32);
```

Set the Requestor field of an event.

```
procedure Set_Message_Type
  (Event              :      Gdk_Event;
   Typ                :      Gdk.Types.Gdk_Atom);
```

Set the Message_Type field of an event.

```
procedure Set_String
  (Event              :      Gdk_Event;
   Str                :      String);
```

Set the string associated with an event.

40.2.3 General functions

```
function Get_Type               return GType;
```

Return the type corresponding to a Gdk_Event.

```
procedure Deep_Copy
  (From               :      Gdk_Event;
   To                 : out  Gdk_Event);
```

Deep copy for an event. The C structure is itself duplicated.
You need to deallocated it yourself with a call to Free below.

```
procedure Get_Graphics_Expose
  (Event              : out  Gdk_Event_Expose;
   Window             :      Gdk.Gdk_Window);
```

Waits for a GraphicsExpose or NoExpose event
If it gets a GraphicsExpose event, it returns a pointer to it, otherwise it returns an event for which Is_Created is False.

This function can be used to implement scrolling: you must call Gdk.GC.Set_Exposures with True on the GC you are using for the drawing, so that a events are generated for obscured areas and every time a new part of the widget is drawn. However, there is a race condition if multiple scrolls happen before you have finished processing the first one. A workaround is to call Get_Graphics_Expose after every scroll until it returns a null event.

```
function Events_Pending         return Boolean;
```

Is there any event pending on the queue ?

```
procedure Get
  (Event              : out  Gdk_Event);
```

Get the next event on the queue.

```
procedure Peek
  (Event              : out  Gdk_Event);
```

Look at the next event on the queue, but leave if there.

```
procedure Put
  (Event              :      Gdk_Event);
```

Add an event on the queue - Better to use Gtk.Signal.Emit_By_Name

```
procedure Set_Show_Events
  (Show_Events        :      Boolean := True);
```

For debug purposes, you can choose whether you want to see the events GtkAda receives.

```
function Get_Show_Events       return Boolean;
```

Return the current state of Show_Events.

```
procedure Send_Client_Message_To_All
  (Event              :      Gdk_Event);
```

Low level routine to send an Event to every window.

```
function Send_Client_Message
  (Event              :      Gdk_Event;
   Xid                :      Guint32)
   return Boolean;
```

Low level routine to send an Event to a specified X window.

```
procedure Allocate
  (Event              : out   Gdk_Event;
   Event_Type         :      Gdk_Event_Type;
   Window             :      Gdk.Gdk_Window);
```

Create an event, whose fields are uninitialized.
You need to use the function Set_* above to modify them, before you can send the event with Emit_By_Name. !!Note!!: The event has to be freed if you have called this function. Use the function Free below.

```
procedure Free
  (Event              : in out Gdk_Event);
```

Free the memory (and C structure) associated with an event.
You need to call this function only if the event was created through Allocate, not if it was created by GtkAda itself (or you would get a segmentation fault).

```
procedure Event_Handler_Set
  (Func               :      Event_Handler_Func;
   Data               :      System.Address);
```

Set up a new event handler.
This handler replaces the default GtkAda event handler, and thus should make sure that all events are correctly handled.

Note that managing the memory for Data is your responsability, and Data is passed as is to Func.

```
function From_Address
  (C                  :      System.Address)
   return Gdk_Event;
```

Convert a C handler to the matching Event structure.

```
function To_Address
  (C                  :      Gdk_Event)
   return System.Address;
```

Convert an event to the underlying C handler.

```
function Is_Created
  (E                  :      Gdk_Event)
   return Boolean;
```

Return True if the underlying C event has been created.

40.2.4 GValue support

```
function Get_Event
  (Value              :      Glib.Values.GValue)
   return Gdk_Event;
```

Convert a value into a Gdk_Event.

40.2.5 Event Recording

```
procedure Set_Follow_Events
  (Follow_Events      :      Boolean := True);
```

Set whether windows should follow events that they normally don't
(such as motion events) for event recording purposes. This function is used in cunjonction with GtkAda.Macro

```
function Get_Follow_Events     return Boolean;
```

Return follow_events value.
See Set_Follow_Events for more details.

41 Package Gdk.Font

!!! Important note !!!: this package is now considered as deprecated in GtkAda 2.x. You should use the types and subprograms in the Pango hierarchy, which correctly support internationalization, right-to-left writings, easy resizing of fonts, truetype fonts....

For backward compatibility, a new subprogram From_Description has been added to this package, which gives access to the more advanced font handling.

This is the base package for handling fonts. GtkAda knows about bitmap and vectorial fonts, and can draw both. The list of fonts available to you depends on what is installed on your system.

The name of the font is indicated in the standard X11 fashion, namely: (example extracted from the Xlib manual):

-adobe-courier-bold-o-normal–10-100-75-75-m-60-iso8859-1 where: @itemize @bullet @item adobe : foundry @item courier : font family @item bold : weight (e.g. bold, medium) @item o : slant (e.g. roman, italic, oblique) @item normal : set width (e.g. normal, condensed, narrow, double) @item 10 : pixels @item 100 : points (in tenths of a point) @item 75 : horizontal resolution in dpi @item 75 : vertical resolution in dpi @item m : spacing (e.g. monospace or proportional) @item 60 : average width (in tenths of a pixel) @item iso8859-1 : character set

@end itemize Any of the fields can have a '*' instead, so that the system will automatically find a font that matches the rest of the string, and won't care about that specific field.

An easy way to select a font is by using some external programs, for instance xfontsel, xlsfont, gfontsel, or even the font selection dialog example in the testgtk/ directory of the GtkAda distribution.

But the easiest way to create a font is to use a Pango_Font_Description. See package Pango.Font for more details about this structure.

Some of the functions below should be used only for wide-character strings. This is needed for languages with more than 256 characters.

Wide character values between 0 and 127 are always identical in meaning to the ASCII character codes. An alternative to wide characters is multi-byte characters, which extend normal char strings to cope with larger character sets. As the name suggests, multi-byte characters use a different number of bytes to store different character codes. For example codes 0-127 (i.e. the ASCII codes) often use just one byte of memory, while other codes may use 2, 3 or even 4 bytes. Multi-byte characters have the advantage that they can often be used in an application with little change, since strings are still represented as arrays of char values. However multi-byte strings are much easier to manipulate since the character are all of the same size.

On Unix systems, the external utility 'xfd' can be used to display all the characters in a font.

41.1 Types

subtype Gdk_Font **is** Gdk.Gdk_Font;

A font used to draw text. This can represent a bitmap font, a scalable (vectorial) font, or a fontset. A fontset is a list of comma-separated fonts, that permits GtkAda to obtain the

fonts needed for a variety of locales from a single locale-independent base font name. The single base font name should name a family of fonts whose members are encoded in the various charsets needed by the locales of interest. The algorithm used to select the font is described in the manual page for XCreateFontSet(3X).

41.2 Subprograms

```
function Get_Type                return Glib.GType;
```

Return the internal value associated with Gdk_Font.

```
procedure Load
  (Font                : out   Gdk_Font;
   Font_Name           :       String);
```

Load a new font, given its name.
This is the first step before using a font. The font is first looked up in the cache, and if it was already loaded, it is not reloaded again. Thus, it does not harm to call this function multiple times with the same Font_Name. Null_Font is returned if the font could not be loaded.

See From_Description below for another way of creating a Gdk_Font.

```
procedure Fontset_Load
  (Font                : out   Gdk_Font;
   Fontset_Name        :       String);
```

Load a new font set.
Fontset_Name is a comma-separated list of fonts that will be loaded as part of the fontset.

```
function From_Description
  (Font_Desc           :       Pango.Font.Pango_Font_Description)
   return Gdk.Font.Gdk_Font;
```

Create a new Gdk_Font from the given Pango_Font_Description.
This is a convenient function to create fonts from, because a Pango_Font_Description is a higher level description of a font attributes.

```
procedure Ref
  (Font                :       Gdk_Font);
```

Increment the reference counter for the font.
You should not make any assumption of the initial value of the fonts returned by Load or Fontset_Load, since these can be extracted from a cache.

```
procedure Unref
  (Font                :       Gdk_Font);
```

Decrement the reference counter for the font.
When this counter reaches 0, the font is deleted from memory.

```
function Id
  (Font                :       Gdk_Font)
   return Gint;
```

Return the X font id for the font.
This Id will only be needed if you want to call directly X11 functions, you won't need it with GtkAda.

```
function Equal
  (Fonta, Fontb        :       Gdk_Font)
```

```
   return Boolean;
```

Compare two fonts or two fontsets for equality.
Two fonts are equal if they have the same font Id. Two fontsets are equal if the name given to Fontset_Load was the same.

```
function Get_Ascent
  (Font               :      Gdk_Font)
   return Gint;
```

Return the maximal ascent for the font.
This is the logical extent above the baseline for spacing between two lines.

```
function Get_Descent
  (Font               :      Gdk_Font)
   return Gint;
```

Return the maximal descent for the font.
This is the logical extent below the baseline for spacing between two lines.

```
function String_Width
  (Font               :      Gdk_Font;
   Str                :      String)
   return Gint;
```

Return the width in pixels that Str will occupy if drawn with Font.
The value returned is the distance between the origin of the text and the position at which the next string should be drawn.

```
function String_Width
  (Font               :      Gdk_Font;
   Text               :      Gdk.Types.Gdk_WString)
   return Gint;
```

Return the width in pixels that Text will occupy on the screen.
This function should be used with strings that contain Unicode characters

```
function Char_Width
  (Font               :      Gdk_Font;
   Char               :      Character)
   return Gint;
```

Return the width in pixels occupied by a single character on the screen.
The value returned is the distance between Char's origin on the screen and the origin of the next character in the string.

```
function Char_Width
  (Font               :      Gdk_Font;
   Char               :      Gdk.Types.Gdk_WChar)
   return Gint;
```

Return the width in pixels occupied by a single wide-character.

```
function String_Measure
  (Font               :      Gdk_Font;
   Str                :      String)
   return Gint;
```

Determine the distance from the origin to the rightmost portion of Str.
This is not the correct value for determining the origin of the next portion when drawing text in multiple pieces. See String_Width instead.

```
function Char_Measure
  (Font               :      Gdk_Font;
   Char               :      Character)
```

```
   return Gint;
```

Return the width in pixels of Char.
As opposed to Char_Width, the value returned is not the distance at which the next character should be drawn. This is also called the right bearing of the character.

```
function String_Height
  (Font                :      Gdk_Font;
   Str                 :      String)
   return Gint;
```

Return the height in pixels of the string.
This is the total height, and you can not easily tell how this height is split around the baseline.

```
function Char_Height
  (Font                :      Gdk_Font;
   Char                :      Character)
   return Gint;
```

Return the total height in pixels of a single character.

```
procedure String_Extents
  (Font                :      Gdk.Font.Gdk_Font;
   Str                 :      String;
   Lbearing            : out  Gint;
   Rbearing            : out  Gint;
   Width               : out  Gint;
   Ascent              : out  Gint;
   Descent             : out  Gint);
```

Return the metrics for a given text.
See the picture for more explanations on all the fields. Lbearing : Origin to left edge of character. Rbearing : Origin to right edge of character. Width : Advance to next character's origin. Ascent : Baseline to top edge of character. Descent : Baseline to bottom edge of character.

```
procedure String_Extents
  (Font                :      Gdk_Font;
   Text                :      Gdk.Types.Gdk_WString;
   Lbearing            : out  Gint;
   Rbearing            : out  Gint;
   Width               : out  Gint;
   Ascent              : out  Gint;
   Descent             : out  Gint);
```

Return all the metrics for a given wide-character string.
See the picture for more explanations on the returned values.

42 Package Gdk.GC

A graphic context is a structure that describes all the attributes used by the drawing functions in Gdk. The colors, line styles, Fill styles and so on are defined through this structure.

On X11 systems, this structure is stored directly on the XServer, which speeds up the transfer of the drawing attributes a lot. Instead of transferring all of them every time you call one of the drawing functions, you simply specify which GC you want to use.

Thus, it is recommended to create as many GCs as you need, instead of creating a single one that is modified every time you need to modify one of the attributes.

On Unix machines, you should have a look at the external utility 'xgc' which demonstrates all the basic settings of the graphic contexts.

42.1 Types

type Gdk_Cap_Style **is**
(Cap_Not_Last, Cap_Butt, Cap_Round, Cap_Projecting);

type Gdk_Fill **is**
(Solid, Tiled, Stippled, Opaque_Stippled);

type Gdk_Function **is**
(Copy,
Invert,
Gdk_Xor,
Clear,
Gdk_And,
And_Reverse,
And_Invert,
Noop,
Gdk_Or,
Equiv,
Or_Reverse,
Copy_Invert,
Or_Invert,
Nand,
Set);

subtype Gdk_GC **is** Gdk.Gdk_GC;

A graphic context that contain all the information to draw graphics on the screen. Creating these GC is more efficient than passing a lot of parameters to each of the drawing functions, since these GC are stored on the server side and do not need to be pass through the network.

type Gdk_GC_Values **is new** Gdk.C_Proxy;

A structure used on the client side to store the same information as the GC. Creating a GC from this structure is more efficient than calling a lot of functions to modify the GC directly, since there is a single call to the server.

type Gdk_GC_Values_Mask **is mod** 2 ** 32;

type Gdk_Join_Style **is**
(Join_Miter, Join_Round, Join_Bevel);

type Gdk_Line_Style **is**
(Line_Solid, Line_On_Off_Dash, Line_Double_Dash);

type Gdk_Subwindow_Mode **is**
(Clip_By_Children, Include_Inferiors);

42.2 Subprograms

42.2.1 Gdk_GC

```
procedure Gdk_New
  (GC                 : out   Gdk_GC;
   Drawable           :       Gdk.Gdk_Drawable);
```

Create a new graphic context.
The window must have been realized first (so that it is associated with some resources on the Xserver). The GC can then be used for any window that has the same root window, and same color depth as Window. See the manual page for XCreateGC on Unix systems for more information.

```
procedure Gdk_New
  (GC                 : out   Gdk_GC;
   Drawable           :       Gdk.Gdk_Drawable;
   Values             :       Gdk_GC_Values;
   Values_Mask        :       Gdk_GC_Values_Mask);
```

Create a new graphic context.
It is directly created with the values set in Values, and whose associated field has been set in Values_Mask. This is faster than calling the simple Gdk_New function and each of other functions in this package, since each of them requires a call to the server.

```
function Get_Type                return Glib.GType;
```

Return the internal value associated with Gdk_GC.

```
procedure Destroy
  (GC                 :      Gdk_GC);
```

Free the memory allocated on the server for the graphic context.
Graphic contexts are never freed automatically by GtkAda, this is the user responsibility to do so. This procedure is deprecated. Use Unref instead.

```
procedure Ref
  (GC                 :      Gdk_GC);
```

Increment the reference counting for the graphic context.

```
procedure Unref
  (GC                 :      Gdk_GC);
```

Decrement the reference counting for the graphic context.
When this reaches 0, the graphic context is destroyed.

```
procedure Get_Values
  (GC                 :      Gdk_GC;
   Values             :      Gdk_GC_Values);
```

Get the values set in the GC.
This copies the values from the server to client, allowing faster modifications. Values can then be copied back to the server by creating a new graphic context with the function Gdk_New above. Values should have been allocated first with a call to Gdk_New.

```
procedure Set_Values
  (GC                 :      Gdk_GC;
   Values             :      Gdk_GC_Values;
   Mask               :      Gdk_GC_Values_Mask);
```

Set the values in the GC.
Mask indicates which values should be taken from Values and set in GC.

```
procedure Set_Foreground
  (GC                 :      Gdk_GC;
   Color              :      Gdk.Color.Gdk_Color);
```

Set the foreground color for the graphic context.
This color is the one that is used by most drawing functions.

```
procedure Set_Background
  (GC                 :      Gdk_GC;
   Color              :      Gdk.Color.Gdk_Color);
```

Set the background color for the graphic context.

```
procedure Set_Font
  (GC                 :      Gdk_GC;
   Font               :      Gdk.Font.Gdk_Font);
```

Set the font used by the graphic context.
This font is used by the function Gdk.Drawable.Draw_Text.

```
procedure Set_Function
  (GC                 :      Gdk_GC;
   Func               :      Gdk_Function);
```

Set the function in the graphic context.
This function specifies how the points are put on the screen, ie if GtkAda how GtkAda should mix the point already on the screen and the new point being put. Note that setting the function to Gdk_Xor is not the right way to do animation. You should instead save the background pixmap, put the image, and then restore the background.

In general, there are three basic steps to drawing: reading the source pixels, reading the destination pixels, and writing the destination pixels. Some functions only perform the third step (Set and Clear), some do not need the middle step (Copy), whereas most require the three steps, and thus can be much slower.

```
procedure Set_Fill
  (GC                  :      Gdk_GC;
   Fill                :      Gdk_Fill);
```

Set the pattern used for filling the polygons.

```
procedure Set_Tile
  (GC                  :      Gdk_GC;
   Tile                :      Gdk.Gdk_Pixmap);
procedure Set_Stipple
  (GC                  :      Gdk_GC;
   Stipple             :      Gdk.Gdk_Pixmap);
procedure Set_Ts_Origin
  (GC                  :      Gdk_GC;
   X, Y                :      Gint);
```

Set the Tile and Stipple origin in the graphic context.

```
procedure Set_Clip_Origin
  (GC                  :      Gdk_GC;
   X, Y                :      Gint);
```

Set the origin of the clip mask.
See the functions Set_Clip_Rectangle, Set_Clip_Region and Gdk.Bitmap.Set_Clip_Mask for more explanation.

```
procedure Set_Clip_Mask
  (GC                  :      Gdk.GC.Gdk_GC;
   Mask                :      Gdk.Gdk_Bitmap);
```

If Mask is set to Null_Bitmap, then no clip_mask is used for drawing.
Points will be drawn through this GC only where the bits are set to 1 in the mask. See also the function Set_Clip_Origin for how to move the mask inside the GC.

```
procedure Set_Clip_Rectangle
  (GC                  :      Gdk_GC;
   Rectangle           :      Gdk.Rectangle.Gdk_Rectangle);
procedure Set_Clip_Rectangle
  (GC                  :      Gdk_GC;
   Rectangle           :      Gdk.Rectangle.Gdk_Rectangle_Access
                        := null);
```

Set the clip rectangle.
Only the points that are drawn inside this rectangle will be displayed on the screen. The clip origin is modified automatically. See Set_Clip_Mask to delete the current clip mask.

```
procedure Set_Clip_Region
  (GC                  :      Gdk_GC;
   Region              :      Gdk.Region.Gdk_Region);
```

Define a clip region on the screen.
This is just like Set_Clip_Rectangle, except that a region is a more complex region, that can be the intersection or union of multiple rectangles. Note that the Clip_Origin can have an influence on this function.

```
procedure Set_Subwindow
  (GC                  :      Gdk_GC;
   Mode                :      Gdk_Subwindow_Mode);
```

Set the subwindow mode for the graphic context.
This specifies whether the drawing routines should be clipped to the specific window they are drawn into, or if they should extend to subwindows as well.

```
procedure Set_Exposures
  (GC                 :      Gdk_GC;
   Exposures          :      Boolean);
```

Exposures indicates whether you want "expose" and "noexpose" events to be reported when calling Copy_Area and Copy_Plane with this GC. You should disable this if you don't need the event and want to optimize your application. If Exposures is True, then any call to Copy_Area or Draw_Pixmap will generate an expose event. Otherwise, these will generate a no_expose event.

```
procedure Set_Line_Attributes
  (GC                 :      Gdk_GC;
   Line_Width         :      Gint;
   Line_Style         :      Gdk_Line_Style;
   Cap_Style          :      Gdk_Cap_Style;
   Join_Style         :      Gdk_Join_Style);
```

Set the line attributes for this GC.
Line_Width is the width of the line. If its value is 0, the line is as thin as possible, possibly even more so than if the width is 1. It is also faster to draw a line with width 0 than any other line width.

Line_Style specifies whether the line should be solid or dashed. With Line_On_Off_Dash, the colors are alternatively the foreground color, and blank. With Line_Double_Dash, the colors are alternatively the foreground and background colors.

Cap_Style specifies how the line should end, either flat or rounded.

Join_Style specifies how two consecutive lines drawn by Draw_Lines are connected.

```
procedure Set_Dashes
  (Gc                 :      Gdk_GC;
   Dash_Offset        :      Gint;
   Dash_List          :      Guchar_Array);
```

Specify the dash pattern when the line's style is anything but solid.
The values in the array alternatively give the length (in pixels) of the plain dash, the empty dash, the second plain dash, ... None of these values can be 0. If there is an odd number of items in Dash_List, this is equivalent to giving the array concatenated with itself. Dash_Offset specifies the phase of the pattern to start with.

```
procedure Copy
  (Dst_GC             :      Gdk_GC;
   Src_GC             :      Gdk_GC);
```

Copy a Src_GC to Dst_GC.

```
procedure Set_Colormap
  (Gc                 :      Gdk_GC;
   Colormap           :      Gdk.Gdk_Colormap);

function Get_Colormap
  (Gc                 :      Gdk_GC)
   return Gdk.Gdk_Colormap;

procedure Set_Rgb_Fg_Color
  (Gc                 :      Gdk_GC;
   Color              :      Gdk.Color.Gdk_Color);
```

```
procedure Set_Rgb_Bg_Color
  (Gc                 :      Gdk_GC;
   Color              :      Gdk.Color.Gdk_Color);
```

42.2.2 Gdk_Color_Values

```
function Gdk_New                return Gdk_GC_Values;
```

Allocate a new Values structure on the client.
Note that this function allocates a C structure, and thus needs to be freed with a call to Free below.

```
procedure Free
  (Values             :      Gdk_GC_Values);
```

Free the C structure associated with Values.

```
procedure Set_Foreground
  (Values             :      Gdk_GC_Values;
   Color              :      Gdk.Color.Gdk_Color);
```

Same as Set_Foreground, but on the client side

```
procedure Set_Background
  (Values             :      Gdk_GC_Values;
   Color              :      Gdk.Color.Gdk_Color);
```

Same as Set_Background, but on the client side

```
procedure Set_Font
  (Values             :      Gdk_GC_Values;
   Font               :      Gdk.Font.Gdk_Font);
```

Same as Set_Font, but on the client side

```
procedure Set_Function
  (Values             :      Gdk_GC_Values;
   Func               :      Gdk_Function);
```

Same as Set_Function, but on the client side

```
procedure Set_Fill
  (Values             :      Gdk_GC_Values;
   Fill               :      Gdk_Fill);
```

Same as Set_Fill, but on the client side

```
procedure Set_Ts_Origin
  (Values             :      Gdk_GC_Values;
   X, Y               :      Gint);
```

Same as Set_Ts_Origin, but on the client side

```
procedure Set_Clip_Origin
  (Values             :      Gdk_GC_Values;
   X, Y               :      Gint);
```

Same as Set_Clip_Origin, but on the client side

```
procedure Set_Subwindow
  (Values             :      Gdk_GC_Values;
   Mode               :      Gdk_Subwindow_Mode);
```

Same as Set_Subwindow, but on the client side

```
procedure Set_Exposures
  (Values             :      Gdk_GC_Values;
   Exposures          :      Boolean);
```

Same as Set_Exposures, but on the client side

```
procedure Set_Line_Attributes
  (Values             :      Gdk_GC_Values;
   Line_Width         :      Gint;
   Line_Style         :      Gdk_Line_Style;
   Cap_Style          :      Gdk_Cap_Style;
   Join_Style         :      Gdk_Join_Style);
```

Same as Set_Line_Attributes, but on the client side

43 Package Gdk.Main

This package provides routines to handle initialization and set up of the Gdk library.

43.1 Types

```
type Gdk_Grab_Status is
          (Grab_Success,
           Grab_Already_Grabbed,
           Gdk_Grab_Invalid_Time,
           Gdk_Grab_Not_Viewable,
           Gdk_Grab_Frozen);
```

43.2 Subprograms

```
procedure Init;
```

Initialize the library for use.
The command line arguments are modified to reflect any arguments which were not handled. (Such arguments should either be handled by the application or dismissed).

```
procedure Gdk_Exit
  (Error_Code         :      Gint);
```

Restore the library to an un-itialized state and exits
the program using the "exit" system call. Error_Code is the error value to pass to "exit". Allocated structures are freed and the program exits cleanly. This function is deprecated.

```
function Set_Locale             return String;
```

Initialize handling of internationalization of strings.
See Gtkada.Intl for more details.

```
procedure Set_Locale;
```

Drops the string returned by the Set_Locale function;

```
procedure Set_Use_Xshm
  (Use_Xshm           :      Boolean := True);
```

Set whether shared memory (when supported by the graphic server) should
be used.

```
function Get_Use_Xshm           return Boolean;
```

Return whether shared memory on the graphic server is used.

```
function Get_Display            return String;
```

Return the name of the display.

```
function Pointer_Grab
  (Window             :      Gdk.Window.Gdk_Window;
   Owner_Events       :      Boolean := True;
   Event_Mask         :      Gdk.Event.Gdk_Event_Mask;
   Confine_To         :      Gdk.Window.Gdk_Window
                      := Gdk.Window.Null_Window;
   Cursor             :      Gdk.Cursor.Gdk_Cursor
                      := Gdk.Cursor.Null_Cursor;
   Time               :      Guint32 := 0)
```

```
    return Gdk_Grab_Status;
```

Grab the pointer to a specific window.

- Window is the window which will receive the grab
- Owner_Events specifies whether events will be reported as is, or relative to Window
- Event_Mask masks only interesting events
- Confine_To limits the cursor movement to the specified window
- Cursor changes the cursor for the duration of the grab
- Time specifies the time Requires a corresponding call to Pointer_Ungrab

```
  procedure Pointer_Ungrab
    (Time               :      Guint32 := 0);
```

Release any pointer grab.

```
  function Pointer_Is_Grabbed    return Boolean;
```

Tell wether there is an active pointer grab in effect.

```
  function Keyboard_Grab
    (Window             :      Gdk.Window.Gdk_Window;
     Owner_Events       :      Boolean := True;
     Time               :      Guint32 := 0)
     return Gdk_Grab_Status;
```

Grab the keyboard to a specific window.

- Window is the window which will receive the grab
- Owner_Events specifies whether events will be reported as is, or relative to Window
- Time specifies the time Requires a corresponding call to Keyboard_Ungrab

```
  procedure Keyboard_Ungrab
    (Time               :      Guint32 := 0);
```

Release any keyboard grab.

```
  function Screen_Width          return Gint;
```

Return the width of the screen.

```
  function Screen_Height         return Gint;
```

Return the height of the screen.

```
  function Screen_Width_MM       return Gint;
```

Return the width of the screen in millimeters.

```
  function Screen_Height_MM      return Gint;
```

Return the height of the screen in millimeters.

```
  procedure Flush;
```

Flush the queue of graphic events and then wait
until all requests have been received and processed.

```
  procedure Beep;
```

Emit a beep.

```
  procedure Set_Double_Click_Time
    (Msec               :      Guint);
```

44 Package Gdk.Pixbuf

This object provides image manipulation routines.

The following image formats are known, but some depend on external libraries for the proper loading of files (indicated with * in the list): PNG*, JPEG*, TIFF*, GIF, XPM, PNM, Sun raster file (ras), ico, bmp.

With this package, you can load images from file, display them on the screen, re-scale them and compose them with other images. All the functions fully support alpha channels (opacity).

Different filters are provided, depending on the quality of output you expect and the speed you need.

44.1 Types

```
type Alpha_Mode is
          (Alpha_Bilevel, Alpha_Full);
```

Alpha compositing mode. This indicates how the alpha channel (for opacity) is handled when rendering. pragma Convention (C, Alpha_Mode);

```
type Alpha_Range is range 0 .. 255;
```

Valid values for alpha parameters. pragma Convention (C, Alpha_Range);

```
type File_Format is
          (JPEG, PNG, ICO, BMP);
```

Possible formats when saving a file.

```
type Gdk_Colorspace is
          (Colorspace_RGB);
```

Type of the image. The only possible value is currently RGB, but extensions will exist with CMYK, Gray, Lab, ... pragma Convention (C, Gdk_Colorspace);

```
type Gdk_Interp_Type is
          (Interp_Nearest,
             -- Nearest neighbor. It is the fastest and lowest quality.

           Interp_Tiles,
             -- Accurate simulation of the Postscript image operator
             -- without any interpolation enabled; each pixel is rendered as a tiny
             -- parallelogram of solid color, the edges of which are implemented
             -- with anti-aliasing. It resembles nearest neighbor for enlargement,
             -- and bilinear for reduction.

           Interp_Bilinear,
             -- Bilinear interpolation. For enlargement, it is equivalent to
             -- point-sampling the ideal bilinear-interpolated image. For reduction,
             -- it is equivalent to laying down small tiles and integrating over the
             -- coverage area.
```

```
Interp_Hyper
   -- Filter_Hyper is the highest quality reconstruction function. It is
   -- derived from the hyperbolic filters in Wolberg's "Digital Image
   -- Warping," and is formally defined as the hyperbolic-filter sampling
   -- the ideal hyperbolic-filter interpolated image (the filter is
   -- designed to be idempotent for 1:1 pixel mapping). It is the slowest
```

and highest quality.); Interpolation methods. pragma Convention (C, Gdk_Interp_Type);

type Gdk_Pixbuf_Animation **is new** Glib.C_Proxy;

Type used for animations.

type Gdk_Pixbuf_Animation_Iter **is new** Glib.C_Proxy;

Type used to iterate through an animation.

type Image_Quality **is range** 0 .. 100;

For a JPEG image only, quality of the image in percentage.

44.2 Subprograms

44.2.1 Get_Type

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gdk_Pixbuf.

44.2.2 Accessing the fields

```
function Get_Colorspace
  (Pixbuf             :      Gdk_Pixbuf)
   return Gdk_Colorspace;
```

Query the color space of a pixbuf.

```
function Get_N_Channels
  (Pixbuf             :      Gdk_Pixbuf)
   return Gint;
```

Number of channels in the image.

```
function Get_Has_Alpha
  (Pixbuf             :      Gdk_Pixbuf)
   return Boolean;
```

Return True if the image has an alpha channel (opacity information).

```
function Get_Bits_Per_Sample
  (Pixbuf             :      Gdk_Pixbuf)
   return Gint;
```

Number of bits per color sample.

```
function Get_Pixels
  (Pixbuf             :      Gdk_Pixbuf)
   return Gdk.Rgb.Rgb_Buffer_Access;
```

Return a pointer to the pixel data of the image.

```
function Get_Width
  (Pixbuf             :      Gdk_Pixbuf)
   return Gint;
```

Return the width of the image in pixels.

```
function Get_Height
  (Pixbuf             :      Gdk_Pixbuf)
   return Gint;
```

Return the height of the image in pixels.

```
function Get_Rowstride
  (Pixbuf             :      Gdk_Pixbuf)
   return Gint;
```

Return the number of bytes between rows in the image data.

44.2.3 Creating

```
function Gdk_New
  (Colorspace         :      Gdk_Colorspace
                      := Colorspace_RGB;
   Has_Alpha          :      Boolean := False;
   Bits_Per_Sample    :      Gint := 8;
   Width              :      Gint;
   Height             :      Gint)
   return Gdk_Pixbuf;
```

Create a blank pixbuf with an optimal row stride and a new buffer.
The buffer is allocated, but not cleared. The reference counting is initialized to 1.

```
function Copy
  (Pixbuf             :      Gdk_Pixbuf)
   return Gdk_Pixbuf;
```

Copy a pixbuf.

```
function Gdk_New_Subpixbuf
  (Src_Pixbuf         :      Gdk_Pixbuf;
   Src_X              :      Gint;
   Src_Y              :      Gint;
   Width              :      Gint;
   Height             :      Gint)
   return Gdk_Pixbuf;
```

Create a pixbuf which points to the pixels of another pixbuf

```
procedure Gdk_New_From_File
  (Pixbuf             : out   Gdk_Pixbuf;
   Filename           :      String;
   Error              : out   GError);
```

Load an image from file.

```
function Gdk_New_From_Data
  (Data               :      Guchar_Array_Access;
   Colorspace         :      Gdk_Colorspace
                      := Colorspace_RGB;
   Has_Alpha          :      Boolean := False;
   Bits_Per_Sample    :      Gint := 8;
   Width              :      Gint;
   Height             :      Gint;
   Rowstride          :      Gint;
   Auto_Destroy_Data  :      Boolean := True)
   return Gdk_Pixbuf;
```

Create a pixbuf out of in-memory image data.
Currently only RGB images with 8 bits per sample are supported. Width and Height must be > 0. Rowstride is the distance in bytes between row starts. A typical value is 4*Width when there is an Alpha channel. If Auto_Destroy_Data is true, passed data will be automatically freed when the reference count of the pixbuf reaches 1. Otherwise, data is never freed.

```
function Gdk_New_From_Xpm_Data
  (Data               :      Interfaces.C.Strings.chars_ptr_array)
   return Gdk_Pixbuf;
```

Create an image from a XPM data.

```
procedure Fill
  (Pixbuf             :      Gdk_Pixbuf;
   Pixel              :      Guint32);
```

Fill pixbuf with a given pixel value.

```
procedure Save
  (Pixbuf             :      Gdk_Pixbuf;
   Filename           :      String;
   Format             :      File_Format;
   Error              : out   GError;
   Quality            :      Image_Quality
                      := Image_Quality'Last;
   Depth              :      Integer := 32);
```

Save pixbuf to a file.
Quality is only taken into account for JPEG images. Depth is only taken into account for ICO images and can take the values 16, 24 or 32. Error is set to null on success, and set to a GError otherwise.

```
function Add_Alpha
  (Pixbuf             :      Gdk_Pixbuf;
   Substitute_Color   :      Boolean := False;
   Red                :      Guchar := 0;
   Green              :      Guchar := 0;
   Blue               :      Guchar := 0)
   return Gdk_Pixbuf;
```

Add an alpha channel.
Return a newly allocated image copied from Pixbuf, but with an extra alpha channel. If Pixbuf already had an alpha channel, the two images have exactly the same contents. If Substitute_Color is True, the color (Red, Green, Blue) is substituted for zero opacity. If Substitute_Color is False, Red, Green and Blue are ignored, and a new color is created with zero opacity.

```
procedure Copy_Area
  (Src_Pixbuf         :      Gdk_Pixbuf;
   Src_X              :      Gint;
   Src_Y              :      Gint;
   Width              :      Gint;
   Height             :      Gint;
   Dest_Pixbuf        :      Gdk_Pixbuf;
   Dest_X             :      Gint;
   Dest_Y             :      Gint);
```

Copy a rectangular area from Src_pixbuf to Dest_pixbuf.
Conversion of pixbuf formats is done automatically.

```
procedure Saturate_And_Pixelate
  (Src                 :      Gdk_Pixbuf;
   Dest                :      Gdk_Pixbuf;
   Saturation          :      Gfloat;
   Pixelate            :      Boolean := True);
```

Brighten/darken and optionally make it pixelated-looking.

44.2.4 Rendering

```
procedure Render_Threshold_Alpha
  (Pixbuf              :      Gdk_Pixbuf;
   Bitmap              :      Gdk.Bitmap.Gdk_Bitmap;
   Src_X               :      Gint;
   Src_Y               :      Gint;
   Dest_X              :      Gint;
   Dest_Y              :      Gint;
   Width               :      Gint;
   Height              :      Gint;
   Alpha_Threshold     :      Alpha_Range);
```

Take the opacity values in a rectangular portion of a pixbuf and
thresholds them to produce a bi-level alpha mask that can be used as a clipping mask for a drawable. Bitmap is the bitmap where the bilevel mask will be painted to. Alpha_Threshold are the opacity values below which a pixel will be painted as zero. All other values will be painted as one.

```
procedure Render_To_Drawable
  (Pixbuf              :      Gdk_Pixbuf;
   Drawable            :      Gdk.Drawable.Gdk_Drawable;
   GC                  :      Gdk.GC.Gdk_GC;
   Src_X               :      Gint;
   Src_Y               :      Gint;
   Dest_X              :      Gint;
   Dest_Y              :      Gint;
   Width               :      Gint;
   Height              :      Gint;
   Dither              :      Gdk.Rgb.Gdk_Rgb_Dither
                       := Gdk.Rgb.Dither_Normal;
   X_Dither            :      Gint := 0;
   Y_Dither            :      Gint := 0);
```

Render a rectangular portion of a pixbuf to a drawable while using the
specified GC. This is done using Gdk.RGB, so the specified drawable must have the Gdk.RGB visual and colormap. Note that this function will ignore the opacity information for images with an alpha channel; the GC must already have the clipping mask set if you want transparent regions to show through.

For an explanation of dither offsets, see the Gdk.RGB documentation. In brief, the dither offset is important when re-rendering partial regions of an image to a rendered version of the full image, or for when the offsets to a base position change, as in scrolling. The dither matrix has to be shifted for consistent visual results. If you do not have any of these cases, the dither offsets can be both zero.

```
procedure Render_To_Drawable_Alpha
  (Pixbuf              :      Gdk_Pixbuf;
   Drawable            :      Gdk.Drawable.Gdk_Drawable;
   Src_X               :      Gint;
   Src_Y               :      Gint;
```

```
   Dest_X             :      Gint;
   Dest_Y             :      Gint;
   Width              :      Gint;
   Height             :      Gint;
   Alpha              :      Alpha_Mode;
   Alpha_Threshold    :      Alpha_Range;
   Dither             :      Gdk.Rgb.Gdk_Rgb_Dither
                      := Gdk.Rgb.Dither_Normal;
   X_Dither           :      Gint := 0;
   Y_Dither           :      Gint := 0);
```

Render a rectangular portion of a pixbuf to a drawable.
This is done using Gdk.RGB, so the specified drawable must have the Gdk_RGB visual and colormap. When used with Alpha_Bilevel, this function has to create a bitmap out of the thresholded alpha channel of the image and, it has to set this bitmap as the clipping mask for the GC used for drawing. This can be a significant performance penalty depending on the size and the complexity of the alpha channel of the image. If performance is crucial, consider handling the alpha channel yourself (possibly by caching it in your application) and using Render_To_Drawable or Gdk.RGB directly instead.

If the image does have opacity information and Alpha_Mode is Alpha_Bilevel, specifies the threshold value for opacity values

```
procedure Render_Pixmap_And_Mask
  (Pixbuf             :      Gdk_Pixbuf;
   Pixmap             : out  Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out  Gdk.Bitmap.Gdk_Bitmap;
   Alpha_Threshold    :      Alpha_Range);

procedure Render_Pixmap_And_Mask_For_Colormap
  (Pixbuf             :      Gdk_Pixbuf;
   Colormap           :      Gdk.Color.Gdk_Colormap;
   Pixmap             : out  Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out  Gdk.Bitmap.Gdk_Bitmap;
   Alpha_Threshold    :      Alpha_Range);
```

Creates a pixmap and a mask bitmap which are returned in the Pixmap
and Mask arguments, respectively, and renders a pixbuf and its corresponding tresholded alpha mask to them. This is merely a convenience function; applications that need to render pixbufs with dither offsets or to given drawables should use Render_To_Drawable_Alpha or Render_To_Drawable The pixmap that is created uses Colormap. This colormap must match the colormap of the window where the pixmap will eventually be used or an error will result.

```
function Get_From_Drawable
  (Dest               :      Gdk_Pixbuf;
   Src                :      Gdk.Drawable.Gdk_Drawable;
   Cmap               :      Gdk.Color.Gdk_Colormap;
   Src_X              :      Gint;
   Src_Y              :      Gint;
   Dest_X             :      Gint;
   Dest_Y             :      Gint;
   Width              :      Gint;
   Height             :      Gint)
   return Gdk_Pixbuf;
```

Transfer image data from a Gdk drawable and converts it to an RGB(A)
representation inside a Gdk_Pixbuf.

If the drawable src is a pixmap, then a suitable colormap must be specified, since pixmaps are just blocks of pixel data without an associated colormap. If the drawable is a window, the Cmap argument will be ignored and the window's own colormap will be used instead.

If the specified destination pixbuf Dest is Null_Pixbuf, then this function will create an RGB pixbuf with 8 bits per channel and no alpha, with the same size specified by the Width and Height arguments. In this case, the Dest_x and Dest_y arguments must be specified as 0, otherwise the function will return Null_Pixbuf. If the specified destination pixbuf is not Null_Pixbuf and it contains alpha information, then the filled pixels will be set to full opacity.

If the specified drawable is a pixmap, then the requested source rectangle must be completely contained within the pixmap, otherwise the function will return Null_Pixbuf.

If the specified drawable is a window, then it must be viewable, i.e. all of its ancestors up to the root window must be mapped. Also, the specified source rectangle must be completely contained within the window and within the screen. If regions of the window are obscured by non-inferior windows, the contents of those regions are undefined. The contents of regions obscured by inferior windows of a different depth than that of the source window will also be undefined.

Return value: The same pixbuf as Dest if it was non-NULL, or a newly-created pixbuf with a reference count of 1 if no destination pixbuf was specified.

44.2.5 Scaling

```
procedure Scale
  (Src                :      Gdk_Pixbuf;
   Dest               :      Gdk_Pixbuf;
   Dest_X             :      Gint;
   Dest_Y             :      Gint;
   Dest_Width         :      Gint;
   Dest_Height        :      Gint;
   Offset_X           :      Gdouble := 0.0;
   Offset_Y           :      Gdouble := 0.0;
   Scale_X            :      Gdouble := 1.0;
   Scale_Y            :      Gdouble := 1.0;
   Inter_Type         :      Gdk_Interp_Type
                      := Interp_Bilinear);
```

Transform the source image by scaling by Scale_x and Scale_y then
translating by Offset_x and Offset_y. The image is then rendered in the rectangle (Dest_x, Dest_y, Dest_width, Dest_height) of the resulting image onto the destination drawable replacing the previous contents.

```
procedure Composite
  (Src                :      Gdk_Pixbuf;
   Dest               :      Gdk_Pixbuf;
   Dest_X             :      Gint;
   Dest_Y             :      Gint;
   Dest_Width         :      Gint;
   Dest_Height        :      Gint;
   Offset_X           :      Gdouble := 0.0;
   Offset_Y           :      Gdouble := 0.0;
   Scale_X            :      Gdouble := 1.0;
   Scale_Y            :      Gdouble := 1.0;
   Inter_Type         :      Gdk_Interp_Type
                      := Interp_Bilinear;
```

```
   Overall_Alpha      :      Alpha_Range := 128);
```

Transform the source image by scaling by Scale_X and Scale_Y then translating by Offset_X and Offset_Y, then composite the rectangle (Dest_X, Dest_Y, Dest_Width, Dest_Height) of the resulting image onto the destination drawable.

```
procedure Composite_Color
  (Src                :      Gdk_Pixbuf;
   Dest               :      Gdk_Pixbuf;
   Dest_X             :      Gint;
   Dest_Y             :      Gint;
   Dest_Width         :      Gint;
   Dest_Height        :      Gint;
   Offset_X           :      Gdouble := 0.0;
   Offset_Y           :      Gdouble := 0.0;
   Scale_X            :      Gdouble := 1.0;
   Scale_Y            :      Gdouble := 1.0;
   Inter_Type         :      Gdk_Interp_Type
                      := Interp_Bilinear;
   Overall_Alpha      :      Alpha_Range := 128;
   Check_X            :      Gint := 0;
   Check_Y            :      Gint := 0;
   Check_Size         :      Gint := 0;
   Color1             :      Guint32 := 0;
   Color2             :      Guint32 := 0);
```

Transform the source image by scaling by Scale_x and Scale_y then translating by Offset_x and Offset_y, then composites the rectangle (Dest_X, Dest_Y, Dest_Width, Dest_Height) of the resulting image with a checkboard of the colors Color1 and Color2 and renders it onto the destination drawable. The origin of checkboard is at (Check_x, Check_y) Color1 is the color at the upper left of the check.

```
function Scale_Simple
  (Src                :      Gdk_Pixbuf;
   Dest_Width         :      Gint;
   Dest_Height        :      Gint;
   Inter_Type         :      Gdk_Interp_Type
                      := Interp_Bilinear)
   return Gdk_Pixbuf;
```

Scale the Src image to Dest_width x Dest_height and render the result into a new pixbuf.

```
function Composite_Color_Simple
  (Src                :      Gdk_Pixbuf;
   Dest_Width         :      Gint;
   Dest_Height        :      Gint;
   Inter_Type         :      Gdk_Interp_Type
                      := Interp_Bilinear;
   Overall_Alpha      :      Alpha_Range := 128;
   Color1             :      Guint32 := 0;
   Color2             :      Guint32 := 0)
   return Gdk_Pixbuf;
```

Scale Src to Dest_width x Dest_height and composite the result with a checkboard of colors Color1 and Color2 and render the result into a new pixbuf.

44.2.6 Animation support

```
function Get_Type_Animation    return Glib.GType;
```

Return the internal value associated with a Gdk_Pixbuf_Animation.

```
procedure Gdk_New_From_File
  (Animation          : out   Gdk_Pixbuf_Animation;
   Filename           :       String;
   Error              : out   GError);
```

Create a new animation by loading it from a file.
The file format is detected automatically. If the file's format does not support multi-frame images, then an animation with a single frame will be created. Possible errors are in the Pixbuf_Error and GFile_Error domains. On return, Animation is a newly created animation with a reference count of 1, or null if any of several error conditions ocurred: the file could not be opened, there was no loader for the file's format, there was not enough memory to allocate the image buffer, or the image file contained invalid data.

```
procedure Ref
  (Animation          :       Gdk_Pixbuf_Animation);
```

Increment the reference counting on the animation.

```
procedure Unref
  (Animation          :       Gdk_Pixbuf_Animation);
```

Decrement the reference counting on the animation.

```
function Get_Width
  (Animation          :       Gdk_Pixbuf_Animation)
   return Gint;
```

Return the width of the bounding box of a pixbuf animation.

```
function Get_Height
  (Animation          :       Gdk_Pixbuf_Animation)
   return Gint;
```

Return the height of the bounding box of a pixbuf animation.

```
function Is_Static_Image
  (Animation          :       Gdk_Pixbuf_Animation)
   return Boolean;
```

If you load a file with Gdk_New_From_File and it turns out to be a
plain, unanimated image, then this function will return True. Use Get_Static_Image to retrieve the image.

```
function Get_Static_Image
  (Animation          :       Gdk_Pixbuf_Animation)
   return Gdk_Pixbuf;
```

If an animation is really just a plain image (has only one frame),
this function returns that image. If the animation is an animation, this function returns a reasonable thing to display as a static unanimated image, which might be the first frame, or something more sophisticated. If an animation hasn't loaded any frames yet, this function will return null.

```
function Get_Iter
  (Animation          :       Gdk_Pixbuf_Animation;
   Start_Time         :       GTime_Val_Access := null)
   return Gdk_Pixbuf_Animation_Iter;
```

Get an iterator for displaying an animation. The iterator provides
the frames that should be displayed at a given time. It should be freed after use with Unref.

Start_Time would normally come from G_Get_Current_Time, and marks the beginning of animation playback. After creating an iterator, you should immediately display the pixbuf returned by Get_Pixbuf. Then, you should install a timeout (with Timeout_Add)

or by some other mechanism to ensure that you'll update the image after Get_Delay_Time milliseconds. Each time the image is updated, you should reinstall the timeout with the new, possibly-changed delay time.

As a shortcut, if Start_Time is equal to null, the result of G_Get_Current_Time will be used automatically.

To update the image (i.e. possibly change the result of Get_Pixbuf to a new frame of the animation), call Advance.

If you're using Gdk_Pixbuf_Loader, in addition to updating the image after the delay time, you should also update it whenever you receive the area_updated signal and On_Currently_Loading_Frame returns True. In this case, the frame currently being fed into the loader has received new data, so needs to be refreshed. The delay time for a frame may also be modified after an area_updated signal, for example if the delay time for a frame is encoded in the data after the frame itself. So your timeout should be reinstalled after any area_updated signal.

A delay time of -1 is possible, indicating "infinite."

44.2.7 Iterators

```
function Get_Type_Animation_Iterreturn Glib.GType;
```

Return the internal value associated with a Gdk_Pixbuf_Animation_Iter.

```
procedure Ref
  (Iter               :      Gdk_Pixbuf_Animation_Iter);
```

Increment the reference counting on the iterator.

```
procedure Unref
  (Iter               :      Gdk_Pixbuf_Animation_Iter);
```

Decrement the reference counting on the iterator.

```
function Get_Delay_Time
  (Iter               :      Gdk_Pixbuf_Animation_Iter)
   return Gint;
```

Return the number of milliseconds the current pixbuf should be displayed
or -1 if the current pixbuf should be displayed forever. Timeout_Add conveniently takes a timeout in milliseconds, so you can use a timeout to schedule the next update.

```
function Get_Pixbuf
  (Iter               :      Gdk_Pixbuf_Animation_Iter)
   return Gdk_Pixbuf;
```

Return the current pixbuf which should be displayed.
The pixbuf will be the same size as the animation itself (Get_Width, Get_Height). This pixbuf should be displayed for Get_Delay_Time milliseconds. The caller of this function does not own a reference to the returned pixbuf; the returned pixbuf will become invalid when the iterator advances to the next frame, which may happen anytime you call Advance. Copy the pixbuf to keep it (don't just add a reference), as it may get recycled as you advance the iterator.

```
function On_Currently_Loading_Frame
  (Iter               :      Gdk_Pixbuf_Animation_Iter)
   return Boolean;
```

Used to determine how to respond to the area_updated signal on
Gdk_Pixbuf_Loader when loading an animation. area_updated is emitted for an area of the

frame currently streaming in to the loader. So if you're on the currently loading frame, you need to redraw the screen for the updated area.

```
function Advance
  (Iter               :      Gdk_Pixbuf_Animation_Iter;
   Current_Timer      :      GTime_Val_Access := null)
   return Boolean;
```

Possibly advance an animation to a new frame.
Chooses the frame based on the start time passed to Get_Iter.

Current_Time would normally come from G_Get_Current_Time, and must be greater than or equal to the time passed to Get_Iter, and must increase or remain unchanged each time Get_Pixbuf is called. That is, you can't go backward in time; animations only play forward.

As a shortcut, pass null for the current time and G_Get_Current_Time will be invoked on your behalf. So you only need to explicitly pass Current_Time if you're doing something odd like playing the animation at double speed.

If this function returns False, there's no need to update the animation display, assuming the display had been rendered prior to advancing; if True, you need to call Get_Pixbuf and update the display with the new pixbuf.

44.2.8 Cursors

```
procedure Gdk_New_From_Pixbuf
  (Cursor             : out   Gdk.Cursor.Gdk_Cursor;
   Display            :      Gdk.Display.Gdk_Display
                      := Gdk.Display.Get_Default;
   Pixbuf             :      Gdk_Pixbuf;
   X                  :      Glib.Gint;
   Y                  :      Glib.Gint);
```

Create a cursor from a pixbuf.
Not all GDK backends support RGBA cursors. If they are not supported, a monochrome approximation will be displayed. The functions gdk.display.supports_cursor_alpha and gdk.display.supports_cursor_color can be used to determine whether RGBA cursors are supported; gdk.display.get_default_cursor_size and gdk.display.get_maximal_cursor_size give information about cursor sizes. On the X backend, support for RGBA cursors requires a sufficently new version of the X Render extension.

```
function Get_Image
  (Cursor             :      Gdk.Cursor.Gdk_Cursor)
   return Gdk_Pixbuf;
```

Return the image stored in the cursor

45 Package Gdk.Pixmap

Pixmaps are off-screen drawables. They can be drawn upon with the standard drawing primitives, then copied to another drawable (such as a Gdk_Window) with Gdk.Drawable.Draw_Drawable. The depth of a pixmap is the number of bits per pixels. Bitmaps are simply pixmaps with a depth of 1. (That is, they are monochrome bitmaps - each pixel can be either on or off). See Gdk.Bitmap for more details on bitmap handling.

45.1 Types

subtype Gdk_Pixmap **is** Gdk.Gdk_Pixmap;

A server-side image. You can create an empty pixmap, or load if from external files in bitmap and pixmap format. See Gdk.Pixbuf if you need to load images in other formats.

45.2 Subprograms

```
procedure Gdk_New
  (Pixmap             : out   Gdk_Pixmap;
   Window             :       Gdk.Window.Gdk_Window;
   Width              :       Gint;
   Height             :       Gint;
   Depth              :       Gint := -1);
```

Create a new pixmap with a given size.
Window is used to determine default values for the new pixmap. Can be eventually null. Width is the width of the new pixmap in pixels. Height is the height of the new pixmap in pixels. Depth is the depth (number of bits per pixel) of the new pixmap. If -1, and window is not null, the depth of the new pixmap will be equal to that of window. Automatically reference the pixmap once.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with Gdk_Pixmap.

```
procedure Ref
  (Pixmap             :       Gdk_Pixmap);
```

Add a reference to a pixmap.

```
procedure Unref
  (Pixmap             :       Gdk_Pixmap);
```

This is the usual way to destroy a pixmap. The memory is freed when
there is no more reference

```
procedure Create_From_Data
  (Pixmap             : out   Gdk_Pixmap;
   Window             :       Gdk.Window.Gdk_Window;
   Data               :       String;
   Width              :       Gint;
   Height             :       Gint;
   Depth              :       Gint;
   Fg                 :       Color.Gdk_Color;
   Bg                 :       Color.Gdk_Color);
```

Create a pixmap from data in XBM format.
Window is used to determine default values for the new bitmap, can be null in which case

the root window is used. Data is the XBM data. Width is the width of the new bitmap in pixels. Height is the height of the new bitmap in pixels. Depth is the depth (number of bits per pixel) of the new pixmap. Fg is the foreground color. Bg is the background color.

```
procedure Create_From_Xpm
  (Pixmap             : out    Gdk_Pixmap;
   Window             :        Gdk.Window.Gdk_Window;
   Mask               : in out Gdk.Bitmap.Gdk_Bitmap;
   Transparent        :        Gdk.Color.Gdk_Color;
   Filename           :        String);
```

Create a pixmap from a XPM file.
Window is used to determine default values for the new pixmap. Mask is a pointer to a place to store a bitmap representing the transparency mask of the XPM file. Can be null, in which case transparency will be ignored. Transparent is the color to be used for the pixels that are transparent in the input file. Can be null, in which case a default color will be used. Filename is the filename of a file containing XPM data.

```
procedure Create_From_Xpm
  (Pixmap             : out    Gdk_Pixmap;
   Window             :        Gdk.Window.Gdk_Window;
   Colormap           :        Gdk.Color.Gdk_Colormap;
   Mask               : in out Gdk.Bitmap.Gdk_Bitmap;
   Transparent        :        Gdk.Color.Gdk_Color;
   Filename           :        String);
```

Create a pixmap from a XPM file using a particular colormap.
Window is used to determine default values for the new pixmap. Can be null if colormap is given. Colormap is the Gdk_Colormap that the new pixmap will use. If omitted, the colormap for window will be used. Mask is a pointer to a place to store a bitmap representing the transparency mask of the XPM file. Can be null, in which case transparency will be ignored. Transparent is the color to be used for the pixels that are transparent in the input file. Can be null, in which case a default color will be used. Filename is the filename of a file containing XPM data.

```
procedure Create_From_Xpm_D
  (Pixmap             : out    Gdk_Pixmap;
   Window             :        Gdk.Window.Gdk_Window;
   Mask               : in out Gdk.Bitmap.Gdk_Bitmap;
   Transparent        :        Gdk.Color.Gdk_Color;
   Data               :        Gtkada.Types.Chars_Ptr_Array);
```

Create a pixmap from data in XPM format.
Window is used to determine default values for the new pixmap. Mask is a pointer to a place to store a bitmap representing the transparency mask of the XPM file. Can be null, in which case transparency will be ignored. Transparent will be used for the pixels that are transparent in the input file. Can be null in which case a default color will be used. Data is a pointer to a string containing the XPM data.

```
procedure Create_From_Xpm_D
  (Pixmap             : out    Gdk_Pixmap;
   Window             :        Gdk.Window.Gdk_Window;
   Colormap           :        Gdk.Color.Gdk_Colormap;
   Mask               : in out Gdk.Bitmap.Gdk_Bitmap;
   Transparent        :        Gdk.Color.Gdk_Color;
   Data               :        Gtkada.Types.Chars_Ptr_Array);
```

Create a pixmap from data in XPM format using a particular colormap.
Window is used to determine default values for the new pixmap. Colormap is the

Gdk_Colormap that the new pixmap will be use. If omitted, the colormap for window will be used. Mask is a pointer to a place to store a bitmap representing the transparency mask of the XPM file. Can be null, in which case transparency will be ignored. Transparent will be used for the pixels that are transparent in the input file. Can be null in which case a default color will be used. Data is a pointer to a string containing the XPM data.

46 Package Gdk.Rgb

This package implements a client-side pixmap. As opposed to the pixmaps found in Gdk.Pixmap, this one simply implements a local buffer, which can be manipulated at the pixel level easily. This buffer then needs to be sent to the server. The major efficiency difference is that the same amount of data needs to be sent to the server no matter how much things were modified. Gdk.Pixmaps requires one communication with the server per drawing function. Some X servers are also optimized so that the buffers in this package can be implemented in shared memory with the server, which of course makes it much faster to transfer the data. This package is basically an implementation of XImage (on X-Window), which means that it handles transparently different depths, byte ordering,... It also provides some color dithering functions.

See the commands Get_Visual and Get_Cmap below on how to use the colormaps and visual with this package

Dithering simulates a higher number of colors than what is available on the current visual (only for 8-bit and 16-bit displays).

46.1 Types

type Gdk_Rgb_Cmap **is new** Gdk.C_Proxy;

This is the full colormap, ie a set of 256 Rgb items. You can extract values using the functions Get or Set below.

type Gdk_Rgb_Dither **is**
(Dither_None, Dither_Normal, Dither_Max);

The three kinds of dithering that are implemented in this package: - Dither_None: No dithering will be done - Dither_Normal: Specifies dithering on 8 bit displays, but not 16-bit. Usually the best choice. - Dither_Max: Specifies dithering on every kind of display for Gdk_Rgb_Dither'Size use Glib.Gint'Size;

type Rgb_Buffer **is array** (Glib.Guint **range <>**) **of** Rgb_Record;

type Rgb_Buffer_Access **is access all** Unchecked_Rgb_Buffer;

type Rgb_Cmap_Index **is new** Natural **range** 0 .. 255;

subtype Rgb_Item **is** Glib.Guint32;

This represents the coding for a rbg value. The exact encoding depends on the visual used and its depth (pseudo-color, true-color, ...)

```
type Unchecked_Rgb_Buffer is array (Glib.Guint) of Rgb_Record;
```

46.2 Subprograms

```
function Get_Visual             return Gdk.Visual.Gdk_Visual;
```

See Get_Cmap.

```
function Get_Cmap               return Gdk.Color.Gdk_Colormap;
```

Return the visual and the color map used internally in this package.
Note that these are not the same as returned by Gtk.Widget or Gdk.Window, and you should use these if you are using this package.

The drawable you intend to copy the RGB buffer to must use this visual and this colormap. Therefore, before creating the widget, you need to do the following:

- Gtk.Widget.Push_Colormap (Gdk.Rgb.Get_Cmap);
- Gtk_New (....)
- Gtk.Widget.Pop_Colormap;

46.2.1 Color manipulation

```
function Xpixel_From_Rgb
  (Value              : in    Rgb_Item)
   return Glib.Gulong;
```

Convert the Rgb representation to the usual one found in Gdk.Color.
pragma Deprecated (Xpixel_From_Rgb);

```
procedure GC_Set_Foreground
  (GC                 :       Gdk.GC.Gdk_GC;
   Value              :       Rgb_Item);
```

See GC_Set_Background.
pragma Deprecated (GC_Set_Foreground);

```
procedure GC_Set_Background
  (GC                 :       Gdk.GC.Gdk_GC;
   Value              :       Rgb_Item);
```

Modify the foreground and the background of a graphic context with a
value. These are exactly the same functions has found in Gdk.Gc, but do not use the same parameters. pragma Deprecated (GC_Set_Background);

46.2.2 Colormap manipulation

```
function Get
  (Cmap               :       Gdk_Rgb_Cmap;
   Index              :       Rgb_Cmap_Index)
   return Rgb_Item;
```

Access an item in a colormap.

```
procedure Set
  (Cmap               :       Gdk_Rgb_Cmap;
   Index              :       Rgb_Cmap_Index;
   Value              :       Rgb_Item);
```

Set an item in Cmap.

```
procedure Gdk_New
  (Cmap               : out   Gdk_Rgb_Cmap;
   Colors             :       Glib.Guint32_Array);
```

Create a colormap.

```
procedure Free
  (Cmap               :       Gdk_Rgb_Cmap);
```

Free a colormap.

46.2.3 Drawing Images

```
procedure Draw_Rgb_Image
  (Drawable           :       Gdk.Drawable.Gdk_Drawable;
   GC                 :       Gdk.GC.Gdk_GC;
   X, Y               :       Glib.Gint;
   Width, Height      :       Glib.Gint;
   Dith               :       Gdk_Rgb_Dither;
   Rgb_Buf            :       Rgb_Buffer;
   Rowstride          :       Glib.Gint);
procedure Draw_Rgb_Image
  (Drawable           :       Gdk.Drawable.Gdk_Drawable;
   GC                 :       Gdk.GC.Gdk_GC;
   X, Y               :       Glib.Gint;
   Width, Height      :       Glib.Gint;
   Dith               :       Gdk_Rgb_Dither;
   Rgb_Buf            :       Unchecked_Rgb_Buffer;
   Rowstride          :       Glib.Gint);
```

Render a Gdk buffer with 24 bit Data.
Such a buffer is a one dimensional array of bytes, where every byte triplet makes up a pixel (byte 0 is red, byte 1 is green and byte 2 is blue).

- Width: Number of pixels (byte triplets) per row of the image.
- Height: Number of rows in the image.
- RowStride: Number of bytes between rows... (row n+1 will start at byte row n + Rowstride). Gdk.Rgb is faster if both the source pointer and the rowstride are aligned to a 4 byte boundary.
- (X, Y, Width, Height): Define a region in the target to copy the buffer to.

```
procedure Draw_Rgb_Image_Dithalign
  (Drawable           :       Gdk.Drawable.Gdk_Drawable;
   GC                 :       Gdk.GC.Gdk_GC;
   X, Y               :       Glib.Gint;
   Width, Height      :       Glib.Gint;
   Dith               :       Gdk_Rgb_Dither;
   Rgb_Buf            :       Rgb_Buffer;
   Rowstride          :       Glib.Gint;
   Xdith, Ydith       :       Glib.Gint);
procedure Draw_Rgb_Image_Dithalign
  (Drawable           :       Gdk.Drawable.Gdk_Drawable;
   GC                 :       Gdk.GC.Gdk_GC;
   X, Y               :       Glib.Gint;
   Width, Height      :       Glib.Gint;
   Dith               :       Gdk_Rgb_Dither;
   Rgb_Buf            :       Unchecked_Rgb_Buffer;
   Rowstride          :       Glib.Gint;
   Xdith, Ydith       :       Glib.Gint);
```

Same kind of function as above, but for different buffer types (???).

```
procedure Draw_Rgb_32_Image
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Rgb_Buffer;
   Rowstride          :      Glib.Gint);
procedure Draw_Rgb_32_Image
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Unchecked_Rgb_Buffer;
   Rowstride          :      Glib.Gint);
```

Same kind of function as above, but for different buffer types (???).

```
procedure Draw_Rgb_32_Image_Dithalign
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Rgb_Buffer;
   Rowstride          :      Glib.Gint;
   Xdith, Ydith       :      Glib.Gint);
procedure Draw_Rgb_32_Image_Dithalign
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Unchecked_Rgb_Buffer;
   Rowstride          :      Glib.Gint;
   Xdith, Ydith       :      Glib.Gint);
```

Same kind of function as above, but for different buffer types (???).

```
procedure Draw_Gray_Image
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Rgb_Buffer;
   Rowstride          :      Glib.Gint);
procedure Draw_Gray_Image
  (Drawable           :      Gdk.Drawable.Gdk_Drawable;
   GC                 :      Gdk.GC.Gdk_GC;
   X, Y               :      Glib.Gint;
   Width, Height      :      Glib.Gint;
   Dith               :      Gdk_Rgb_Dither;
   Rgb_Buf            :      Unchecked_Rgb_Buffer;
   Rowstride          :      Glib.Gint);
```

Same kind of function as above, but for different buffer types (???).

```
procedure Draw_Indexed_Image
```

```
   (Drawable           :      Gdk.Drawable.Gdk_Drawable;
    GC                 :      Gdk.GC.Gdk_GC;
    X, Y               :      Glib.Gint;
    Width, Height      :      Glib.Gint;
    Dith               :      Gdk_Rgb_Dither;
    Rgb_Buf            :      Rgb_Buffer;
    Rowstride          :      Glib.Gint;
    Cmap               :      Gdk_Rgb_Cmap);
procedure Draw_Indexed_Image
   (Drawable           :      Gdk.Drawable.Gdk_Drawable;
    GC                 :      Gdk.GC.Gdk_GC;
    X, Y               :      Glib.Gint;
    Width, Height      :      Glib.Gint;
    Dith               :      Gdk_Rgb_Dither;
    Rgb_Buf            :      Unchecked_Rgb_Buffer;
    Rowstride          :      Glib.Gint;
    Cmap               :      Gdk_Rgb_Cmap);
```

Same kind of function as above, but for different buffer types (???).

47 Package Gdk.Screen

Gdk_Screen objects are the GDK representation of a physical screen. It is used throughout GDK and GTK+ to specify which screen the top level windows are to be displayed on. It is also used to query the screen specification and default settings such as the default colormap (Get_Default_Colormap), the screen width (Get_Width), etc.

Note that a screen may consist of multiple monitors which are merged to form a large screen area.

47.1 Signals

- **"size_changed"**

```
procedure Handler (Screen : access Gdk_Screen_Record'Class);
```

Emitted when the pixel width or height of a screen changes.

47.2 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal type used for screens

47.2.1 Display

These subprograms should really be in gdk-display.ads to match what is@* done for gtk+ itself, but that would create dependency circularities. Ada 2005 has support for these, but we want GtkAda to build with Ada95 compilers.

```
function Get_Screen
  (Display            : access Gdk.Display.Gdk_Display_Record'Class;
   Screen_Num         :      Glib.Gint)
   return Gdk_Screen;
```

Returns a screen object for one of the screens of the display.

```
function Get_Default_Screen
  (Display            : access Gdk.Display.Gdk_Display_Record'Class)
   return Gdk_Screen;
```

Get the default Gdk_Screen for display.

```
procedure Get_Pointer
  (Display            : access Gdk.Display.Gdk_Display_Record'Class;
   Screen             : out   Gdk_Screen;
   X                  : out   Glib.Gint;
   Y                  : out   Glib.Gint;
   Mask               : out   Gdk.Types.Gdk_Modifier_Type);
```

Gets the current location of the pointer and the current modifier
mask for a given display. (X, Y) are coordinates relative to the root window on the display

```
procedure Warp_Pointer
  (Display            : access Gdk.Display.Gdk_Display_Record'Class;
   Screen             : access Gdk_Screen_Record;
   X                  :      Glib.Gint;
   Y                  :      Glib.Gint);
```

Warps the pointer of display to the point x,y on the screen screen,
unless the pointer is confined to a window by a grab, in which case it will be moved as far

as allowed by the grab. Warping the pointer creates events as if the user had moved the mouse instantaneously to the destination.

Note that the pointer should normally be under the control of the user. This function was added to cover some rare use cases like keyboard navigation support for the color picker in the GtkColorSelectionDialog.

47.2.2 Screens

```
function Get_Default            return Gdk_Screen;
```

Gets the default screen for the default display

```
function Get_Display
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Display.Gdk_Display;
```

Gets the display to which the screen belongs.

```
procedure Set_Default_Colormap
  (Screen              : access Gdk_Screen_Record;
   Colormap            :      Gdk.Gdk_Colormap);
function Get_Default_Colormap
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Colormap;
```

Gets the default colormap for screen.

```
function Get_System_Colormap
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Colormap;
```

Gets the system's colormap for screen

```
function Get_System_Visual
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Visual;
```

Get the system's default visual for screen. This is the visual for the root window of the display. The return value should not be freed.

```
function Get_Rgb_Colormap
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Colormap;
```

Gets the preferred colormap for rendering image data on screen. Not a very useful function; historically, GDK could only render RGB image data to one colormap and visual, but in the current version it can render to any colormap and visual. So there's no need to call this function.

```
function Get_Rgb_Visual
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Visual;
```

Gets a "preferred visual" chosen by GdkRGB for rendering image data on screen. In previous versions of GDK, this was the only visual GdkRGB could use for rendering. In current versions, it's simply the visual GdkRGB would have chosen as the optimal one in those previous versions. GdkRGB can now render to drawables with any visual.

```
function Get_Rgba_Colormap
  (Screen              : access Gdk_Screen_Record)
   return Gdk.Gdk_Colormap;
```

Gets a colormap to use for creating windows or pixmaps with an alpha channel. The windowing system on which GTK+ is running may not support this capability, in which case NULL will be returned. Even if a non-NULL value is returned, its possible that the window's alpha channel won't be honored when displaying the window on the screen: in particular, for X an appropriate windowing manager and compositing manager must be running to provide appropriate display.

```
function Get_Rgba_Visual
  (Screen             : access Gdk_Screen_Record)
   return Gdk.Gdk_Visual;
```

Gets a visual to use for creating windows or pixmaps with an alpha channel. See the docs for Get_Rgba_Colormap for caveats.

```
function Get_Root_Window
  (Screen             : access Gdk_Screen_Record)
   return Gdk.Gdk_Window;
```

Gets the root window of screen.

```
function Get_Number
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
```

Gets the index of screen among the screens in the display to which it belongs.

```
function Get_Width
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
function Get_Height
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
```

Gets the size of screen in pixels

```
function Get_Width_Mm
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
function Get_Height_Mm
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
```

Gets the width of screen in millimeters. Note that on some X servers this value will not be correct.

```
function Make_Display_Name
  (Screen             : access Gdk_Screen_Record)
   return String;
```

Determines the name to pass to Gdk.Display.Open to get a GdkDisplay with this screen as the default screen.

```
function Get_N_Monitors
  (Screen             : access Gdk_Screen_Record)
   return Glib.Gint;
```

Returns the number of monitors which screen consists of.

47.2.3 Monitors

```
procedure Get_Monitor_Geometry
  (Screen             : access Gdk_Screen_Record;
   Monitor_Num        :      Glib.Gint;
```

```
   Dest               : out   Gdk.Rectangle.Gdk_Rectangle);
```

Retrieves the Gdk_Rectangle representing the size and position of the individual monitor within the entire screen area. Note that the size of the entire screen area can be retrieved via Get_Width and Get_Height.

```
function Get_Monitor_At_Point
  (Screen             : access Gdk_Screen_Record;
   X                  :      Glib.Gint;
   Y                  :      Glib.Gint)
   return Glib.Gint;
```

Returns the monitor number in which the point (X,Y) is located. These are coordinates within the virtual screen. The closest monitor is returned when (X, Y) is not in any monitor

```
function Get_Monitor_At_Window
  (Screen             : access Gdk_Screen_Record;
   Window             :      Gdk.Gdk_Window)
   return Glib.Gint;
```

Returns the number of the monitor in which the largest area of the bounding rectangle of Window resides.

```
procedure Get_Setting
  (Screen             : access Gdk_Screen_Record;
   Name               :      String;
   Value              : out   Glib.Values.GValue;
   Found              : out   Boolean);
```

Retrieves a desktop-wide setting such as double-click time for the Gdk_Screen screen.

48 Package Gdk.Threads

This package provides simple primitives to write multi-threaded applications with GtkAda.
See the GtkAda User's Guide for more details (section Tasking with GtkAda).

48.1 Subprograms

```
procedure G_Init
  (Vtable             :      System.Address
                       := System.Null_Address);
```

Initialize the Glib internal threading support.
This procedure must be called before any call to Enter or Leave. The parameter Vtable should never be used for now.

```
procedure Init;
```

Initialize the Gdk internal threading support.
This function must be called after G_Init and before any call to Enter or Leave.

```
procedure Enter;
```

Take the GtkAda global lock.
See the GtkAda User's Guide for more details (section Tasking with GtkAda).

```
procedure Leave;
```

Release the GtkAda global lock.
See the GtkAda User's Guide for more details (section Tasking with GtkAda).

49 Package Glade

This package is a binding to the libglade library that provides routines to create widgets dynamically from an XML definition file. See also Glade.XML

49.1 Subprograms

49.1.1 dynamic loading of libglade extensions

```
procedure Require
  (Library            :      String);
procedure Provide
  (Library            :      String);
```

50 Package Glade.XML

This package is a binding to the libglade library that provides routines to create widgets dynamically from an XML definition file. See also glade.ads

50.1 Subprograms

```
procedure Gtk_New
  (XML                : out   Glade_XML;
   Fname              :       String;
   Root               :       String := "";
   Domain             :       String := "");
```

Create a new GladeXML object (and the corresponding widgets)
from the XML file fname. Optionally it will only build the interface from the widget node Root (if it is not empty). This feature is useful if you only want to build say a toolbar or menu from the XML file, but not the window it is embedded in. Note also that the XML parse tree is cached to speed up creating another GladeXML object for the same file.

Domain, if not null, is the international domain to use for string translation. See Gtkada.Intl for more information.

```
procedure Gtk_New_From_Buffer
  (XML                : out   Glade_XML;
   Buffer             :       String;
   Root               :       String := "";
   Domain             :       String := "");
```

Create a new Glade_XML.
Similar to previous procedure, but the XML contents are read from memory directly.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Glade_XML.

```
procedure Signal_Connect
  (XML                : access Glade_XML_Record;
   Handlername        :       String;
   Func               :       System.Address;
   User_Data          :       System.Address);
```

Warning: Func should be a lowel level C callback, taking a low level
C widget as the first parameter, e.g: procedure Func (Widget : Gtk.Item_Factory.Limited_Widget);

```
function Get_Widget
  (XML                : access Glade_XML_Record;
   Name               :       String)
   return Gtk_Widget;
```

This function is used to get the Gtk_Widget corresponding to
name in the interface description. You would use this if you have to do anything to the widget after loading.

```
function Relative_File
  (XML                : access Glade_XML_Record;
   Filename           :       String)
   return String;
```

This function resolves a relative pathname, using the directory
of the XML file as a base. If the pathname is absolute, then the original filename is returned.

```
function Get_Widget_Name
  (Widget             : access Gtk_Widget_Record'Class)
   return String;
function Get_Widget_Tree
  (Widget             : access Gtk_Widget_Record'Class)
   return Glade_XML;
```

This function is used to get the GladeXML object that built this widget.

51 Package Glade_XML

52 Package Gnome

This is the root of the Gnome hierarchy. It provides initialization routines.

52.1 Types

type Gnome_Preferences_Type **is**
(Preferences_Never, Preferences_User, Preferences_Always);

Do something never, only when the user wants, or always.

52.2 Subprograms

```
function Init
  (App_Id             :      String;
   App_Version        :      String)
   return Boolean;
```

Initialize Gnome.
You should call this function before anything other gnome related actions. Return True in case of success, False otherwise.

53 Package Gnome.App_Helper

This module lets you easily create menus and toolbars for your applications. You basically define a hierarchy of arrays of UI_Info structures, and you later call the provided functions to create menu bars or tool bars.

53.1 Types

type Generic_Callback **is access procedure**
(Widget : **access** Gtk_Widget_Record'Class);

type UI_Info **is private**;

type UI_Info_Array **is array** (Natural **range** <>) **of** UI_Info;

This is the structure that defines an item in a menu bar or toolbar. The idea is to create an array of such structures with the information needed to create menus or toolbars. The most convenient way to create such a structure is to use the UI_Info_* functions provided below.

type UI_Info_Array_Access **is access** UI_Info_Array;

type UI_Info_Configurable_Types **is**
(Configurable_Item_New,
Configurable_Item_Open,
Configurable_Item_Save,
Configurable_Item_Save_As,
Configurable_Item_Revert,
Configurable_Item_Print,
Configurable_Item_Print_Setup,
Configurable_Item_Close,
Configurable_Item_Exit,
Configurable_Item_Cut,
Configurable_Item_Copy,
Configurable_Item_Paste,
Configurable_Item_Clear,
Configurable_Item_Undo,
Configurable_Item_Redo,
Configurable_Item_Find,
Configurable_Item_Find_Again,
Configurable_Item_Replace,
Configurable_Item_Properties,
Configurable_Item_Preferences,
Configurable_Item_About,
Configurable_Item_Select_All,
Configurable_Item_New_Window,
Configurable_Item_Close_Window,
Configurable_Item_New_Game,
Configurable_Item_Pause_Game,

```
   Configurable_Item_Restart_Game,
   Configurable_Item_Undo_Move,
   Configurable_Item_Redo_Move,
   Configurable_Item_Hint,
   Configurable_Item_Scores,
   Configurable_Item_End_Game);
```

```
type UI_Pixmap_Type is
   (Pixmap_None,
      -- No pixmap specified

   Pixmap_Stock,
      -- Use a stock pixmap (Gnome.Stock)
```

53.2 Subprograms

```
function UI_New_Item
  (Label              :      String;
   Hint               :      String := "";
   Callback           :      Generic_Callback := null;
   Pixmap_Type        :      UI_Pixmap_Type := Pixmap_None;
   Pixmap_Info        :      String := "";
   Accelerator_Key    :      Gdk_Key_Type := 0;
   Ac_Mods            :      Gdk_Modifier_Type := 0)
   return UI_Info;
```

Return a normal item, or radio item if it is inside a radioitems group.
Label: String to use in the label Hint: The status bar message Callback : Function to call when the item is activated Pixmap_Type: Type of pixmap for the item Pixmap_Info:

- For Pixmap_Stock, the stock icon name.
- For Pixmap_Data, the inline xpm data ???
- For Pixmap_Filename, the filename string. Accelerator_Key: Accelerator key Ac_Mods: Mask of modifier keys for the accelerator

```
function UI_New_Subtree
  (Label              :      String;
   Info               :      UI_Info_Array_Access;
   Pixmap_Type        :      UI_Pixmap_Type := Pixmap_None;
   Pixmap_Info        :      String := "";
   Accelerator_Key    :      Gdk_Key_Type := 0;
   Ac_Mods            :      Gdk_Modifier_Type := 0)
   return UI_Info;
```

Item that defines a subtree/submenu

```
function UI_Info_Item
  (Label              :      String;
   Tooltip            :      String;
   Callback           :      Generic_Callback;
   Xpm_Data           :      Chars_Ptr_Array)
   return UI_Info;
```

Insert an item with an inline xpm icon

```
function UI_Info_Item_Stock
  (Label              :      String;
   Tooltip            :      String;
   Callback           :      Generic_Callback;
   Stock_Id           :      String)
   return UI_Info;
```

Insert an item with a stock icon

```
function UI_Info_Item_None
  (Label              :      String;
   Tooltip            :      String;
   Callback           :      Generic_Callback)
   return UI_Info;
```

Insert an item with no icon

```
function UI_Info_Toggleitem
  (Label              :      String;
   Tooltip            :      String;
   Callback           :      Generic_Callback;
   Xpm_Data           :      Chars_Ptr_Array)
   return UI_Info;
```

Insert a toggle item (check box) with an inline xpm icon

```
function UI_Info_Help
  (App_Name           :      String)
   return UI_Info;
```

Insert all the help topics based on the application's id

```
function UI_Info_Subtree
  (Label              :      String;
   Tree               :      UI_Info_Array_Access)
   return UI_Info;
```

Insert a subtree (submenu)

```
function UI_Info_Subtree_Hint
  (Label              :      String;
   Hint               :      String;
   Tree               :      UI_Info_Array_Access)
   return UI_Info;
```

Insert a subtree with a hint

```
function UI_Info_Subtree_Stock
  (Label              :      String;
   Tree               :      UI_Info_Array_Access;
   Stock_Id           :      String)
   return UI_Info;
```

Insert a subtree (submenu) with a stock icon

```
function UI_Info_Radioitem
  (Label              :      String;
   Tooltip            :      String;
   Callback           :      Generic_Callback;
   Xpm_Data           :      Chars_Ptr_Array)
   return UI_Info;
```

Insert a radio item with an inline xpm icon

```
function UI_Info_Menu_New_Item
  (Label              :      String;
   Tooltip            :      String;
```

```
   Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_New_Subtree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;
```

If you have more than one new type, use this tree

```
function UI_Info_Menu_Open_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Save_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Save_As_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Revert_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Print_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Print_Setup_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Close_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Exit_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Cut_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Copy_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Paste_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Select_All_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Clear_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Undo_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Redo_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Find_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
```

```
function UI_Info_Menu_Find_Again_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Replace_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Properties_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Preferences_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_New_Window_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Close_Window_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_About_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_New_Game_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Pause_Game_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Restart_Game_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Undo_Move_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Redo_Move_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Hint_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_Scores_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function UI_Info_Menu_End_Game_Item
  (Callback           :      Generic_Callback)
   return UI_Info;
function Helper_Gettext
  (Str                :      String)
   return String;
function UI_Info_Menu_File_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;
function UI_Info_Menu_Edit_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;
```

```
function UI_Info_Menu_View_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

function UI_Info_Menu_Settings_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

function UI_Info_Menu_Files_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

function UI_Info_Menu_Windows_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

function UI_Info_Menu_Help_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

function UI_Info_Menu_Game_Tree
  (Tree               :      UI_Info_Array_Access)
   return UI_Info;

procedure Gnome_Accelerators_Sync;
```

Flush the accelerator definitions into the application specific configuration file ~/.gnome/accels/<app-id>.

```
procedure Fill_Menu
  (Menu_Shell         : access Gtk_Menu_Shell_Record'Class;
   Info               :      UI_Info_Array_Access;
   Accel_Group        :      Gtk_Accel_Group := null;
   Uline_Accels       :      Boolean := False;
   Pos                :      Gint := 0;
   Object             :      Gtk_Widget := null);
```

Fill the specified menu shell with items created from the specified info, inserting them from the item no. pos on. The accel group will be used as the accel group for all newly created sub menus and serves as the global accel group for all menu item hotkeys. If it is passed as null, global hotkeys will be disabled. The Uline_Accels argument determines whether underline accelerators will be featured from the menu item labels. Object, if not null, will be passed to the callbacks as the emitter (similarly to what is done in Object_Connect).

```
procedure Create_Menus
  (App                : access Gnome_App_Record'Class;
   Info               :      UI_Info_Array_Access);
```

Construct a menu bar and attach it to the specified application window

```
procedure Fill_Toolbar
  (Toolbar            : access Gtk_Toolbar_Record'Class;
   Info               :      UI_Info_Array_Access;
   Accel_Group        :      Gtk_Accel_Group := null);
```

Fill the specified toolbar with buttons created from the specified info. If Accel_Group is not null, then the items' accelerator keys are put into it.

```
procedure Create_Toolbar
  (App                : access Gnome_App_Record'Class;
   Info               :      UI_Info_Array_Access);
```

Construct a toolbar and attach it to the specified application window

```
function Find_Menu_Pos
  (Parent             : access Gtk_Widget_Record'Class;
   Path               :      String;
   Pos                :      Gint)
   return Gtk_Widget;
```

Find menu item described by path (see below for details) starting in the Gtk_Menu_Shell top and return its parent Gtk_Menu_Shell and the position after this item in pos: Gtk.Menu_Shell.Insert (P, W, Pos) would then insert widget w in Gtk_Menu_Shell P right after the menu item described by path. The path argument should be in the form "File/.../.../Something". "" will insert the item as the first one in the menubar "File/" will insert it as the first one in the File menu "File/Settings" will insert it after the Setting item in the File menu use of "File/<Separator>" should be obvious. However this stops after the first separator.

```
procedure Remove_Menus
  (App                : access Gnome_App_Record'Class;
   Path               :      String;
   Items              :      Gint);
```

Remove num items from the existing app's menu structure begining with item described by path

```
procedure Remove_Menu_Range
  (App                : access Gnome_App_Record'Class;
   Path               :      String;
   Start              :      Gint;
   Items              :      Gint);
```

Same as the above, except it removes the specified number of items from the existing app's menu structure begining with item described by path, plus the number specified by start - very useful for adding and removing Recent document items in the File menu.

```
procedure Insert_Menus
  (App                : access Gnome_App_Record'Class;
   Path               :      String;
   Menu_Info          :      UI_Info_Array_Access);
```

what does it do ???

```
procedure Install_Statusbar_Menu_Hints
  (Bar                :      Gtk_Status_Bar;
   Info               :      UI_Info_Array_Access);
procedure Install_Menu_Hints
  (App                :      Gnome_App;
   Info               :      UI_Info_Array_Access);
```

54 Package Gnome.Color_Picker

The Gnome_Color_Picker widget is a simple color picker in a button. The button displays a sample of the currently selected color. When the user clicks on the button, a color selection dialog pops up. The color picker emits the "color_changed" signal when the color is set By default, the color picker does dithering when drawing the color sample box. This can be disabled for cases where it is useful to see the allocated color without dithering.

54.1 Signals

- **"color_set"**

```
procedure Handler (Cpicker : access Gnome_Color_Picker_Record'Class;
R       : Guint;
G       : Guint;
B       : Guint;
A       : Guint);
```

The color is set

54.2 Subprograms

```
procedure Gnome_New
  (Color_Picker       : out   Gnome_Color_Picker);
```

Create a new Color_Picker

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Color_Picker.

```
procedure Set
  (Cpicker            : access Gnome_Color_Picker_Record;
   R                  : in    Gdouble;
   G                  : in    Gdouble;
   B                  : in    Gdouble;
   A                  : in    Gdouble := 0.0);
```

Set the color in the picker, as doubles range [0.0, 1.0]

```
procedure Get
  (Cpicker            : access Gnome_Color_Picker_Record;
   R                  : out   Gdouble;
   G                  : out   Gdouble;
   B                  : out   Gdouble;
   A                  : out   Gdouble);
```

Get the color in the picker, as doubles range [0.0, 1.0]

```
procedure Set
  (Cpicker            : access Gnome_Color_Picker_Record;
   R                  : in    Guint8;
   G                  : in    Guint8;
   B                  : in    Guint8;
   A                  : in    Guint8 := 0);
```

Set the color in the picker, as guint8s range [0, 255]

```
procedure Get
  (Cpicker            : access Gnome_Color_Picker_Record;
   R                  : out   Guint8;
   G                  : out   Guint8;
   B                  : out   Guint8;
```

```
   A                    : out   Guint8);
```

Get the color in the picker, as guint8s range [0, 255]

```
procedure Set
  (Cpicker              : access Gnome_Color_Picker_Record;
   R                    : in    Gushort;
   G                    : in    Gushort;
   B                    : in    Gushort;
   A                    : in    Gushort := 0);
```

Set the color in the picker, as gushorts range [0, 65535]

```
procedure Get
  (Cpicker              : access Gnome_Color_Picker_Record;
   R                    : out   Gushort;
   G                    : out   Gushort;
   B                    : out   Gushort;
   A                    : out   Gushort);
```

Get the color in the picker, as gushorts range [0, 65535]

```
procedure Set_Dither
  (Cpicker              : access Gnome_Color_Picker_Record;
   Dither               : in    Boolean);
```

Set whether the picker should dither the color sample
or just paint a solid rectangle.

```
procedure Set_Use_Alpha
  (Cpicker              : access Gnome_Color_Picker_Record;
   Use_Alpha            : in    Boolean);
```

Set whether the picker should use the alpha channel or not.

```
procedure Set_Title
  (Cpicker              : access Gnome_Color_Picker_Record;
   Title                : in    String);
```

Set the title of the color selection dialog.

55 Package Gnome.Stock

These functions provide an applications programmer with default icons for toolbars, menu pixmaps, etc. One such 'icon' should have at least three pixmaps to reflect it's state. There is a 'regular' pixmap, a 'disabled' pixmap and a 'focused' pixmap. You can get either each of these pixmaps by calling Gnome.Stock.Pixmap or you can get a widget by calling Gnome.Stock.Pixmap_Widget. This widget is a container which shows the pixmap, that is reflecting the current state of the widget. If for example you Gtk.Container.Add this widget to a button, which is currently not sensitive, the widget will just show the 'disabled' pixmap. If the state of the button changes to sensitive, the widget will change to the 'regular' pixmap. The 'focused' pixmap will be shown, when the mouse pointer enters the widget.

We now have stock buttons too. To use them, just replace any Gtk.Button.Gtk_New with Gnome.Stock.Button (Button_...). This function returns a Gtk_Button with a gettexted default text and an icon.

55.1 Subprograms

```
function Get_Type                    return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

56 Package Gnome.UI_Defs

This file defines standard sizes, spacings, and whatever else seems standardizable via simple definitions.

57 Package Gnome_Color_Picker

58 Package Gnome_Stock

59 Package Gtk

This package provides some basic Gtk+ functionalities such as getting the version number. For general GtkAda initializations, see Gtk.Main.

59.1 Types

type Gtk_Notebook_Page **is new** Gdk.C_Proxy;

A page of the notebook. It can contain a single child, and is also associated with a tab label used to select that page in the notebook.

subtype Gtk_Type **is** Glib.GType;

Renaming used for compatiblity. Note: Gtk_Type_* constants have been replaced by GType_* constants in Glib.

59.2 Subprograms

```
function Major_Version          return Guint;
```

Return the major version number for Gtk+ that was linked.
Note that this is not necessarily the same as for GtkAda. It could also be different when your application is running, if the dynamic linker find some other GtkAda library. Use Gtk.Main.Check_Version to ensure that the two versions are compatible If the version is 1.2.6, returns 1.

```
function Minor_Version          return Guint;
```

Return the minor version number for Gtk+.
Note that this is not necessarily the same as for GtkAda. If the version is 1.2.6, returns 2.

```
function Micro_Version          return Guint;
```

Return the micro version number for Gtk+.
Note that this is not necessarily the same as for GtkAda. If the version is 1.2.6, returns 6.

60 Package Gtk.About_Dialog

The Gtk_About_Dialog offers a simple way to display information about a program like its logo, name, copyright, website and license. It is also possible to give credits to the authors, documenters, translators and artists who have worked on the program. An about dialog is typically opened when the user selects the About option from the Help menu. All parts of the dialog are optional.

About dialog often contain links and email addresses. Gtk_About_Dialog supports this by offering global hooks, which are called when the user clicks on a link or email address, see Set_Email_Hook and Set_Url_Hook. Email addresses in the authors, documenters and artists properties are recognized by looking for <user@host>, URLs are recognized by looking for http://url, with url extending to the next space, tab or line break.

To make constructing a Gtk_About_Dialog as convenient as possible, you can use the function gtk_show_about_dialog which constructs and shows a dialog and keeps it around so that it can be shown again.

60.1 Types

type Activate_Link_Func **is access procedure**

60.2 Subprograms

```
procedure Gtk_New
  (About              : out   Gtk_About_Dialog);

function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Button.

```
procedure Set_Artists
  (About              : access Gtk_About_Dialog_Record;
   Artists            :      GNAT.Strings.String_List);

function Get_Artists
  (About              : access Gtk_About_Dialog_Record)
   return GNAT.Strings.String_List;
```

Returns the strings which are displayed in the artists tab
of the secondary credits dialog. The returned value must be freed by the caller, as well as the Artists parameter.

```
procedure Set_Authors
  (About              : access Gtk_About_Dialog_Record;
   Authors            :      GNAT.Strings.String_List);

function Get_Authors
  (About              : access Gtk_About_Dialog_Record)
   return GNAT.Strings.String_List;
```

Returns the string which are displayed in the authors tab
of the secondary credits dialog. The returned value must be freed by the caller, as well as the Authors paramaeter.

```
procedure Set_Comments
  (About              : access Gtk_About_Dialog_Record;
   Comments           :      String);

function Get_Comments
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the comments string.

```
procedure Set_Copyright
  (About              : access Gtk_About_Dialog_Record;
   Copyright          :      String);

function Get_Copyright
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the copyright string.

```
procedure Set_Documenters
  (About              : access Gtk_About_Dialog_Record;
   Documenters        :      GNAT.Strings.String_List);

function Get_Documenters
  (About              : access Gtk_About_Dialog_Record)
   return GNAT.Strings.String_List;
```

Returns the string which are displayed in the documenters
tab of the secondary credits dialog. The returned value must be freed by the caller, as well as the Documenters parameter.

```
procedure Set_License
  (About              : access Gtk_About_Dialog_Record;
   License            :      String);

function Get_License
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the license information.

```
procedure Set_Logo
  (About              : access Gtk_About_Dialog_Record;
   Logo               :      Gdk.Pixbuf.Gdk_Pixbuf);

function Get_Logo
  (About              : access Gtk_About_Dialog_Record)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Returns the pixbuf displayed as logo in the about dialog.
The returned value is owned by the dialog. If you want to keep a reference to it, you must call Ref on it. Set_Logo sets the pixbuf to be displayed as logo in the about dialog. If it is null, the default window icon set with Gtk.Window.Set_Default_Icon will be used.

```
procedure Set_Logo_Icon_Name
  (About              : access Gtk_About_Dialog_Record;
   Icon_Name          :      String := "");

function Get_Logo_Icon_Name
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the icon name displayed as logo in the about dialog.
If the Icon_Name is set to the empty string, the default window icon set with Gtk.Window.Set_Default_Icon will be used.

```
function Get_Program_Name
  (About              : access Gtk_About_Dialog_Record)
   return String;

procedure Set_Program_Name
  (About              : access Gtk_About_Dialog_Record;
   Name               :      String);
```

Returns or sets the program name displayed in the about dialog.
(since 2.12)

```
procedure Set_Translator_Credits
  (About              : access Gtk_About_Dialog_Record;
   Translator_Credits  :      String);

function Get_Translator_Credits
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Sets the translator credits string which is displayed in
the translators tab of the secondary credits dialog.

The intended use for this string is to display the translator of the language which is currently used in the user interface. Using Gtkada.Intl.Gettext, a simple way to achieve that is to mark the string for translation: Set_Translator_Credits (About, -"translator-credits"); It is a good idea to use the customary msgid "translator-credits" for this purpose, since translators will already know the purpose of that msgid, and since Gtk_About_Dialog will detect if "translator-credits" is untranslated and hide the tab.

```
procedure Set_Version
  (About              : access Gtk_About_Dialog_Record;
   Version            :      String);

function Get_Version
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the version string.

```
procedure Set_Website
  (About              : access Gtk_About_Dialog_Record;
   Website            :      String);

function Get_Website
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the website URL. This URL must start with http:// to be
properly recognized as an hyper link. You must also have called Set_Url_Hook before calling this function.

```
procedure Set_Website_Label
  (About              : access Gtk_About_Dialog_Record;
   Website_Label      :      String);

function Get_Website_Label
  (About              : access Gtk_About_Dialog_Record)
   return String;
```

Returns the label used for the website link. It defaults to the URL.

```
procedure Set_Wrap_License
  (About              : access Gtk_About_Dialog_Record;
   Wrap_License       :      Boolean);
```

```
function Get_Wrap_License
  (About               : access Gtk_About_Dialog_Record)
   return Boolean;
```

Returns whether the license text in About is automatically wrapped.

```
function Set_Email_Hook
  (Func                :      Activate_Link_Func;
   Data                :      System.Address;
   Destroy             :      Glib.G_Destroy_Notify_Address)
   return Activate_Link_Func;
```

Installs a global function to be called whenever the user activates an email link in an about dialog. Return value: the previous email hook.

```
function Set_Url_Hook
  (Func                :      Activate_Link_Func;
   Data                :      System.Address;
   Destroy             :      Glib.G_Destroy_Notify_Address)
   return Activate_Link_Func;
```

Installs a global function to be called whenever the user activates a URL link in an about dialog. Return value: the previous URL hook.

```
procedure Set_Name
  (About               : access Gtk_About_Dialog_Record;
   Name                :      String);
```

```
function Get_Name
  (About               : access Gtk_About_Dialog_Record)
   return String;
```

Get_Name
Returns the program name displayed in the about dialog.

61 Package Gtk.Accel_Group

An accel group represents a group of keyboard accelerators, generally attached to a toplevel window. Accelerators are different from mnemonics. Accelerators are shortcuts for activating a menu item. They appear alongside the menu item they are a shortcut for. Mnemonics are shortcuts for GUI elements, such as buttons. They appear as underline characters. Menu items can have both.

61.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Accel_Group       (Package Gtk.Accel_Group)
```

61.2 Signals

- **"accel_activate"**

  ```
  procedure Handler
  (Group         : access Gtk_Accel_Group_Record'Class;
  Acceleratable : access GObject_Record'Class;
  Keyval        : Gdk_Key_Type;
  Modifier      : Gdk_Modifier_Type);
  ```

  This is an implementation detail, not meant to be used by applications

- **"accel_changed"**

  ```
  procedure Handler
  (Group         : access Gtk_Accel_Group_Record'Class;
  Keyval        : Gdk_Key_Type;
  Modifier      : Gdk_Modifier_Type;
  Closure       : GClosure);
  ```

  Emitted when a Gtk_Accel_Group_Entry is added to or removed from the accel group. Widgets like Gtk_Accel_Label which display an associated accelerator should connect to this signal, and rebuild their visual representation if the accel_closure is theirs.

61.3 Types

type Gtk_Accel_Flags **is new** Guint;

type Gtk_Accel_Group_Activate **is access function**
```
          (Accel_Group   : access Gtk_Accel_Group_Record'Class;
```

type Gtk_Accel_Group_Entry **is new** Gdk.C_Proxy;

type Gtk_Accel_Key **is record**
```
          Accel_Key  : Gdk.Types.Gdk_Key_Type;
          Accel_Mods : Gdk.Types.Gdk_Modifier_Type;
          Flags      : Gtk_Accel_Flags;
```

```
end record;
```

61.4 Subprograms

```
procedure Gtk_New
  (Accel_Group        : out   Gtk_Accel_Group);
function Get_Type             return Glib.GType;
```

Return the internal value associated with a Gtk_Accel_Group.

```
procedure Lock
  (Accel_Group        : access Gtk_Accel_Group_Record);
procedure Unlock
  (Accel_Group        : access Gtk_Accel_Group_Record);
function Get_Is_Locked
  (Accel_Group        : access Gtk_Accel_Group_Record)
   return Boolean;
```

Locks or unlocks the group. When a group is locked, the accelerators contained in it cannot be changed at runtime by the user. See Gtk_Accel_Map.Change_Entry about runtime accelerator changes. Unlock must be called the same number of time that Lock was called.

61.4.1 Groups

```
function Accel_Groups_Activate
  (Object             : access Gtk.Object.Gtk_Object_Record'Class;
   Accel_Key          :      Gdk.Types.Gdk_Key_Type;
   Accel_Mods         :      Gdk.Types.Gdk_Modifier_Type)
   return Boolean;
```

Find the first accelerator in any group, attached to Object that matches the given key and modifier, and activate that accelerator. Returns True if an accelerator was activated.

```
function From_Object
  (Object             : access Gtk.Object.Gtk_Object_Record'Class)
   return Object_List.GSlist;
```

Gets a list of all accel groups which are attached to Object.

61.4.2 Accelerators

```
function Accelerator_Valid
  (Keyval             :      Gdk.Types.Gdk_Key_Type;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type)
   return Boolean;
```

Determines whether a given keyval and modifier constitute a valid accelerator. For instance, GDK_Control_L is not a valid accelerator, whereas Gdk_L associated with Control_Mask is valid.

```
procedure Accelerator_Parse
  (Accelerator        :      String;
   Accelerator_Key    : out   Gdk.Types.Gdk_Key_Type;
   Accelerator_Mods   : out   Gdk.Types.Gdk_Modifier_Type);
```

Parse a string representing an accelerator. The format looks like "<Control>a", "<Shift><Alt>a" or "<Release>z" (the last one applies to a key release. Abbreviations such as "Ctrl" are allowed.

```
function Accelerator_Name
  (Accelerator_Key    :      Gdk.Types.Gdk_Key_Type;
   Accelerator_Mods   :      Gdk.Types.Gdk_Modifier_Type)
   return String;
```

Converts an accelerator keyval and modifier mask into a string parseable by Accelerator_Parse. For example, if you pass in GDK_q and GDK_CONTROL_MASK, this function returns "<Control>q". If you need to display accelerators in the user interface, see Accelerator_Get_Label.

```
function Accelerator_Get_Label
  (Accelerator_Key    :      Gdk.Types.Gdk_Key_Type;
   Accelerator_Mods   :      Gdk.Types.Gdk_Modifier_Type)
   return String;
```

Converts an accelerator keyval and modifier mask into a string which can be used to represent the accelerator to the user.

```
procedure Set_Default_Mod_Mask
  (Default_Mod_Mask   :      Gdk.Types.Gdk_Modifier_Type);

function Get_Default_Mod_Mask  return Gdk.Types.Gdk_Modifier_Type;
```

Sets the modifiers that will be considered significant for keyboard accelerators. The default mod mask is GDK_CONTROL_MASK | GDK_SHIFT_MASK | GDK_MOD1_MASK, that is, Control, Shift, and Alt. Other modifiers will by default be ignored by GtkAccelGroup. You must include at least the three default modifiers in any value you pass to this function.

The default mod mask should be changed on application startup, before using any accelerator groups.

```
function Get_Modifier_Mask
  (Accel_Group        : access Gtk_Accel_Group_Record)
   return Gdk.Types.Gdk_Modifier_Type;
```

Gets a modifier type representing the mask for this Accel_Group. For example, GDK_CONTROL_MASK, GDK_SHIFT_MASK, etc.

62 Package Gtk.Accel_Label

The Gtk_Accel_Label widget is a child of Gtk_Label that also displays an accelerator key on the right of the label text, e.g. 'Ctl+S'. It is commonly used in menus to show the keyboard short-cuts for commands.

The accelerator key to display is not set explicitly. Instead, the Gtk_Accel_Label displays the accelerators which have been added to a particular widget. This widget is set by calling Set_Accel_Widget.

For example, a Gtk_Menu_Item widget may have an accelerator added to emit the "activate" signal when the 'Ctl+S' key combination is pressed. A Gtk_Accel_Label is created and added to the Gtk_Menu_Item, and Set_Accel_Widget is called with the Gtk_Menu_Item as the second argument. The Gtk_Accel_Label will now display 'Ctl+S' after its label.

Note that creating a Gtk_Menu_Item with Gtk_New and a non null "label" parameter (ditto for Gtk_Check_Menu_Item and Gtk_Radio_Menu_Item) automatically adds a Gtk_Accel_Label to the Gtk_Menu_Item and calls Set_Accel_Widget to set it up for you.

A Gtk_Accel_Label will only display accelerators which have the Accel_Visible (see Gtk.Accel_Group) flag set. A Gtk_Accel_Label can display multiple accelerators and even signal names, though it is almost always used to display just one accelerator key.

62.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Misc             (Package Gtk.Misc)
         \___ Gtk_Label         (Package Gtk.Label)
            \___ Gtk_Accel_Label (Package Gtk.Accel_Label)
```

62.2 Subprograms

```
procedure Gtk_New
  (Accel_Label        : out   Gtk_Accel_Label;
   Str                :      UTF8_String);
```

Create a new Gtk_Accel_Label.
Str is the label string.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Accel_Label.

```
procedure Set_Accel_Widget
  (Accel_Label        : access Gtk_Accel_Label_Record;
   Accel_Widget       : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add an accelerator to a particular widget.

```
function Get_Accel_Widget
  (Accel_Label        : access Gtk_Accel_Label_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget monitored by Accel_Label.

```
function Get_Accel_Width
  (Accel_Label        : access Gtk_Accel_Label_Record)
   return Guint;
```

Return the width needed to display the accelerator key(s).
This is used by menus to align all of the Gtk_Menu_Item widgets, and shouldn't be needed by applications.

```
function Refetch
  (Accel_Label        : access Gtk_Accel_Label_Record)
   return Boolean;
```

Recreate the string representing the accelerator keys.
This should not be needed since the string is automatically updated whenever accelerators are added or removed from the associated widget. Always return False.

62.3 Example

```
Creating a simple menu item with an accelerator key.

Save_Item   : Gtk_Menu_Item;
Accel_Group : Gtk_Accel_Group;

--  Create a Gtk_Accel_Group and add it to the window.
Gtk_New (Accel_Group);
Add_Accel_Group (Window, Accel_Group);

--  Create the menu item using the convenience function.
Gtk_New (Save_Item, "Save");
Show (Save_Item);
Add (Menu, Save_Item);

--  Now add the accelerator to the Gtk_Menu_Item. Note that since we called
--  Gtk_New with a label to create the Gtk_Menu_Item the
--  Gtk_Accel_Label is automatically set up to display the Gtk_Menu_Item
--  accelerators. We just need to make sure we use Accel_Visible here.

Add_Accelerator
  (Save_Item, "activate", Accel_Group,
   GDK_S, Control_Mask, Accel_Visible);
```

63 Package Gtk.Accel_Map

An accel_map provides support for loading and saving accelerators (see also Gtk.Accel_Group).

63.1 Signals

- **"changed"**

```
procedure Handler
(Map : Gtk_Accel_Map;
Accel_Path : String;
Accel_Key  : Gdk_Key_Type;
Accel_Mods : Gdk_Modifier_Type);
```

  Notifies of a change in the global accelerator map. The path is also used as the detail for the signal, so it is possible to connect to changed::accel_path.

63.2 Types

type Gtk_Accel_Map_Foreach **is access procedure**

63.3 Subprograms

```
function Get                    return Gtk_Accel_Map;
```

Gets the singleton global Gtk_Accel_Map object. This object is useful only for notification of changes to the accelerator map via the ::changed signal; it isn't a parameter to the other accelerator map functions.

```
function Get_Type               return Glib.GType;
```

Return the internal type used for a Gtk_Accel_Map

```
procedure Save
  (File_Name          :      String);
```

Save the key shortcuts to a file. These are the shortcuts that might have been changed dynamically by the user, if the RC file (see Gtk.RC) contained the line "gtk-can-change-accels=1"

```
procedure Load
  (File_Name          :      String);
```

Load the key shortcuts from a file

```
procedure Add_Entry
  (Accel_Path         :      String;
   Accel_Key          :      Gdk.Types.Gdk_Key_Type;
   Accel_Mods         :      Gdk.Types.Gdk_Modifier_Type);
```

Register a new accelerator for a given menu item, within the global accelerator map. This function should only be called once per Accel_Path. To change it programmatically during runtime, use Change_Entry. Accel_Path is of the form: <app>/Category1/Category2/.../Action", where "app" is a unique, application-specific identifier (for examples of valid Accel_Path, check the file created by Save above).

For instance, the path in the testgtk application for the menu File->Open would be <testgtk>/file/open

Generally, the path need to be set explicitely for an item, through a call to Gtk.Menu_Item.Set_Accel_Path or Gtk.Widget.Set_Accel_Path. However, if the widget is created automatically through a Gtk.Item_Factory, this is done automatically.

It is better to use this function instead of Add_Accelerator, since when the accelerators are changed interactively by the user, the new value will be shown properly in the menu, which wouldn't happen if they had been forced by Add_Accelerator.

```
procedure Lookup_Entry
  (Accel_Path         :      String;
   Key                : out   Gtk.Accel_Group.Gtk_Accel_Key;
   Found              : out   Boolean);
```

Look up the accelerator for Accel_Path, and set Key appropriately. If no accelerator was set, Found is set to False, and the value of Key is meaningless.

```
function Change_Entry
  (Accel_Path         :      String;
   Accel_Key          :      Gdk.Types.Gdk_Key_Type;
   Accel_Mods         :      Gdk.Types.Gdk_Modifier_Type;
   Replace            :      Boolean)
   return Boolean;
```

Change the accelerator currently associated wtih Accel_Path.
A change may not always be possible due to conflicts with other accelerators. Replace should be set to True if other accelerators may be deleted to resolve such conflicts. Returns True if the entry could be changed

```
procedure Lock_Path
  (Accel_Path         :      String);

procedure Unlock_Path
  (Accel_Path         :      String);
```

Locks the given accelerator path. If the accelerator map doesn't yet contain an entry for Accel_Path, a new one is created.

Locking an accelerator path prevents its accelerator from being changed during runtime. A locked accelerator path can be unlocked by Unlock_Path. Refer to Change_Entry for information about runtime accelerator changes.

If called more than once, Accel_Path remains locked until Unlock_Path has been called an equivalent number of times.

Note that locking of individual accelerator paths is independent from locking the Gtk_Accel_Group containing them. For runtime accelerator changes to be possible both the accelerator path and its accel group have to be unlocked.

63.3.1 Foreach

```
procedure Add_Filter
  (Filter_Pattern     :      String);
```

Adds a filter to the global list of accel path filters.
Accel map entries whose accel path matches one of the filters are skipped by Foreach. This function is intended for GTK+ modules that create their own menus, but don't want them to be saved into the applications accelerator map dump.

```
procedure Foreach
  (Data                :      System.Address;
   Func                :      Gtk_Accel_Map_Foreach);
```

Calls Func for each of the currently defined key shortcuts. Data is passed as is to Func

```
procedure Foreach_Unfiltered
  (Data                :      System.Address;
   Func                :      Gtk_Accel_Map_Foreach);
```

Loops over all entries in the accelerator map, and execute Func on each.

64 Package Gtk.Action

Actions represent operations that the user can perform, along with some information on how it should be presented in the interface. Each action provides methods to create icons, menu items and toolbar items representing itself.

As well as the callback that is called when the action gets activated, the following also gets associated with the action: @itemize @bullet @item a name (not translated, for path lookup) @item a label (translated, for display) @item an accelerator @item whether label indicates a stock id @item a tooltip (optional, translated) @item a toolbar label (optional, shorter than label)

@end itemize The action will also have some state information: @itemize @bullet @item visible (shown/hidden) @item sensitive (enabled/disabled)

@end itemize Apart from regular actions, there are toggle actions, which can be toggled between two states and radio actions, of which only one in a group can be in the "active" state. Other actions can be implemented as Gtk_Action subclasses.

Each action can have one or more proxy menu item, toolbar button or other proxy widgets. Proxies mirror the state of the action (text label, tooltip, icon, visible, sensitive, etc), and should change when the action's state changes. When the proxy is activated, it should activate its action.

64.1 Signals

- **"activate"**

```
procedure Handler (Action : access Gtk_Action_Record'Class);
```

  The "activate" signal is emitted when the action is activated.

64.2 Subprograms

```
procedure Gtk_New
  (Action             : out   Gtk_Action;
   Name               :       String;
   Label              :       String;
   Tooltip            :       String := "";
   Stock_Id           :       String := "");
function Convert
  (C_Object           :       System.Address)
   return Gtk_Action;
```

Convert a C object to a Gtk_Action. The type of the C object must match, of course.

```
function Get_Type               return GType;
```

Return the internal value associated with a Gtk_Action

```
procedure Activate
  (Action             : access Gtk_Action_Record);
```

Emits the "activate" signal on the specified action, if it isn't insensitive. This gets called by the proxy widgets when they get activated. It can also be used to manually activate an action.

```
procedure Connect_Accelerator
  (Action             : access Gtk_Action_Record);
```

```
procedure Disconnect_Accelerator
  (Action             : access Gtk_Action_Record);
```

Installs the accelerator for Action if Action has an accel path and group. See Set_Accel_Path and Set_Accel_Group. Since multiple proxies may independently trigger the installation of the accelerator, the Action counts the number of times this function has been called and doesn't remove the accelerator until Disconnect_Accelerator has been called as many times.

```
function Create_Icon
  (Action             : access Gtk_Action_Record;
   Icon_Size          :      Gtk.Enums.Gtk_Icon_Size)
   return Gtk.Widget.Gtk_Widget;
```

This function is intended for use by action implementations to create icons displayed in the proxy widgets. Returns a widget that displays the icon for this action.

```
function Create_Menu
  (Action             : access Gtk_Action_Record)
   return Gtk.Widget.Gtk_Widget;
```

If Action provides a Gtk_Menu widget as a submenu for the menu item or the toolbar item it creates, this function returns an instance of that menu. Since: 2.12

```
function Create_Menu_Item
  (Action             : access Gtk_Action_Record)
   return Gtk.Widget.Gtk_Widget;
```

Creates a menu item widget that proxies for the given action.

```
function Create_Tool_Item
  (Action             : access Gtk_Action_Record)
   return Gtk.Widget.Gtk_Widget;
```

Creates a toolbar item widget that proxies for the given action.

```
procedure Set_Accel_Group
  (Action             : access Gtk_Action_Record;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group
                      := null);
```

Sets the Gtk_Accel_Group in which the accelerator for this action will be installed.

```
procedure Set_Accel_Path
  (Action             : access Gtk_Action_Record;
   Accel_Path         :      String);
function Get_Accel_Path
  (Action             : access Gtk_Action_Record)
   return String;
```

Sets the accel path for this action. All proxy widgets associated with the action will have this accel path, so that their accelerators are consistent.

```
function Get_Name
  (Action             : access Gtk_Action_Record)
   return String;
```

Returns the name of the action.

```
procedure Set_Sensitive
  (Action             : access Gtk_Action_Record;
   Sensitive          :      Boolean);
```

```
function Get_Sensitive
  (Action             : access Gtk_Action_Record)
   return Boolean;
```

Returns whether the action itself is sensitive. Note that this doesn't necessarily mean effective sensitivity. See Is_Sensitive for that.

```
function Is_Sensitive
  (Action             : access Gtk_Action_Record)
   return Boolean;
```

Returns whether the action is effectively sensitive.
Returns True if teh action and its associated action group are both sensitive.

```
procedure Set_Visible
  (Action             : access Gtk_Action_Record;
   Visible            :      Boolean);
function Get_Visible
  (Action             : access Gtk_Action_Record)
   return Boolean;
```

Returns whether the action itself is visible. Note that this doesn't necessarily mean effective visibility. See Is_Visible for that.

```
function Is_Visible
  (Action             : access Gtk_Action_Record)
   return Boolean;
```

Returns whether the action is effectively visible.
Returns True if the action and its associated action group are both visible.

64.2.1 Proxies

```
function Get_Proxies
  (Action             : access Gtk_Action_Record)
   return Gtk.Widget.Widget_SList.GSlist;
```

Returns the proxy widgets for an action. The returned list must not be modified

```
function Gtk_Widget_Get_Action
  (Widget             : access Gtk.Widget.Gtk_Widget_Record)
   return Gtk_Action;
```

Returns the action that Widget is a proxy for.
See also Get_Proxies. Since: 2.10

```
procedure Connect_Proxy
  (Action             : access Gtk_Action_Record;
   Proxy              : access Gtk.Widget.Gtk_Widget_Record'Class);
procedure Disconnect_Proxy
  (Action             : access Gtk_Action_Record;
   Proxy              : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Connects a widget to an action object as a proxy. Synchronises various properties of the action with the widget (such as label text, icon, tooltip, etc), and attaches a callback so that the action gets activated when the proxy widget does. If the widget is already connected to an action, it is disconnected first. Disconnect_Proxy does not destroy the widget.

```
procedure Block_Activate_From
  (Action             : access Gtk_Action_Record;
   Proxy              : access Gtk.Widget.Gtk_Widget_Record'Class);
```

```
procedure Unblock_Activate_From
  (Action             : access Gtk_Action_Record;
   Proxy              : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Disables calls to the Activate function by signals on the given proxy widget. This is used to break notification loops for things like check or radio actions. This function is intended for use by action implementations.

65 Package Gtk.Action_Group

Actions are organised into groups. An action group is essentially a map from names to Gtk_Action objects.

All actions that would make sense to use in a particular context should be in a single group. Multiple action groups may be used for a particular user interface. In fact, it is expected that most nontrivial applications will make use of multiple groups. For example, in an application that can edit multiple documents, one group holding global actions (e.g. quit, about, new), and one group per document holding actions that act on that document (eg. save, cut/copy/paste, etc). Each window's menus would be constructed from a combination of two action groups.

Accelerators are handled by the GTK+ accelerator map. All actions are assigned an accelerator path (which normally has the form "<Actions>/group-name/action-name") and a shortcut is associated with this accelerator path. All menu items and tool items take on this accelerator path. The GTK+ accelerator map code makes sure that the correct shortcut is displayed next to the menu item.

65.1 Signals

- **"connect_proxy"**

  ```
  procedure Handler
  (Group  : access Gtk_Action_Group_Record'Class;
  Action : access Gtk_Action_Record'Class;
  Proxy  : access Gtk_Widget_Record'Class);
  ```

  The connect_proxy signal is emitted after connecting a proxy to an action in the group. Note that the proxy may have been connected to a different action before. This is intended for simple customizations for which a custom action class would be too clumsy, e.g. showing tooltips for menuitems in the statusbar. Gtk_UI_Manager proxies the signal and provides global notification just before any action is connected to a proxy, which is probably more convenient to use.

- **"disconnect_proxy"**

  ```
  procedure Handler
  (Group  : access Gtk_Action_Group_Record'Class;
  Action : access Gtk_Action_Record'Class;
  Proxy  : access Gtk_Widget_Record'Class);
  ```

  The disconnect_proxy signal is emitted after disconnecting a proxy from an action in the group. Gtk_UI_Manager proxies the signal and provides global notification just before any action is connected to a proxy, which is probably more convenient to use.

- **"post_activate"**

  ```
  procedure Handler
  (Group  : access Gtk_Action_Group_Record'Class;
  Action : access Gtk_Action_Record'Class);
  ```

  The post_activate signal is emitted just after the action in the action_group is activated This is intended for Gtk_UI_Manager to proxy the signal and provide global notification just after any action is activated.

- **"pre_activate"**

```
procedure Handler
(Group  : access Gtk_Action_Group_Record'Class;
Action : access Gtk_Action_Record'Class);
```

The post_activate signal is emitted just before the action in the action_group is activated This is intended for Gtk_UI_Manager to proxy the signal and provide global notification just after any action is activated.

65.2 Types

type Action_Callback **is access procedure**

type Action_Entry **is private**;

type Action_Entry_Array **is array** (Natural **range** <>) **of** Action_Entry;

type Radio_Action_Callback **is access procedure**

type Radio_Action_Entry **is private**;

type Toggle_Action_Entry **is private**;

An opaque structure describing an action entry

65.3 Subprograms

```
procedure Gtk_New
  (Group              : out   Gtk_Action_Group;
   Name               :       String);
function Get_Type               return GType;
```

Return the internal value associated with a Gtk_Action_Group.

```
function Get_Action
  (Action_Group       : access Gtk_Action_Group_Record;
   Action_Name        :       String)
   return Gtk.Action.Gtk_Action;
```

Looks up an action in the action group by name, or null if no such action exists.

```
function Get_Name
  (Action_Group       : access Gtk_Action_Group_Record)
   return String;
```

Gets the name of the action group.

```
procedure Set_Sensitive
  (Action_Group       : access Gtk_Action_Group_Record;
   Sensitive          :      Boolean);
function Get_Sensitive
  (Action_Group       : access Gtk_Action_Group_Record)
   return Boolean;
```

Returns True if the group is sensitive. The constituent actions can only be logically sensitive (see Gtk.Action.Is_Sensitive) if they are sensitive (see Gtk.Action.Get_Sensitive) and their group is sensitive.

```
procedure Set_Visible
  (Action_Group       : access Gtk_Action_Group_Record;
   Visible            :      Boolean);
function Get_Visible
  (Action_Group       : access Gtk_Action_Group_Record)
   return Boolean;
```

Returns True if the group is visible. The constituent actions can only be logically visible (see Gtk.Action.Is_Visible) if they are visible (see Gtk.Action.Get_Visible) and their group is visible.

```
function List_Actions
  (Action_Group       : access Gtk_Action_Group_Record)
   return Glib.Object.Object_Simple_List.Glist;
```

Lists the actions in the action group. The returned list must be freed by the user.

```
procedure Set_Translation_Domain
  (Action_Group       : access Gtk_Action_Group_Record;
   Domain             :      String);
```

Sets the translation domain and uses dgettext() for translating the Label and Tooltip of Gtk_Action_Entry's added by Add_Actions.

65.3.1 Adding and removing actions

```
function Create
  (Name               :      String;
   Label              :      String := "";
   Stock_Id           :      String := "";
   Accelerator        :      String := "";
   Tooltip            :      String := "";
   Callback           :      Action_Callback := null)
   return Action_Entry;
```

Create a new Action_Entry. The returned value must be freed by the caller.

```
function Create
  (Name               :      String;
   Label              :      String := "";
   Stock_Id           :      String := "";
   Accelerator        :      String := "";
   Tooltip            :      String := "";
   Callback           :      Action_Callback := null;
   Is_Active          :      Boolean := True)
   return Toggle_Action_Entry;
```

Create a new Action_Entry. The returned value must be freed by the caller. Is_Active is the initial state of the button.

```
function Create
  (Name               :      String;
   Label              :      String;
   Stock_Id           :      String := "";
   Accelerator        :      String := "";
   Tooltip            :      String := "";
   Value              :      Glib.Gint)
   return Radio_Action_Entry;
```

Create a new Radio_Action_Entry. Value is the value set on the radio action (see Gtk.Radio_Action.Get_Current_Value)

```
procedure Free
  (Action             : in out Action_Entry);

procedure Free
  (Actions            : in out Action_Entry_Array);

procedure Free
  (Action             : in out Radio_Action_Entry);

procedure Free
  (Actions            : in out Radio_Action_Entry_Array);

procedure Free
  (Action             : in out Toggle_Action_Entry);

procedure Free
  (Actions            : in out Toggle_Action_Entry_Array);
```

Free Action and Actions

```
procedure Add_Action
  (Action_Group       : access Gtk_Action_Group_Record;
   Action             : access Gtk.Action.Gtk_Action_Record'Class);
```

Adds an action object to the action group. Note that this function does not set up the accel path of the action, which can lead to problems if a user tries to modify the accelerator of a menuitem associated with the action. Therefore you must either set the accel path yourself with Gtk.Action.Set_Accel_Path, or use Add_Action_With_Accel.

```
procedure Remove_Action
  (Action_Group       : access Gtk_Action_Group_Record;
   Action             : access Gtk.Action.Gtk_Action_Record'Class);
```

Removes an action object from the action group.

```
procedure Add_Action_With_Accel
  (Action_Group       : access Gtk_Action_Group_Record;
   Action             : access Gtk.Action.Gtk_Action_Record'Class;
   Accelerator        :      String := "");
```

Adds an action object to the action group and sets up the accelerator. If Accelerator is unspecified, attempts to use the accelerator associated with the stock_id of the action. Accel paths are set to <Actions>/group-name/action-name.

```
procedure Add_Actions
  (Action_Group       : access Gtk_Action_Group_Record;
   Entries            :      Action_Entry_Array;
   User_Data          :      System.Address
                      := System.Null_Address;
   Destroy            :      Glib.G_Destroy_Notify_Address
                      := null);
```

This is a convenience function to create a number of actions and add them to the action group. Destroy is called when User_Data is no longer needed.

The "activate" signals of the actions are connected to the callbacks in Entries, and their accel paths are set to <Actions>/group-name/action-name.

```
procedure Add_Radio_Actions
  (Action_Group       : access Gtk_Action_Group_Record;
   Entries            :      Radio_Action_Entry_Array;
   Value              :      Glib.Gint;
   On_Change          :      Radio_Action_Callback;
   User_Data          :      System.Address
                      := System.Null_Address;
   Destroy            :      Glib.G_Destroy_Notify_Address
                      := null);
```

This is a convenience routine to create a group of radio actions and add them to the action group.

The "changed" signal of the first radio action is connected to the On_Change callback and the accel paths of the actions are set to <Actions>/group-name/action-name

Value is the value of the action to activate initially, or -1 if no action should be activated. Destroy is called when User_Data is no longer necessary.

```
procedure Add_Toggle_Actions
  (Action_Group       : access Gtk_Action_Group_Record;
   Entries            :      Toggle_Action_Entry_Array;
   User_Data          :      System.Address
                      := System.Null_Address;
   Destroy            :      Glib.G_Destroy_Notify_Address
                      := null);
```

This is a convenience function to create a number of toggle actions and add them to the action group. The "activate" signals of the actions are connected to the callbacks and their accel paths are set to <Actions>/group-name/action-name. Destroy is called when User_Data is no longer necessary.

66 Package Gtk.Adjustment

This object represents an adjustable bounded value. It is used in many other widgets that have such internal values, like Gtk_Scrollbar, Gtk_Spin_Button, Gtk_Range, ... Modifying the value of these widgets is done through their associated adjustments.

The modification of the value is left to the user, who should call Value_Changed or Changed to emit the relevant signals.

The meaning of the most important fields can be explained on the following figure (imagine this is a scrollbar): @example

[——-|==========|————-] lower value value + page_size upper @end example

66.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Adjustment        (Package Gtk.Adjustment)
```

66.2 Signals

- **"changed"**

  ```
  procedure Handler (Adjustment : access Gtk_Adjustment_Record'Class);
  ```

  This signal is emitted every time one of the parameters is modified, except the value.

- **"value_changed"**

  ```
  procedure Handler (Adjustment : access Gtk_Adjustment_Record'Class);
  ```

  This signal is emitted every time the value of the adjustment is modified

66.3 Subprograms

```
procedure Gtk_New
  (Adjustment         : out   Gtk_Adjustment;
   Value              :       Gdouble;
   Lower              :       Gdouble;
   Upper              :       Gdouble;
   Step_Increment     :       Gdouble;
   Page_Increment     :       Gdouble;
   Page_Size          :       Gdouble := 0.0);
```

Create a new adjustment.
Value is the initial value of the adjustment. It must be in the range (Lower .. Upper) and the adjustment's value will never be outside this range. Step_Increment is the value used to make minor adjustments, such as when the user clicks on the arrows of a scrollbar. Page_Increment is used to make major adjustments, such as when the user clicks in the through on a scrollbar. Page_Size is deprecated, use the default value.

```
procedure Configure
  (Adjustment         : access Gtk_Adjustment_Record;
   Value              :       Gdouble;
   Lower              :       Gdouble;
   Upper              :       Gdouble;
   Step_Increment     :       Gdouble;
   Page_Increment     :       Gdouble;
```

```
   Page_Size          :      Gdouble);
```

Sets all properties of the adjustment at once.
Use this function to avoid multiple emissions of the "changed" signal. See Set_Lower for an alternative way of compressing multiple emissions of "changed" into one. Since: 2.14

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Adjustment.

```
procedure Set_Value
  (Adjustment         : access Gtk_Adjustment_Record;
   Value              :      Gdouble);
function Get_Value
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the current value of the adjustment.
You do not need to call Value_Changed after modifying this value, this is done automatically.

```
procedure Set_Lower
  (Adjustment         : access Gtk_Adjustment_Record;
   Lower              :      Gdouble);
function Get_Lower
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the lower bound of the adjustment.
When setting multiple adjustment properties via their individual setters, multiple "changed" signals will be emitted. However, since the emission of the "changed" signal is tied to the emission of the "GObject::notify" signals of the changed properties, it's possible to compress the "changed" signals into one by calling Glib.Object.Freeze_Notify and Glib.Object.Thaw_Notify around the calls to the individual setters. Alternatively, using Configure has the same effect of compressing "changed" emissions.

```
procedure Set_Upper
  (Adjustment         : access Gtk_Adjustment_Record;
   Upper              :      Gdouble);
function Get_Upper
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the upper bound of the adjustment.
You should call Changed after modifying this value.

```
procedure Set_Step_Increment
  (Adjustment         : access Gtk_Adjustment_Record;
   Step_Increment     :      Gdouble);
function Get_Step_Increment
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the step increment of the adjustment.
You should call Changed after modifying this value.

```
procedure Set_Page_Increment
  (Adjustment         : access Gtk_Adjustment_Record;
   Page_Increment     :      Gdouble);
function Get_Page_Increment
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the page increment of the adjustment.
You should call Changed after modifying this value.

```
procedure Set_Page_Size
  (Adjustment         : access Gtk_Adjustment_Record;
   Page_Size          :      Gdouble);
function Get_Page_Size
  (Adjustment         : access Gtk_Adjustment_Record)
   return Gdouble;
```

Modify the page size of the adjustment.
You should call Changed after modifying this value.

66.3.1 Misc functions

```
procedure Clamp_Page
  (Adjustment         : access Gtk_Adjustment_Record;
   Lower              :      Gdouble;
   Upper              :      Gdouble);
```

Update the Adjustment value to ensure that the range between Lower and Upper is in the current page (i.e. between value and value + page_size). If the range is larger than the page size, then only the start of it will be in the current page. A "value_changed" signal will be emitted if the value is changed.

66.3.2 Signals emission

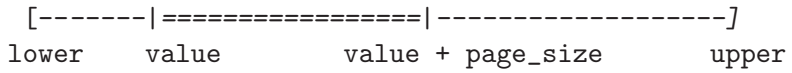
```
procedure Changed
  (Adjustment         : access Gtk_Adjustment_Record);
```

Emit the "changed" signal on Adjustment.
This warns any listener that some field other than the value has been changed.

```
procedure Value_Changed
  (Adjustment         : access Gtk_Adjustment_Record);
```

Emit the "value_changed" signal on Adjustment.
This warns any listener that the value has been changed.

66.4 Example



67 Package Gtk.Alignment

A Gtk_Alignment controls the size and alignment of its single child inside the area allocated to the alignment widget.

The scale/size settings indicate how much the child will expand to fill the container. The values should be in the range 0.0 (no expansion) to 1.0 (full expansion). Note that the scale only indicates the minimal size for the child, it does not force an absolute size.

The alignment settings indicate where in the alignment widget the child should be located. The values are in the range 0.0 (top or left) to 1.0 (bottom or right). These settings are irrelevant if the child is fully expanded.

67.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Alignment (Package Gtk.Alignment)
```

67.2 Subprograms

```
procedure Gtk_New
  (Alignment          : out   Gtk_Alignment;
   Xalign             :       Gfloat;
   Yalign             :       Gfloat;
   Xscale             :       Gfloat;
   Yscale             :       Gfloat);
```

Create a new alignment widget, with initial values for the settings.
See the description of the settings above.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Alignment.

```
procedure Set
  (Alignment          : access Gtk_Alignment_Record;
   Xalign             :       Gfloat;
   Yalign             :       Gfloat;
   Xscale             :       Gfloat;
   Yscale             :       Gfloat);
```

Modify the settings for the alignment.
See the description of the settings above.

```
function Get_Xalign
  (Alignment          : access Gtk_Alignment_Record)
   return Gfloat;
```

Return the X alignment value.
Its value is in the range 0.0 .. 1.0, from left to right.

```
function Get_Yalign
  (Alignment          : access Gtk_Alignment_Record)
   return Gfloat;
```

Return the Y alignment value.
Its value is in the range 0.0 .. 1.0, from top to bottom.

```
function Get_Xscale
  (Alignment          : access Gtk_Alignment_Record)
   return Gfloat;
```

Return the X expansion value, in the range 0.0 .. 1.0.
0.0 means no expansion while 1.0 means full expansion.

```
function Get_Yscale
  (Alignment          : access Gtk_Alignment_Record)
   return Gfloat;
```

Return the Y expansion value, in the range 0.0 .. 1.0
0.0 means no expansion while 1.0 means full expansion.

```
procedure Set_Padding
  (Alignment          : access Gtk_Alignment_Record;
   Padding_Top        :      Guint;
   Padding_Bottom     :      Guint;
   Padding_Left       :      Guint;
   Padding_Right      :      Guint);
procedure Get_Padding
  (Alignment          : access Gtk_Alignment_Record;
   Padding_Top        : out    Guint;
   Padding_Bottom     : out    Guint;
   Padding_Left       : out    Guint;
   Padding_Right      : out    Guint);
```

Sets the padding on the different sides of the widget.
The padding adds blank space to the sides of the widget. For instance, this can be used to indent the child widget towards the right by adding padding on the left.

68 Package Gtk.Arguments

This package is obsolete and replaced by Glib.Values. Future versions of GtkAda will no longer provide this package.

This package provides a convenient interface to C, providing easy conversion from a C's (void*) pointer to any Ada type used in GtkAda. Although this package has been designed to be easily reusable by being as general as possible, these functions are mainly used when writing callbacks and/or marshallers (see Gtk.Marshallers and Gtk.Handlers).

Therefore, the main type in this package is Gtk_Args, which is the equivalent of the C's (GtkArg*) array, i.e an array of unions. This package provides functions to extract the values from this type.

69 Package Gtk.Arrow

Gtk_Arrow should be used to draw simple arrows that need to point in one of the four cardinal directions (up, down, left, or right). The style of the arrow can be one of shadow in, shadow out, etched in, or etched out. Note that these directions and style types may be ammended in versions of Gtk to come.

Gtk_Arrow will fill any space alloted to it, but since it is inherited from Gtk_Misc, it can be padded and/or aligned, to fill exactly the space you desire.

Arrows are created with a call to Gtk_New. The direction or style of an arrow can be changed after creation by using Set.

69.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Misc           (Package Gtk.Misc)
         \___ Gtk_Arrow       (Package Gtk.Arrow)
```

69.2 Subprograms

```
procedure Gtk_New
  (Arrow              : out   Gtk_Arrow;
   Arrow_Type         :       Gtk.Enums.Gtk_Arrow_Type;
   Shadow_Type        :       Gtk.Enums.Gtk_Shadow_Type);
```

Create a new arrow widget.

```
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Arrow.

```
procedure Set
  (Arrow              : access Gtk_Arrow_Record;
   Arrow_Type         :       Gtk.Enums.Gtk_Arrow_Type;
   Shadow_Type        :       Gtk.Enums.Gtk_Shadow_Type);
```

Set the direction and style of the Arrow.

70 Package Gtk.Aspect_Frame

A Gtk_Aspect_Frame is the same type of widget as a frame, but it constrains its child to a specific aspect ratio between its width and its height.

This ratio can either be given explicitly by the user, or chosen from the widget's initial size request (might be different from the one if was actually given).

70.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Frame    (Package Gtk.Frame)
               \___ Gtk_Aspect_Frame (Package Gtk.Aspect_Frame)
```

70.2 Subprograms

```
procedure Gtk_New
  (Aspect_Frame       : out   Gtk_Aspect_Frame;
   Label              :       UTF8_String;
   Xalign             :       Gfloat;
   Yalign             :       Gfloat;
   Ratio              :       Gfloat;
   Obey_Child         :       Boolean);
```

Create a new Aspect_Frame.
If Label is the empty string, then the frame won't have any title. Xalign and Yalign are constrained to the range 0.0 .. 1.0 and specify the alignment of the child inside the frame (0.0 means either left or top aligned, 1.0 means right or bottom aligned). Ratio is the ratio width/height for the child of the frame. If Obey_Child is True, then Ratio is ignored and the effective ratio is taken from the child's requisition (ie the ideal size it asked for at creation time).

```
function Get_Type                return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Aspect_Frame.

```
procedure Set
  (Aspect_Frame       : access Gtk_Aspect_Frame_Record;
   Xalign             :       Gfloat;
   Yalign             :       Gfloat;
   Ratio              :       Gfloat;
   Obey_Child         :       Boolean);
```

Modify the frame's parameters (see the description of these parameters for Gtk_New.

```
function Get_Ratio
  (Aspect_Frame       : access Gtk_Aspect_Frame_Record)
   return Gfloat;
```

Return the current ratio for the frame (width / height)

```
function Get_Xalign
  (Aspect_Frame       : access Gtk_Aspect_Frame_Record)
   return Gfloat;
```

Return the current X alignment for the frame.
0.0 means the child is left aligned, 1.0 that it is right aligned.

```
function Get_Yalign
  (Aspect_Frame       : access Gtk_Aspect_Frame_Record)
   return Gfloat;
```

Return the current Y alignment for the frame.
1.0 means the child is top aligned, 1.0 that it is bottom aligned.

71 Package Gtk.Bin

Base class for containers that have only one child. This widget can not be instantiated directly.

71.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
```

71.2 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Bin.

```
function Get_Child
  (Bin                : access Gtk_Bin_Record)
   return Gtk_Widget;
```

Return the child associated with Bin.

72 Package Gtk.Bindings

Gtk_Bindings provides a mechanism for configuring Gtk+ key bindings through RC files. This eases key binding adjustments for application developers as well as users and provides Gtk+ users or administrators with high key binding configurability which requires no application or toolkit side changes.

Installing a key binding ==========

A resource file binding consists of a 'binding' definition and a match statement to apply the binding to specific widget types. Details on the matching mechanism are described under Pathnames and patterns. Inside the binding definition, key combinations are bound to specific signal emissions on the target widget. Key combinations are strings consisting of an optional Gdk_Modifier_Type name and key names such as those defined in Gdk.Types.Keysyms or returned from gdk_keyval_name(), they have to be parsable by gtk_accelerator_parse(). Specifications of signal emissions consist of a string identifying the signal name, and a list of signal specific arguments in parenthesis. For example for binding Control and the left or right cursor keys of a Gtk_Entry widget to the Gtk_Entry::move-cursor signal, so movement occurs in 3 letter steps, the following binding can be used:

binding "MoveCursor3" { bind "<Control>Right" { "move-cursor" (visual-positions, 3, 0) } bind "<Control>Left" { "move-cursor" (visual-positions, -3, 0) } } class "GtkEntry" binding "MoveCursor3"

Unbinding existing key bindings ==========

Gtk+ already defines a number of useful bindings for the widgets it provides. Because custom bindings set up in RC files take precedence over the default bindings shipped with Gtk+, overriding existing bindings as demonstrated in Installing a key binding works as expected. The same mechanism can not be used to "unbind" existing bindings, however.

binding "MoveCursor3" { bind "<Control>Right" { } bind "<Control>Left" { } } class "GtkEntry" binding "MoveCursor3"

The above example will not have the desired effect of causing "<Control>Right" and "<Control>Left" key presses to be ignored by Gtk+. Instead, it just causes any existing bindings from the bindings set "MoveCursor3" to be deleted, so when "<Control>Right" or "<Control>Left" are pressed, no binding for these keys is found in binding set "MoveCursor3". Gtk+ will thus continue to search for matching key bindings, and will eventually lookup and find the default Gtk+ bindings for entries which implement word movement. To keep Gtk+ from activating its default bindings, the "unbind" keyword can be used like this:

binding "MoveCursor3" { unbind "<Control>Right" unbind "<Control>Left" } class "GtkEntry" binding "MoveCursor3"

Now, Gtk+ will find a match when looking up "<Control>Right" and "<Control>Left" key presses before it resorts to its default bindings, and the match instructs it to abort ("unbind") the search, so the key presses are not consumed by this widget. As usual, further processing of the key presses, e.g. by an entries parent widget, is now possible.

72.1 Types

type Gtk_Binding_Set **is new** Glib.C_Proxy;

A binding set maintains a list of activatable key bindings. A single binding set can match multiple types of widgets. Similar to styles, widgets can be mapped by widget name paths, widget class paths or widget class types. When a binding within a set is matched upon activation, an action signal is emitted on the target widget to carry out the actual activation.

72.2 Subprograms

```
function Binding_Set_New
  (Set_Name           :      String)
   return Gtk_Binding_Set;
```

Gtk+ maintains a global list of binding sets. Each binding set has a unique name which needs to be specified upon creation.

```
function Binding_Set_By_Class
  (Object_Class       :      Glib.Object.GObject_Class)
   return Gtk_Binding_Set;
```

This function returns the binding set named after the type name of the passed in class structure. New binding sets are created on demand by this function.

```
function Binding_Set_Find
  (Set_Name           :      String)
   return Gtk_Binding_Set;
```

Find a binding set by its globally unique name. The set_name can either be a name used for Binding_Set_New or the type name of a class used in Binding_Set_By_Class.

```
function Activate
  (Object             : access Gtk.Object.Gtk_Object_Record'Class;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type)
   return Boolean;
```

Find a key binding matching keyval and modifiers and activate the binding on object.

```
function Activate_Event
  (Object             : access Gtk.Object.Gtk_Object_Record;
   Event              :      Gdk.Event.Gdk_Event_Key)
   return Boolean;
```

Looks up key bindings for Object to find one matching Event, and if one was found, activate it. Return value: True if a matching key binding was found

```
function Binding_Set_Activate
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type;
   Object             : access Gtk.Object.Gtk_Object_Record'Class)
   return Boolean;
```

Find a key binding matching keyval and modifiers within binding_set and activate the binding on object.

```
procedure Binding_Entry_Skip
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
```

```
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type);
```

Install a binding on Binding_Set which causes key lookups
to be aborted, to prevent bindings from lower priority sets to be activated. Since: 2.12

```
procedure Add_Signal
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type;
   Signal_Name        :      String);
procedure Add_Signal
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type;
   Signal_Name        :      String;
   Arg1               :      Gint);
procedure Add_Signal
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type;
   Signal_Name        :      String;
   Arg1               :      Boolean);
procedure Add_Signal
  (Binding_Set        :      Gtk_Binding_Set;
   Keyval             :      Guint;
   Modifiers          :      Gdk.Types.Gdk_Modifier_Type;
   Signal_Name        :      String;
   Arg1               :      Gint;
   Arg2               :      Gint);
```

Override or install a new key binding for keyval with modifiers on
binding_set. When the binding is activated, signal_name will be emitted on the target widget, with the given arguments as argument.

73 Package Gtk.Box

A box is a container that can have multiple children, organized either horizontally or vertically. Two subtypes are provided, Gtk_Hbox and Gtk_Vbox, to conform to the C API. In Ada, you do not need to distinguish between the two, but note that the Gtk_Box type is conceptually an abstract type: there is no way to create a "Gtk_Box", only ways to create either an horizontal box, or a vertical box.

Children can be added to one of two positions in the box, either at the beginning (ie left or top) or at the end (ie right or bottom). Each of these positions can contain multiple widgets.

Every time a child is added to the start, it is placed to the right (resp. the bottom) of the previous widget added to the start.

Every time a child is added to the end, it is placed to the left (resp. the top) of the previous widget added to the end.

There are a number of parameters to specify the behavior of the box when it is resized, and how the children should be reorganized and/or resized.

See the testgtk example in the GtkAda distribution to see concrete examples on how all the parameters for the boxes work.

73.1 Types

subtype Gtk_Hbox **is** Gtk_Box;

subtype Gtk_Vbox **is** Gtk_Box;

73.2 Subprograms

```
procedure Gtk_New_Vbox
  (Box               : out   Gtk_Box;
   Homogeneous       :       Boolean := False;
   Spacing           :       Gint := 0);
```

Create a new vertical box.
Its children will be placed one above the other. If Homogeneous is True, all the children will be allocated exactly the same screen real-estate, whereas if it is False, each child can have its own size. Spacing is the space left between two adjacent children.

```
procedure Gtk_New_Hbox
  (Box               : out   Gtk_Box;
   Homogeneous       :       Boolean := False;
   Spacing           :       Gint := 0);
```

Create a new horizontal box.
Its children will be placed one besides the other. If Homogeneous is True, all the children will be allocated exactly the same screen real-estate, whereas if it is False, each child can have its own size. Spacing is the space left between two adjacent children.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Box.

```
function Get_Hbox_Type          return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_HBox.

```
function Get_Vbox_Type          return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_VBox.

```
procedure Pack_Start
  (In_Box             : access Gtk_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             :      Boolean := True;
   Fill               :      Boolean := True;
   Padding            :      Gint := 0);
```

Add a new child to the beginning of the box (ie left or top part).
It is added to the right (resp. the bottom) of the previous child added to the beginning of the box. Note that a child added to the beginning of the box will always remain on the left (resp. top) of all the children added to the end of the box.

If Expand is False, the size allocated for each size will be the one requested by the widget (or the largest child if Homogeneous was set to true when the box was created). Otherwise, the total size of the box is divided between all the children. Note that this does not mean that the children have to occupy all the space given to them...

If Fill is True, then the widget will be resized so as to occupy all the space allocated to them. This is only relevant if Expand is True, since otherwise the space allocated is the same one as the child's size.

Padding is the amount of space left around the widget when it is drawn.

```
procedure Pack_End
  (In_Box             : access Gtk_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             :      Boolean := True;
   Fill               :      Boolean := True;
   Padding            :      Gint := 0);
```

Add a new child to the end of the box (ie right or bottom part).
It is added to the left (resp. top) of the previous child added to the end of the box. Note that a child added to the end of the box will always remain on the right (resp. bottom) of all the children added to the beginning of the box.

See Pack_Start for an explanation of all the parameters.

```
procedure Set_Homogeneous
  (In_Box             : access Gtk_Box_Record;
   Homogeneous        :      Boolean);

function Get_Homogeneous
  (In_Box             : access Gtk_Box_Record)
   return Boolean;
```

Modify or get the homogeneous parameter for the box.
If the box is homogeneous, then all its children will be allocated the same amount of space, even if they are not resized to occupy it (depending on the parameters given to Pack_Start and Pack_End).

```
procedure Set_Spacing
  (In_Box             : access Gtk_Box_Record;
   Spacing            :      Gint);
```

```
function Get_Spacing
  (In_Box             : access Gtk_Box_Record)
   return Gint;
```

Modify the spacing for the box.
I.e. the amount of space left between two adjacent children.

```
procedure Reorder_Child
  (In_Box             : access Gtk_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Pos                :      Guint);
```

Move the Child to a new position.
Nothing is done if Child is not in the box. Pos starts at 0, and indicates the position of the child relative to all other children, no matter where they were packed (the beginning or the end of the box).

```
procedure Set_Child_Packing
  (In_Box             : access Gtk_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             :      Boolean;
   Fill               :      Boolean;
   Padding            :      Gint;
   Pack_Type          :      Gtk.Enums.Gtk_Pack_Type);
procedure Query_Child_Packing
  (In_Box             : access Gtk_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             : out    Boolean;
   Fill               : out    Boolean;
   Padding            : out    Gint;
   PackType           : out    Gtk.Enums.Gtk_Pack_Type);
```

Get information on how the child was packed in the box.
The results are undefined if Child is not in the box.

```
function Get_Child
  (Box                : access Gtk_Box_Record;
   Num                :      Gint)
   return Gtk.Widget.Gtk_Widget;
```

Return the Num-th child of the box, or null if there is no such child.

74 Package Gtk.Button

This package implements a general button widget. This button can be clicked on by the user to start any action. This button does not have multiple states, it can just be temporarily pressed while the mouse is on it, but does not keep its pressed state.

The gtk+ sources provide the following drawing that explains the role of the various spacings that can be set for a button:

74.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Button   (Package Gtk.Button)
```

74.2 Signals

- **"activate"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  You should emit this signal to force the activation of this widget, as if the user has clicked on it. This is generally only useful when bound to a key binding.

- **"clicked"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  Emitted when the button has been clicked on by the user. This is the signal you should use to start your own actions.

- **"enter"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  Emitted when the mouse enters the button. The clicked signal can only be emitted when the mouse is inside the button.

- **"leave"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  Emitted when the mouse leaves the button.

- **"pressed"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  Emitted when the user presses the mouse button on the widget. The default implementation modifies the widget state and its visual aspect.

- **"released"**

  ```
  procedure Handler (Button : access Gtk_Button_Record'Class);
  ```

  Emitted when the user releases the mouse button and is inside of the widget. The default implementation modifies the widget state and its visual aspect. If the mouse is still inside the widget, then the "clicked" signal is emitted.

74.3 Subprograms

```
procedure Gtk_New
  (Button             : out   Gtk_Button;
   Label              :       UTF8_String := "");
```

Create a new button.
if Label is not the empty string, then the text appears in the button (and the child of the button is a Gtk_Label). On the other hand, if Label is the empty string, then no child is created for the button and it is your responsibility to add one. This is the recommended way to put a pixmap inside the button.

```
procedure Gtk_New_From_Stock
  (Button             : out   Gtk_Button;
   Stock_Id           :       String);
```

Create a new button containing the image and text from a stock item.
Some stock ids have predefined contants like Gtk.Stock.Stock_OK or Gtk.Stock.Stock_Apply. See Gtk.Stock for a complete list of predefined stock items. Stock_Id: the name of the stock item.

```
procedure Gtk_New_With_Mnemonic
  (Button             : out   Gtk_Button;
   Label              :       UTF8_String);
```

Create a new button containing a label.
Label: The text of the button, with an underscore in front of the mnemonic character If characters in Label are preceded by an underscore, they are underlined indicating that they represent a keyboard accelerator called a mnemonic. Pressing Alt and that key activates the button.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Button.

```
procedure Set_Relief
  (Button             : access Gtk_Button_Record;
   New_Style          :       Gtk.Enums.Gtk_Relief_Style);
function Get_Relief
  (Button             : access Gtk_Button_Record)
   return Gtk.Enums.Gtk_Relief_Style;
```

Modify the relief style for the button.
This modifies only its visual aspect, not its behavior.

```
procedure Set_Label
  (Button             : access Gtk_Button_Record;
   Label              :       UTF8_String);
function Get_Label
  (Button             : access Gtk_Button_Record)
   return UTF8_String;
```

Set or gets the label of the button.
This text is also used to select an icon if Set_Use_Stock was called with a parameter set to True.

```
procedure Set_Use_Underline
  (Button             : access Gtk_Button_Record;
   Use_Underline      :       Boolean);
function Get_Use_Underline
  (Button             : access Gtk_Button_Record)
   return Boolean;
```

Sets whether an underscore used in the button's label designates an
accelerator. If True, then if the user presses alt and the character following the underscore, then the button will act as if it had been pressed.

```
procedure Set_Use_Stock
  (Button             : access Gtk_Button_Record;
   Use_Stock          :      Boolean);
function Get_Use_Stock
  (Button             : access Gtk_Button_Record)
   return Boolean;
```

Sets or Gets whether a stock item is used by the button.

```
procedure Set_Alignment
  (Button             : access Gtk_Button_Record;
   Xalign             :      Gfloat := 0.5;
   Yalign             :      Gfloat := 0.5);
procedure Get_Alignment
  (Button             : access Gtk_Button_Record;
   Xalign             : out   Gfloat;
   Yalign             : out   Gfloat);
```

Specify the alignment of the label inside the button.
Passing (0.0, 0.0) indicates the label should be at the top-left corner of the button. (0.5, 0.5) indicates that the label should be centered. This property has no effect unless the button's child is a child of Gtk_Alignment or Gtk_Misc

```
procedure Set_Focus_On_Click
  (Button             : access Gtk_Button_Record;
   Focus_On_Click     :      Boolean := True);
function Get_Focus_On_Click
  (Button             : access Gtk_Button_Record)
   return Boolean;
```

Sets whether the button will grab focus when it is clicked with the
mouse. Setting Focus_On_Click to False is useful in contexts like toolbars where the focus should not be removed from the main area of the application.

```
procedure Set_Image
  (Button             : access Gtk_Button_Record;
   Image              : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Image
  (Button             : access Gtk_Button_Record)
   return Gtk.Widget.Gtk_Widget;
```

Set the image of the button.
You do not need to call Show on the image yourself. This settings might have no effect, depending on the theme configuration that the application's user is using (in particular, the setting "gtk-button-images" indicates whether or not images should be displayed in buttons).

```
function Get_Image_Position
  (Button             : access Gtk_Button_Record)
   return Gtk.Enums.Gtk_Position_Type;
procedure Set_Image_Position
  (Button             : access Gtk_Button_Record;
   Position           :      Gtk.Enums.Gtk_Position_Type);
```

Gets the position of the image relative to the text
inside the button. Since: 2.10

74.3.1 Signals emission

```
procedure Pressed
  (Button            : access Gtk_Button_Record);
```

Send the "pressed" signal to the button

```
procedure Released
  (Button            : access Gtk_Button_Record);
```

Send the "release" signal to the button

```
procedure Clicked
  (Button            : access Gtk_Button_Record);
```

Send the "clicked" signal to the button

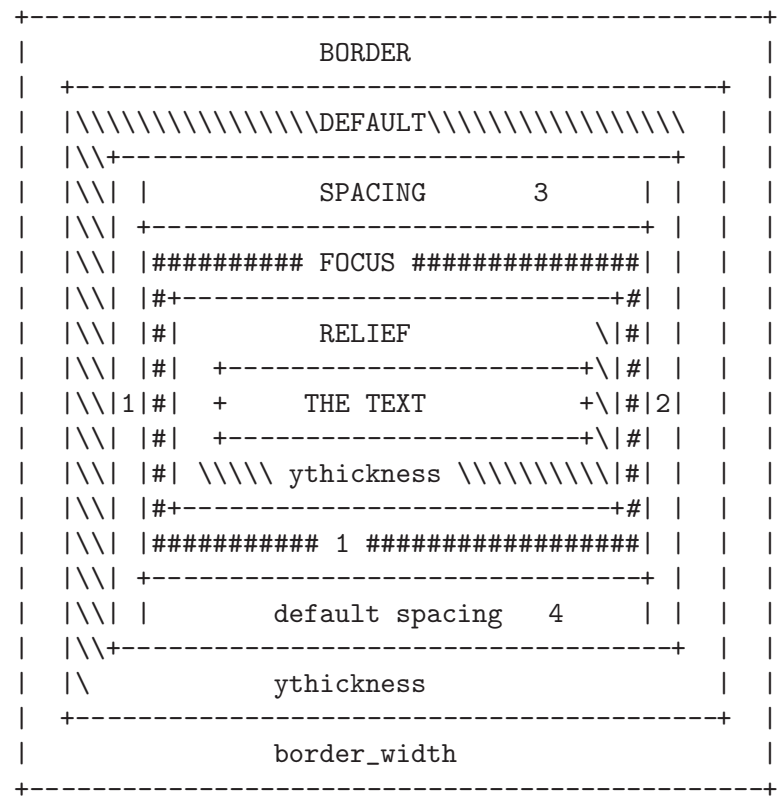
```
procedure Enter
  (Button            : access Gtk_Button_Record);
```

Send the "enter" signal to the button

```
procedure Leave
  (Button            : access Gtk_Button_Record);
```

Send the "leave" signal to the button

74.4 Example



75 Package Gtk.Button_Box

A Gtk_Button_Box is a special type of Gtk_Box specially tailored to contain buttons.

This is only a base class for Gtk_Hbutton_Box and Gtk_Vbutton_Box which provide a way to arrange their children horizontally (resp. vertically). You can not instantiate a Gtk_Button_Box directly, and have to use one the above two instead.

75.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Box          (Package Gtk.Box)
            \___ Gtk_Button_Box (Package Gtk.Button_Box)
```

75.2 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Button_Box.

```
procedure Set_Layout
  (Button_Box         : access Gtk_Button_Box_Record;
   Layout_Style       :      Enums.Gtk_Button_Box_Style);
function Get_Layout
  (Button_Box         : access Gtk_Button_Box_Record)
   return Enums.Gtk_Button_Box_Style;
```

Set the layout to use for the box.
There are four such styles:

- Buttonbox_Spread: The children are spread regularly across the box
- Buttonbox_Edge : Same as Spread, except that the first and last children are aligned on the border of the box.
- Buttonbox_Start : The children are put as much to the left (resp. top) as possible in the box.
- Buttonbox_End : The children are put as much to the right (resp. bottom) as possible in the box.

```
procedure Set_Child_Secondary
  (Button_Box         : access Gtk_Button_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Is_Secondary       :      Boolean);
function Get_Child_Secondary
  (Widget             : access Gtk_Button_Box_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Boolean;
```

Set whether Child should appear in a secondary group of children.
A typical use of a secondary child is the help button in a dialog.

This group appears after the other children if the style is Buttonbox_Start, Buttonbox_Spread or Buttonbox_Edge, and before the other children if the style is Buttonbox_End. For horizontal button boxes, the definition of before/after depends on direction of the widget. (See Gtk.Widget.Set_Direction) If the style is Buttonbox_Start, or Buttonbox_End,

then the secondary children are aligned at the other end of the button box from the main children. For the other styles, they appear immediately next to the main children.

Is_Secondary: if True, the Child appears in a secondary group of the button box.

76 Package Gtk.Calendar

Gtk_Calendar is a widget that displays a calendar, one month at a time. It can be created with Gtk_New.

The month and year currently displayed can be altered with Select_Month. The exact day can be selected from the displayed month using Select_Day.

The way in which the calendar itself is displayed can be altered using Display_Options.

The selected date can be retrieved from a Gtk_Calendar using Get_Date.

If performing many 'mark' operations, the calendar can be frozen to prevent flicker, using Freeze, and 'thawed' again using Thaw.

76.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Calendar       (Package Gtk.Calendar)
```

76.2 Signals

- **"day_selected"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user selects a day on a calendar.

- **"day_selected_double_click"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user double clicks a day on a calendar.

- **"month_changed"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user clicks a button to change the selected month on calendar.

- **"next_month"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user selects the next month on a calendar.

- **"next_year"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user selects the next year on a calendar.

- **"prev_month"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user selects the previous month on a calendar.

- **"prev_year"**

  ```
  procedure Handler (Calendar : access Gtk_Calendar_Record'Class);
  ```

  Emitted when the user selects the previous year on a calendar.

76.3 Types

```
type Gtk_Calendar_Detail_Func is access function
        (Calendar  : access Gtk_Calendar_Record'Class;
```

```
type Gtk_Calendar_Display_Options is mod 2 ** 8;
```

76.4 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Calendar);
```

Create a new Calendar that points to the current date.

```
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Calendar.

```
function Select_Month
  (Calendar           : access Gtk_Calendar_Record;
   Month              :      Guint;
   Year               :      Guint)
   return Boolean;
```

Shift the calendar to a different month/year.
Return True if sucessful.

```
procedure Select_Day
  (Calendar           : access Gtk_Calendar_Record;
   Day                :      Guint);
```

Select a day from the current month.
Only one day can be selected at a time.

```
function Mark_Day
  (Calendar           : access Gtk_Calendar_Record;
   Day                :      Guint)
   return Boolean;
```

Set a specified Day as marked in the Calendar.
This is shown visually as a painted box around the Day. Note that several days can be marked. Return True if successful.

```
function Unmark_Day
  (Calendar           : access Gtk_Calendar_Record;
   Day                :      Guint)
   return Boolean;
```

Undo the marking of Day.
Return True if sucessful.

```
procedure Clear_Marks
  (Calendar           : access Gtk_Calendar_Record);
```

Clear all the marks set by Mark_Day.

```
procedure Get_Date
  (Calendar           : access Gtk_Calendar_Record;
   Year               : out   Guint;
```

```
   Month              : out   Guint;
   Day                : out   Guint);
```

Return the date currently selected.

```
procedure Set_Display_Options
  (Calendar           : access Gtk_Calendar_Record;
   Flags              :      Gtk_Calendar_Display_Options);
function Get_Display_Options
  (Calendar           : access Gtk_Calendar_Record)
   return Gtk_Calendar_Display_Options;
```

Sets display options (whether to display the heading and the month headings).

```
function Get_Detail_Height_Rows
  (Calendar           : access Gtk_Calendar_Record)
   return Gint;
procedure Set_Detail_Height_Rows
  (Calendar           : access Gtk_Calendar_Record;
   Rows               :      Gint);
```

Queries the height of detail cells, in rows.
Since: 2.14

```
function Get_Detail_Width_Chars
  (Calendar           : access Gtk_Calendar_Record)
   return Gint;
procedure Set_Detail_Width_Chars
  (Calendar           : access Gtk_Calendar_Record;
   Chars              :      Gint);
```

Queries the width of detail cells, in characters.
Since: 2.14

76.4.1 Details

```
procedure Set_Detail_Func
  (Calendar           : access Gtk_Calendar_Record;
   Func               :      Gtk_Calendar_Detail_Func;
   Data               :      System.Address;
   Destroy            :      G_Destroy_Notify_Address);
```

Installs a function which provides Pango markup with detail information for each day. Examples for such details are holidays or appointments. That information is shown below each day when the property "show-details" is set. A tooltip containing with full detail information is provided, if the entire text should not fit into the details area, or if show-details is not set.

The size of the details area can be restricted by setting the "detail-width-chars" and "detail-height-rows" properties.

Since: 2.14

77 Package Gtk.Cell_Editable

The Gtk_Cell_Editable interface must be implemented for widgets to be usable when editing the contents of a Gtk_Tree_View cell.

77.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Cell_Editable     (Package Gtk.Cell_Editable)
```

77.2 Signals

- **"editing_done"**

  ```
  procedure Handler (Editable : Gtk_Cell_Editable);
  ```

  Should be emitted when the user has finished editing

- **"remove_widget"**

  ```
  procedure Handler (Editable : Gtk_Cell_Editable);
  ```

  Emitted when the widget is about to be removed

77.3 Types

type Gtk_Cell_Editable **is new** Glib.Types.GType_Interface;

77.4 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Cell_Editable.

```
procedure Start_Editing
  (Cell_Editable      :      Gtk_Cell_Editable;
   Event              :      Gdk.Event.Gdk_Event);
```

Begin editing on a Gtk_Cell_Editable.
Event is the Gdk_Event that began the editing process. It may be null, if the instance that editing was initiated through programatic means.

```
procedure Editing_Done
  (Cell_Editable      :      Gtk_Cell_Editable);
```

Emit the "editing_done" signal.
This signal is a sign for the cell renderer to update its value from the cell.

```
procedure Remove_Widget
  (Cell_Editable      :      Gtk_Cell_Editable);
```

Emit the "remove_widget" signal.
This signal is meant to indicate that the cell is finished editing, and the widget may now be destroyed.

78 Package Gtk.Cell_Layout

Gtk_Cell_Layout is an interface to be implemented by all objects which want to provide a Gtk_Tree_View_Column like API for packing cells, setting attributes and data funcs. The rendering of the widget is done through various Gtk_Cell_Renderer, and by reading data from a Gtk_Tree_Model.

78.1 Types

type Cell_Data_Func **is access procedure**

type Data_Type **is private**;

type Destroy_Notify **is access procedure**
(Data : **in out** Data_Type);

Free the memory used by Data

type Gtk_Cell_Layout **is new** Glib.Types.GType_Interface;

An interface (similar to Java's interfaces) that can be implemented by tagged types derived from Glib.Object.GObject.

78.2 Subprograms

function Get_Type **return** Glib.GType;

Returns the internal type used for a Gtk_Cell_Layout interface

```
procedure Pack_Start
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Expand             :      Boolean);
procedure Pack_End
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Expand             :      Boolean);
```

Adds Cell to the beginning or end of Cell_Layout. If Expand is False,
then the Cell is allocated no more space than it needs. Any unused space is divided evenly between cells for which Expand is True. Note that reusing the same cell renderer is not supported.

```
procedure Add_Attribute
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Attribute          :      String;
   Column             :      Gint);
```

Adds an attribute mapping to the list in Cell_Layout. Column is the
column of the model to get a value from, and Attribute is the parameter on Cell to be set

from the value. So for example if column of the model contains strings, you could have the "text" attribute of Gtk_Cell_Renderer_Text get its values from column 2.

```
procedure Clear
  (Cell_Layout        :      Gtk_Cell_Layout);
```

Unsets all the mappings on all renderers on Cell_Layout and removes all renderers from Cell_Layout.

```
procedure Clear_Attributes
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class);
```

Clears all existing attributes previously set with Add_Attribute.

```
procedure Reorder
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Position           :      Gint);
```

Re-inserts Cell at Position. Note that Cell has already to be packed into Cell_layout for this to function properly.

```
procedure Set_Cell_Data_Func
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Func               :      Cell_Data_Func);
```

Sets the Cell_Data_Func to use for Cell_Layout. This function is used instead of the standard attributes mapping for setting the column value, and should set the value of Cell_layout's cell renderer(s) as appropriate. Func may be null to remove and older one. This allows you to compute the attributes dynamically from several columns of the model for instance

```
procedure Set_Cell_Data_Func
  (Cell_Layout        :      Gtk_Cell_Layout;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Func               :      Cell_Data_Func;
   Data               :      Data_Type;
   Destroy            :      Destroy_Notify := null);
```

Same as the other Set_Cell_Data_Func, but passes an addition user data to the callback.

79 Package Gtk.Cell_Renderer

The Gtk_Cell_Renderer is a base class of a set of objects used for rendering a cell to a Gdk_Drawable. These objects are used primarily by the Gtk_Tree_View widget, though they aren't tied to them in any specific way. It is worth noting that Gtk_Cell_Renderer is not a Gtk_Widget and cannot be treated as such.

The primary use of a Gtk_Cell_Renderer is for drawing a certain graphical elements on a Gdk_Drawable. Typically, one cell renderer is used to draw many cells on the screen. To this extent, it isn't expected that Cell_Renderer keep any permanent state around. Instead, any state is set just prior to use using GObjects property system. Then, the cell is measured using Get_Size(). Finally, the cell is rendered in the correct location using Render().

There are a number of rules that must be followed when writing a new Gtk_Cell_Renderer. First and formost, it's important that a certain set of properties will always yield a cell renderer of the same size, barring GtkStyle change. The Gtk_Cell_Renderer also has a number of generic properties that are expected to be honored by all children.

Beyond merely rendering a cell, cell renderers can optionally provide active user interface elements. A cell renderer can be activatable like Gtk_Cell_Renderer_Toggle, which toggles when it gets activated by a mouse click, or it can be editable like Gtk_Cell_Renderer_Text, which allows the user to edit the text using a Gtk_Entry. To make a cell renderer activatable or editable, you have to implement the activate or start_editing virtual functions, respectively.

79.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Cell_Renderer     (Package Gtk.Cell_Renderer)
```

79.2 Signals

- **"editing-canceled"**

  ```
  procedure Handler (Cell : access Gtk_Cell_Renderer_Record'Class);
  ```

  This signal gets emitted when the user cancels the process of editing cell. For example, an editable cell renderer could be written to cancel editing when the user presses Escape.

- **"editing-started"**

79.3 Types

```
type Gtk_Cell_Renderer_Mode is
          (Cell_Renderer_Mode_Inert,
           Cell_Renderer_Mode_Activatable,
           Cell_Renderer_Mode_Editable);
```

```
type Gtk_Cell_Renderer_State is mod 2 ** 32;
```

```
type Property_Cell_Renderer_Mode is new Cell_Renderer_Mode_Properties.Property;
```

79.4 Subprograms

```
function Convert
  (R                    :      Gtk_Cell_Renderer)
   return System.Address;
function Convert
  (R                    :      System.Address)
   return Gtk_Cell_Renderer;
function Get_Type                return GType;
```

Return the internal value associated with Gtk_Cell_Renderer

```
procedure Get_Size
  (Cell                 : access Gtk_Cell_Renderer_Record;
   Widget               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Cell_Area            : out   Gdk.Rectangle.Gdk_Rectangle;
   X_Offset             : out   Gint;
   Y_Offset             : out   Gint;
   Width                : out   Gint;
   Height               : out   Gint);
```

Obtain the width and height needed to render the cell.
Used by view widgets to determine the appropriate size for the Cell_Area passed to Render. Fill in the x and y offsets (if set) of the cell relative to this location. Please note that the values set in Width and Height, as well as those in X_Offset and Y_Offset are inclusive of the Xpad and Ypad properties. Widget: the widget the renderer is rendering to. Cell_Area: The area a cell will be allocated. X_Offset: X offset of cell relative to Cell_Area. Y_Offset: Y offset of cell relative to Cell_Area. Width: Width needed to render a cell. Height: Height needed to render a cell.

```
procedure Render
  (Cell                 : access Gtk_Cell_Renderer_Record;
   Window               :      Gdk.Window.Gdk_Window;
   Widget               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Background_Area      :      Gdk.Rectangle.Gdk_Rectangle;
   Cell_Area            :      Gdk.Rectangle.Gdk_Rectangle;
   Expose_Area          :      Gdk.Rectangle.Gdk_Rectangle;
   Flags                :      Gtk_Cell_Renderer_State);
```

Invokes the virtual render function of the Gtk_Cell_Renderer. The three
passed-in rectangles are areas of Window. Most renderers will draw within Cell_Area; the Xalign, Yalign, Xpad, and Ypad fields of the GtkCellRenderer should be honored with respect to Cell_Area. Background_Area includes the blank space around the cell, and also the area containing the tree expander; so the Background_Area rectangles for all cells tile to cover the entire Window. Expose_Area is a clip rectangle.

```
function Activate
  (Cell                 : access Gtk_Cell_Renderer_Record;
   Event                :      Gdk.Event.Gdk_Event;
```

```
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Path               :      UTF8_String;
   Background_Area    :      Gdk.Rectangle.Gdk_Rectangle;
   Cell_Area          :      Gdk.Rectangle.Gdk_Rectangle;
   Flags              :      Gtk_Cell_Renderer_State)
   return Boolean;
```

Passes an activate event to the cell renderer for possible processing.
Some cell renderers may use events; for example, Gtk_Cell_Renderer_Toggle toggles when it gets a mouse click.

```
function Start_Editing
  (Cell               : access Gtk_Cell_Renderer_Record;
   Event              :      Gdk.Event.Gdk_Event;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Path               :      UTF8_String;
   Background_Area    :      Gdk.Rectangle.Gdk_Rectangle;
   Cell_Area          :      Gdk.Rectangle.Gdk_Rectangle;
   Flags              :      Gtk_Cell_Renderer_State)
   return Gtk.Cell_Editable.Gtk_Cell_Editable;
```

Passes an activate event to the cell renderer for possible processing.
Cell: a Gtk_Cell_Renderer Event: a Gdk_Event Widget: widget that received the event Path: widget-dependent string representation of the event location; e.g. for Gtk_Tree_View, a string representation of Gtk_Tree_Path Background_Area: background area as passed to Render Cell_Area: cell area as passed to Render

```
procedure Set_Fixed_Size
  (Cell               : access Gtk_Cell_Renderer_Record;
   Width              :      Gint;
   Height             :      Gint);
procedure Get_Fixed_Size
  (Cell               : access Gtk_Cell_Renderer_Record;
   Width              : out    Gint;
   Height             : out    Gint);
```

Sets the renderer size to be explicit, independent of the
properties set.

80 Package Gtk.Cell_Renderer_Combo

A Gtk_Cell_Renderer_Combo can be used to render a combobox in a cell.

80.1 Subprograms

```
procedure Gtk_New
  (Render             : out   Gtk_Cell_Renderer_Combo);
function Get_Type             return GType;
```

Returns the internal value associated with this renderer

81 Package Gtk.Cell_Renderer_Pixbuf

A Gtk_Cell_Renderer_Pixbuf can be used to render an image in a cell. It allows to render either a given Gdk_Pixbuf (set via the pixbuf property) or a stock icon (set via the stock-id property).

To support the tree view, Gtk_Cell_Renderer_Pixbuf also supports rendering two alternative pixbufs, when the is-expander property is TRUE. If the is-expanded property is TRUE and the pixbuf-expander-open property is set to a pixbuf, it renders that pixbuf, if the is-expanded property is FALSE and the pixbuf-expander-closed property is set to a pixbuf, it renders that one.

81.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Cell_Renderer       (Package Gtk.Cell_Renderer)
      \___ Gtk_Cell_Renderer_Pixbuf (Package Gtk.Cell_Renderer_Pixbuf)
```

81.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Cell_Renderer_Pixbuf);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

82 Package Gtk.Cell_Renderer_Progress

A Gtk_Cell_Renderer_Progress can be used to render a progress bar in a cell

82.1 Subprograms

```
procedure Gtk_New
  (Render             : out   Gtk_Cell_Renderer_Progress);
function Get_Type             return GType;
```

Returns the internal value associated with this renderer

83 Package Gtk.Cell_Renderer_Text

A Gtk_Cell_Renderer_Text renders a given text in its cell, using the font, color and style information provided by its properties. The text will be ellipsized if it is too long and the ellipsize property allows it.

If the mode is CELL_RENDERER_MODE_EDITABLE, the Gtk_Cell_Renderer_Text allows to edit its text using an entry.

83.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Cell_Renderer       (Package Gtk.Cell_Renderer)
      \___ Gtk_Cell_Renderer_Text (Package Gtk.Cell_Renderer_Text)
```

83.2 Signals

- **"edited"**

  ```
  procedure Handler
  (Widget : access Gtk_Cell_Renderer_Text_Record'Class;
  Path : UTF8_String;
  New_Text : UTF8_String);
  ```

  Called when the text has been edited interactively . Note that you also need to set the attribute "editable" for users to be able to interactively edit the cell. If you want to take into account the change, you need to change the value in the model appropriately, for instance through a call to Set_Value

83.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Cell_Renderer_Text);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

```
procedure Set_Fixed_Height_From_Font
  (Renderer           : access Gtk_Cell_Renderer_Text_Record;
   Number_Of_Rows     :      Gint);
```

Sets the height of a renderer to explicitly be determined by the "font" and "y_pad" property set on it. Further changes in these properties do not affect the height, so they must be accompanied by a subsequent call to this function. Using this function is unflexible, and should really only be used if calculating the size of a cell is too slow (ie, a massive number of cells displayed). If number_of_rows is -1, then the fixed height is unset, and the height is determined by the properties again.

84 Package Gtk.Cell_Renderer_Toggle

Gtk_Cell_Renderer_Toggle renders a toggle button in a cell. The button is drawn as a radio- or checkbutton, depending on the radio property. When activated, it emits the toggled signal.

84.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Cell_Renderer       (Package Gtk.Cell_Renderer)
      \___ Gtk_Cell_Renderer_Toggle (Package Gtk.Cell_Renderer_Toggle)
```

84.2 Signals

- **"toggled"**

```
procedure Handler
(Widget : access Gtk_Cell_Renderer_Toggle_Record'Class;
Path : UTF8_String);
```

84.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Cell_Renderer_Toggle);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

```
procedure Set_Radio
  (Toggle             : access Gtk_Cell_Renderer_Toggle_Record;
   Radio              :      Boolean);
function Get_Radio
  (Toggle             : access Gtk_Cell_Renderer_Toggle_Record)
   return Boolean;
```

If Setting is True, the cell renderer renders a radio toggle
(i.e. a toggle in a group of mutually-exclusive toggles). If False, it renders a check toggle (a standalone boolean option).

Note that this only affects the visual display, but your application is still responsible for enforcing the behavior, through the toggled signal.

```
procedure Set_Active
  (Toggle             : access Gtk_Cell_Renderer_Toggle_Record;
   Setting            :      Boolean);
function Get_Active
  (Toggle             : access Gtk_Cell_Renderer_Toggle_Record)
   return Boolean;
```

Whether the renderer is currently selected

85 Package Gtk.Cell_View

A Gtk_Cell_View displays a single row of a Gtk_Tree_Model, using cell renderers just like Gtk_Tree_View. Gtk_Cell_View doesn't support some of the more complex features of Gtk_Tree_View, like cell editing and drag and drop.

85.1 Subprograms

```
procedure Gtk_New
  (View                : out   Gtk_Cell_View);
procedure Gtk_New_With_Text
  (View                : out   Gtk_Cell_View;
   Text                :       String);
procedure Gtk_New_With_Markup
  (View                : out   Gtk_Cell_View;
   Markup              :       String);
procedure Gtk_New_With_Pixbuf
  (View                : out   Gtk_Cell_View;
   Pixbuf              :       Gdk.Pixbuf.Gdk_Pixbuf);
function Get_Type                return Glib.GType;
```

Returns the internal value used for Gtk_Cell_View

```
procedure Set_Displayed_Row
  (Cell_View           : access Gtk_Cell_View_Record;
   Path                :       Gtk.Tree_Model.Gtk_Tree_Path);
```

Sets the row of the model that is currently displayed by the Gtk_Cell_View. If the path is unset, then the contents of the cellview "stick" at their last value; this is not normally a desired result, but may be a needed intermediate state if say, the model for the Gtk_Cell_View becomes temporarily empty.

```
function Get_Displayed_Row
  (Cell_View           : access Gtk_Cell_View_Record)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Returns a Gtk_Tree_Path referring to the currently displayed row. If no row is currently displayed, null is returned.

```
function Get_Size_Of_Row
  (Cell_View           : access Gtk_Cell_View_Record;
   Path                :       Gtk.Tree_Model.Gtk_Tree_Path)
   return Gtk.Widget.Gtk_Requisition;
```

Return the size needed by Cell_View to display the model row pointed to by Path.

```
procedure Set_Background_Color
  (Cell_View           : access Gtk_Cell_View_Record;
   Color               :       Gdk.Color.Gdk_Color);
```

Sets the background color of View.

```
procedure Set_Model
  (Cell_View           : access Gtk_Cell_View_Record;
   Model               :       Gtk.Tree_Model.Gtk_Tree_Model);
```

Sets the model for Cell_View. If Cell_View already has a model set, it will remove it before setting the new model. If Model is null, then it will unset the old model.

```
function Get_Cell_Renderers
  (Cell_View          : access Gtk_Cell_View_Record)
   return Gtk.Cell_Renderer.Cell_Renderer_List.Glist;
```

Returns the cell renderers which have been added to Cell_View.
Return value: a list of cell renderers. The list must be freed by the caller.

85.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Cell_Layout" This interface should be used to add new renderers to the view, to render various columns of the model @end itemize

```
function "+"
  (View               : access Gtk_Cell_View_Record'Class)
   return Gtk.Cell_Layout.Gtk_Cell_Layout;
function "-"
  (Layout             :      Gtk.Cell_Layout.Gtk_Cell_Layout)
   return Gtk_Cell_View;
```

Converts to and from the Gtk_Cell_Layout interface

86 Package Gtk.Check_Button

A Gtk_Check_Button places a discrete Gtk_Toggle_Button next to a widget, (usually a Gtk_Label).

86.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Button     (Package Gtk.Button)
               \___ Gtk_Toggle_Button (Package Gtk.Toggle_Button)
                  \___ Gtk_Check_Button (Package Gtk.Check_Button)
```

86.2 Subprograms

```
procedure Gtk_New
  (Check_Button       : out   Gtk_Check_Button;
   Label              :       UTF8_String := "");
```

Create a check button.
if Label is null, then no widget is associated with the button, and any widget can be added to the button (with Gtk.Container.Add).

```
procedure Gtk_New_With_Mnemonic
  (Check_Button       : out   Gtk_Check_Button;
   Label              :       UTF8_String);
```

Create a new check button containing a label.
If characters in Label are preceded by an underscore, they are underlined indicating that they represent a keyboard accelerator called a mnemonic. Pressing Alt and that key activates the checkbutton. Label: The text of the button, with an underscore in front of the mnemonic character

```
function Get_Type                return Glib.GType;
```

Return the internal value associated with a Gtk_Check_Button.

87 Package Gtk.Check_Menu_Item

A Gtk_Check_Menu_Item is a menu item that maintains the state of a boolean value in addition to a Gtk_Menu_Item's usual role in activating application code.

A check box indicating the state of the boolean value is displayed at the left side of the Gtk_Menu_Item. Activating the Gtk_Menu_Item toggles the value.

87.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Item     (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
                  \___ Gtk_Check_Menu_Item (Package Gtk.Check_Menu_Item)
```

87.2 Signals

- **"toggled"**

  ```
  procedure Handler
  (Check_Menu_Item : access Gtk_Check_Menu_Item_Record'Class);
  ```

  Emitted when the state of the check box is changed. A signal handler can call Get_Active to discover the new state.

87.3 Subprograms

```
procedure Gtk_New
  (Check_Menu_Item    : out   Gtk_Check_Menu_Item;
   Label              :      UTF8_String := "");
procedure Gtk_New_With_Mnemonic
  (Check_Menu_Item    : out   Gtk_Check_Menu_Item;
   Label              :      UTF8_String);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Calendar.

```
procedure Set_Active
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record;
   Is_Active          :      Boolean);
function Get_Active
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record)
   return Boolean;
```

Set the active state of the menu item's check box.

```
procedure Set_Inconsistent
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record;
   Setting            :      Boolean);
function Get_Inconsistent
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record)
   return Boolean;
```

If the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a boolean setting, and the current values in that

range are inconsistent, you may want to display the check in an "in between" state. This function turns on "in between" display. Normally you would turn off the inconsistent state again if the user explicitly selects a setting. This has to be done manually, Set_Inconsistent only affects visual appearance, it doesn't affect the semantics of the widget.

```
procedure Set_Draw_As_Radio
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record;
   Draw_As_Radio      :      Boolean);
function Get_Draw_As_Radio
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record)
   return Boolean;
```

Sets whether Check_Menu_Item is drawn like a Radio_Menu_Item.

```
procedure Toggled
  (Check_Menu_Item    : access Gtk_Check_Menu_Item_Record);
```

Emit the "toggled" signal.

88 Package Gtk.Clipboard

The Gtk_Clipboard object represents a clipboard of data shared between different processes or between different widgets in the same process. Each clipboard is identified by a name encoded as a Gdk_Atom. (Conversion to and from strings can be done with gdk.properties.atom_intern and gdk.properties.atom_name().) The default clipboard corresponds to the "CLIPBOARD" atom; another commonly used clipboard is the "PRIMARY" clipboard, which, in X, traditionally contains the currently selected text.

To support having a number of different formats on the clipboard at the same time, the clipboard mechanism allows providing callbacks instead of the actual data. When you set the contents of the clipboard, you can either supply the data directly (via functions like Set_Text), or you can supply a callback to be called at a later time when the data is needed (via Set_With_Data or Set_With_Owner.) Providing a callback also avoids having to make copies of the data when it is not needed.

Set_With_Data and Set_With_Owner are quite similar; the choice between the two depends mostly on which is more convenient in a particular situation. The former is most useful when you want to have a blob of data with callbacks to convert it into the various data types that you advertise. When the clear_func you provided is called, you simply free the data blob. The latter is more useful when the contents of clipboard reflect the internal state of a GObject (As an example, for the PRIMARY clipboard, when an entry widget provides the clipboard's contents the contents are simply the text within the selected region.) If the contents change, the entry widget can call Set_With_Owner() to update the timestamp for clipboard ownership, without having to worry about clear_func being called.

Requesting the data from the clipboard is essentially asynchronous. If the contents of the clipboard are provided within the same process, then direct function call will be made to retrieve the data, but if they are provided by another process, then the data needs to be retrieved from the other process, which may take some time. To avoid blocking the user interface, the call to request the selection, Request_Contents takes a callback that will be called when the contents are received (or when the request fails.) If you don't want to deal with providing a separate callback, you can also use Wait_For_Contents. What this does is run the GLib main loop recursively waiting for the contents. This can simplify the code flow, but you still have to be aware that other callbacks in your program can be called while this recursive mainloop is running.

Along with the functions to get the clipboard contents as an arbitrary data chunk, there are also functions to retrieve it as text, Request_Text and Wait_For_Text. These functions take care of determining which formats are advertised by the clipboard provider, asking for the clipboard in the best available format and converting the results into the UTF-8 encoding. (The standard form for representing strings in GTK+.)

88.1 Signals

- **"owner_change"**

88.2 Types

type Gtk_Clipboard **is new** Glib.C_Proxy;

type Gtk_Clipboard_Clear_Func **is access procedure**

type Gtk_Clipboard_Get_Func **is access procedure**

type Gtk_Clipboard_Image_Received_Func **is access procedure**
```
          (Clipboard : Gtk_Clipboard;
           Pixbuf    : System.Address;
           Data      : System.Address);
```

type Gtk_Clipboard_Received_Func **is access procedure**

type Gtk_Clipboard_Targets_Received_Func **is access procedure**

type Gtk_Clipboard_Text_Received_Func **is access procedure**
```
          (Clipboard : Gtk_Clipboard;
           Text      : Interfaces.C.Strings.chars_ptr;
           Data      : System.Address);
```

88.3 Subprograms

```
function Get_Type              return Glib.GType;
```

Return the internal type used for clipboards

```
function Get_Clipboard
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Selection          :      Gdk.Types.Gdk_Atom)
   return Gtk.Clipboard.Gtk_Clipboard;
```

Returns the clipboard object for the given selection to be used with Widget. Widget must have a Gdk_Display associated with it, so must be attached to a toplevel window.

Return value: the appropriate clipboard object. If no clipboard already exists, a new one will be created. Once a clipboard object has been created, it is persistent for all time.

```
function Get
  (Selection          :      Gdk.Types.Gdk_Atom
                      := Gdk.Types.Gdk_None)
   return Gtk_Clipboard;
```

Return the clipboard object for the given selection. Cut/copy/paste menu items and keyboard shortcuts should use the default clipboard, returned by passing Gdk_None for Selection. The currently-selected object or text should be provided on the clipboard identified by Selection_Primary. Cut/copy/paste menu items conceptually copy the contents of the Selection_Primary clipboard to the default clipboard, i.e. they copy the selection to what the user sees as the clipboard.

(Passing Gdk_None is the same as using Atom_Intern ("CLIPBOARD", False). See http://standards.freedesktop.org/clipboards-spec/ for a detailed discussion of the "CLIPBOARD" vs. "PRIMARY" selections under the X window system. On Win32 the Selection_Primary clipboard is essentially ignored.)

It's possible to have arbitrary named clipboards; if you do invent new clipboards, you should prefix the selection name with an underscore (because the ICCCM requires that nonstandard atoms are underscore-prefixed), and namespace it as well. For example, if your application called "Foo" has a special-purpose clipboard, you might call it "_FOO_SPECIAL_CLIPBOARD".

Selection is a Gdk_Atom which identifies the clipboard to use.

If no clipboard already exists, a new one will be created. Once clipboard object has been created, it is persistent for all time and cannot be freed.

```
procedure Set_Can_Store
  (Clipboard          :      Gtk_Clipboard;
   Targets            :      Gtk.Selection.Target_Entry_Array);
```

Hints that the clipboard data should be stored somewhere when the application exits or when Store is called. This value is reset when the clipboard owner changes. Where the clipboard data is stored is platform dependent. Targets is an array containing information about which forms should be stored, or an empty array to indicate that all forms should be stored.

```
procedure Store
  (Clipboard          :      Gtk_Clipboard);
```

Stores the current clipboard data somewhere so that it will stay around after the application has quit. See also Gdk.Display.Supports_Clipboard_Persistence and Gdk.Display.Store_Clipboard.

```
function Get_Owner
  (Clipboard          :      Gtk_Clipboard)
   return Glib.Object.GObject;
```

If the clipboard contents callbacks were set with Set_With_Owner, and the Set_With_Data or Clear has not subsequently called, returns the owner set by Set_With_Owner.

```
procedure Clear
  (Clipboard          :      Gtk_Clipboard);
```

Clear the contents of the clipboard.
Generally this should only be called between the time you call Set_With_Owner or Set_With_Data, and when the Clear_Func you supplied is called. Otherwise, the clipboard may be owned by someone else.

88.3.1 Text

```
procedure Set_Text
  (Clipboard          :      Gtk_Clipboard;
   Text               :      UTF8_String);
```

Set the contents of the clipboard.

```
function Wait_For_Text
  (Clipboard          :      Gtk_Clipboard)
   return UTF8_String;
```

Requests the contents of the clipboard as text and converts the result to UTF-8 if necessary. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

Return "" if retrieving the selection data failed. (This could happen for various reasons, in particular if the clipboard was empty or if the contents of the clipboard could not be converted into text form)

```
function Wait_Is_Text_Available
  (Clipboard          :      Gtk_Clipboard)
   return Boolean;
```

Test to see if there is text available to be pasted. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

```
procedure Request_Text
  (Clipboard          :      Gtk_Clipboard;
   Callback           :      Gtk_Clipboard_Text_Received_Func;
   User_Data          :      System.Address);
```

Requests the contents of the clipboard as text. When the text is later received, it will be converted to UTF-8 if necessary, and Callback will be called.

88.3.2 Images

```
procedure Set_Image
  (Clipboard          :      Gtk_Clipboard;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf);
```

Sets the contents of the clipboard to the given pixbuf. GTK+ will take responsibility for responding for requests for the image, and for converting the image into the requested format.

```
function Wait_For_Image
  (Clipboard          :      Gtk_Clipboard)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Requests the contents of the clipboard as image and converts the result to a pixbuf. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait. The returned value must be freed with a call to Unref.

```
function Wait_Is_Image_Available
  (Clipboard          :      Gtk_Clipboard)
   return Boolean;
```

Test to see if there is an image available to be pasted. This is done by requesting the TARGETS atom and checking if it contains any of the supported image targets. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait. This function is a little faster than calling Wait_For_Image since it doesn't need to retrieve the actual image data.

```
procedure Request_Image
  (Clipboard          :      Gtk_Clipboard;
   Callback           :      Gtk_Clipboard_Image_Received_Func;
   User_Data          :      System.Address);
```

Requests the contents of the clipboard as image. When the image is later received, it will be converted to a pixbuf, and Callback will be called.

88.3.3 Other contents

```
function Set_With_Data
  (Clipboard          :      Gtk_Clipboard;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Get_Func           :      Gtk_Clipboard_Get_Func;
   Clear_Func         :      Gtk_Clipboard_Clear_Func;
   User_Data          :      System.Address)
   return Boolean;
```

Virtually sets the contents of the specified clipboard by providing a list of supported formats for the clipboard data and a function to call to get the actual data when it is requested. No actual copy of the data is made until someones actually requests it. Targets contains information about the available forms for the clipboard data. This function returns True if setting the clipboard data succeeded.

```
function Set_With_Owner
  (Clipboard          :      Gtk_Clipboard;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Get_Func           :      Gtk_Clipboard_Get_Func;
   Clear_Func         :      Gtk_Clipboard_Clear_Func;
   Owner              : access Glib.Object.GObject_Record'Class)
   return Boolean;
```

Same as Set_With_Data, but an actual object is passed instead of a generic user_data. This takes care of referencing the object as appropriate.

```
function Wait_For_Targets
  (Clipboard          :      Gtk_Clipboard)
   return Gdk.Types.Gdk_Atom_Array;
```

Returns a list of targets that are present on the clipboard, or an empty array if there aren't any targets available. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait.

```
function Wait_For_Contents
  (Clipboard          :      Gtk_Clipboard;
   Target             :      Gdk.Types.Gdk_Atom)
   return Gtk.Selection.Selection_Data;
```

Requests the contents of the clipboard using the given target. This function waits for the data to be received using the main loop, so events, timeouts, etc, may be dispatched during the wait. The result must be freed.

```
function Wait_Is_Target_Available
  (Clipboard          :      Gtk_Clipboard;
   Target             :      Gdk.Types.Gdk_Atom)
   return Boolean;
```

Checks if a clipboard supports pasting data of a given type. This function can be used to determine if a "Paste" menu item should be insensitive or not. If you want to see if there's text available on the clipboard, use Wait_Is_Text_Available instead. The value for Target is similar to the one in Gtk.Selection.Target_Entry

```
procedure Request_Contents
  (Clipboard          :      Gtk_Clipboard;
   Target             :      Gdk.Types.Gdk_Atom;
   Callback           :      Gtk_Clipboard_Received_Func;
   User_Data          :      System.Address);
```

Requests the contents of clipboard as the given target.
When the results of the result are later received the supplied callback will be called.

```
procedure Request_Targets
  (Clipboard          :      Gtk_Clipboard;
   Callback           :      Gtk_Clipboard_Targets_Received_Func;
   User_Data          :      System.Address);
```

Requests the contents of the clipboard as list of supported targets.
When the list is later received, Callback will be called.

89 Package Gtk.Clist

This widget is deprecated. Use Gtk.Tree_View instead.

This widget displays a multi-column list. Each line is made of a number of column, each being able to display any kind of widget.

The intersection of a line and a column is called a Cell. Each cell can have a different type (Cell_Text, Cell_Pixmap, Cell_Pixtext), and display its contents depending on this type. For instance, the text is not displayed in the type is Cell_Pixmap. Note that this type is changed dynamically by some of the subprograms below, like Set_Pixmap, Set_Text, ... and Set_Cell_Contents

This is one of the most powerful widgets in GtkAda, that can be used to display an kind of information. Look also into using Gtk_Ctree, which is a similar widget.

You can add scrolling in a Gtk_Clist by adding it in a Gtk_Scrolled_Window.

89.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Clist       (Package Gtk.Clist)
```

89.2 Signals

- **"abort_column_resize"**

  ```
  procedure Handler (Clist       : access Gtk_Clist_Record'Class);
  ```

  Aborts the current interactive resizing of the column by the user. This releases the grab done on the pointer. It is never emitted internally by GtkAda.

- **"click_column"**

  ```
  procedure Handler (Clist  : access Gtk_Clist_Record'Class;
  Column : Gint);
  ```

  Emitted when the user has clicked on one of the buttons at the top of a column. The first column has number 0.

- **"end_selection"**

  ```
  procedure Handler (Clist  : access Gtk_Clist_Record'Class);
  ```

  Ends the current selection process. This is never emitted internally by GtkAda, but acts as if the user had just released the mouse button.

- **"extend_selection"**

  ```
  procedure Handler (Clist       : access Gtk_Clist_Record'Class;
  Scroll_Type : Gtk.Enums.Gtk_Scroll_Type;
  Position    : Gfloat;
  Auto_Start_Selection : Boolean);
  ```

  Extends the current selection. Position is used only for certain values of Scroll_Type. It is never emitted internally by GtkAda. It has no effect if the selection mode is not Extended.

- **"resize_column"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class;
Column : Gint;
Width  : Gint);
```

Emitted to request a new size for a given column. You should use Set_Column_Width instead.

- **"row_move"**

```
procedure Handler (Clist      : access Gtk_Clist_Record'Class;
Source_Row : Gint;
Dest_Row   : Gint);
```

Emitted to request the change of a Source_Row to Dest_Row. You should use Row_Move instead.

- **"scroll_horizontal"**

```
procedure Handler (Clist       : access Gtk_Clist_Record'Class;
Scroll_Type : Gtk.Enums.Gtk_Scroll_Type;
Position    : Gfloat);
```

Scrolls the clist horizontally. This also modifies the selection. It is never emitted internally by GtkAda. You should consider using Moveto instead.

- **"scroll_vertical"**

```
procedure Handler (Clist       : access Gtk_Clist_Record'Class;
Scroll_Type : Gtk.Enums.Gtk_Scroll_Type;
Position    : Gfloat);
```

Scrolls the clist vertically. This also modifies the selection. It is never emitted internally by GtkAda. You should consider using Moveto instead.

- **"select_all"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Emitted to request the selection of all the rows in the Clist, if the selection mode allows. You should use Select_All instead.

- **"select_row"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class;
Row    : Gint;
Column : Gint;
Event  : Gdk.Event.Gdk_Event);
```

Emitted when a row is selected. Column contains the column number in which the user has clicked, or -1 if the selection was done internally by GtkAda. Event will be null if the selection was not triggered by an event, eg if the row was selected through a call to Select_Row.

- **"start_selection"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Request the start of the selection. This signal is not emitted internally by GtkAda, but acts as if the user had clicked on the focus row (the exact visual modification depends on the selection mode).

- **"toggle_add_mode"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Changes the add_mode for the clist (indicates whether the next line clicked on will be added to the selection or will replace it). This is never emitted internally by GtkAda.

- **"toggle_focus_row"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Emitted to request the change of the selection status (selected/ unselected) of the focus row. This signal is not emitted internally by GtkAda.

- **"undo_selection"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Emitted to request the cancellation of the last select/unselect operation. You should use Undo_Selection instead.

- **"unselect_all"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class);
```

Emitted to request the unselection of all the rows in the Clist, if the selection mode is different from Browse. You should use Unselect_All instead.

- **"unselect_row"**

```
procedure Handler (Clist  : access Gtk_Clist_Record'Class;
Row    : Gint;
Column : Gint;
Event  : Gdk.Event.Gdk_Event);
```

Emitted to request the unselection of a row. Event will be null most of the time when the event is emitted directly by GtkAda. You should use Unselect_Row instead.

89.3 Types

```
type Gtk_Button_Action is new Guint;
```

```
type Gtk_Cell_Type is
          (Cell_Empty,
           Cell_Text,
           Cell_Pixmap,
           Cell_Pixtext,
           Cell_Widget);
```

```
type Gtk_Clist_Compare_Func is access function
          (Clist : access Gtk_Clist_Record'Class;
```

```
type Gtk_Clist_Row is new Gdk.C_Proxy;
```

A row of the clist. Application-specific data can be associated with each row. In the following subprograms, rows can also be accessed via their number, starting from 0.

```
type Gtk_Sort_Type is
          (Ascending, Descending);
```

89.4 Subprograms

89.4.1 Creating a list and setting the attributes

```
procedure Gtk_New
  (Widget             : out   Gtk_Clist;
   Columns            : in    Gint);
```

Create a list with Columns columns.
Each line will have this exact number of column The number of columns can not be changed once the widget has been created.

```
procedure Gtk_New
  (Widget             : out   Gtk_Clist;
   Columns            : in    Gint;
   Titles             : in    Gtkada.Types.Chars_Ptr_Array);
```

Create a new list with Columns columns.
The title of the columns is specified in Titles. The results are undefined (and can raise an exception) if Titles does not have at least Columns items.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Clist.

```
procedure Set_Hadjustment
  (Clist              : access Gtk_Clist_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
```

Set the horizontal adjustment used for the clist.
Note that such an adjustment is automatically created when the clist is added to a Gtk_Scrolled_Window. You should rather use Gtk.Scrolled_Window.Set_Hadjustment if you want to modify the adjustment. If there was already such an adjustment, it is unref-ed, and might be deleted.

```
procedure Set_Vadjustment
  (Clist              : access Gtk_Clist_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
```

Set the vertical adjustment used for the clist.
Note that such an adjustment is automatically created when the clist is added to a Gtk_Scrolled_Window. You should rather use Gtk.Scrolled_Window.Set_Hadjustment if you want to modify the adjustment. If there was already such an adjustment, it is unref-ed, and might be deleted.

```
function Get_Hadjustment
  (Clist              : access Gtk_Clist_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the horizontal adjustment used for the clist.
This indicates what position the clist is presently displaying, and by changing its value, the clist is automatically scrolled horizontally. This is done automatically when the clist's parent is a Gtk_Scrolled_Window.

```
function Get_Vadjustment
  (Clist              : access Gtk_Clist_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the vertical adjustment used for the clist.
This indicates what position the clist is presently displaying, and by changing its value, the clist is automatically scrolled vertically. This is done automatically when the clist's parent is a Gtk_Scrolled_Window.

```
procedure Set_Selection_Mode
  (Clist              : access Gtk_Clist_Record;
   Mode               : in    Gtk.Enums.Gtk_Selection_Mode);
```

Modify the selection mode for the clist.
This indicates whether one or more lines can be selected at the same time in the clist, and how this selection can done by the user (does he have to click explicitly on an item, or can he browse through the clist and select the last item he was on, etc.)

Note that changing the selection mode to Selection_Single or Selection_Browse will deselect all the items in the clist.

```
function Get_Selection_Mode
  (Clist              : access Gtk_Clist_Record)
   return Gtk.Enums.Gtk_Selection_Mode;
```

Return the selection mode for the clist.

89.4.2 Visual aspects

```
procedure Freeze
  (Clist              : access Gtk_Clist_Record);
```

Freeze all visual updates on the list, while you make big changes.
This is more efficient than working on an unfrozen list.

```
procedure Thaw
  (Clist              : access Gtk_Clist_Record);
```

Thaw the list, ie reactivate all the visual updates.
This also forces an immediate refresh of the list. Note that each Freeze must be followed by a Thaw. The visual updates are not reactivated until the last Thaw has been emitted, but there is an immediate refresh every time anyway.

```
procedure Set_Shadow_Type
  (Clist              : access Gtk_Clist_Record;
   The_Type           : in    Gtk.Enums.Gtk_Shadow_Type);
```

Set the border style of the clist.

89.4.3 Modifying the contents

```
function Append
  (Clist              : access Gtk_Clist_Record;
   Text               : in    Gtkada.Types.Chars_Ptr_Array)
   return Gint;
```

Append a new row to the clist, and return the index of the row created.
The row is added at the end of the Clist. The behavior is undefined if Text does not have at least as many items as there are columns in the Clist.

```
function Prepend
  (Clist              : access Gtk_Clist_Record;
   Text               : in    Gtkada.Types.Chars_Ptr_Array)
   return Gint;
```

Add a new row at the beginning of the clist, and return its index.
The behavior is undefined if Text does not have at least as many items as there are columns in the Clist.

```
procedure Insert
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
```

```
   Text               : in    Gtkada.Types.Chars_Ptr_Array);
```

Add a new row in the clist.
The row 0 is the first in the clist. If Row is not in the range for clist, the new row is added at the end. The behavior is undefined if Text does not have enough items.

```
procedure Remove
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint);
```

Remove a row from the clist (0 is the first one).

```
procedure Clear
  (Clist              : access Gtk_Clist_Record);
```

Clears the entire list. This is much faster than doing a Remove on each line.

```
procedure Swap_Rows
  (Clist              : access Gtk_Clist_Record;
   Row1               : in    Gint;
   Row2               : in    Gint);
```

Exchange the position of two rows in the clist.

```
procedure Row_Move
  (Clist              : access Gtk_Clist_Record;
   Source_Row         : in    Gint;
   Dest_Row           : in    Gint);
```

Move the row at Source_Row to Dest_Row (0 indicates the first row in the clist)

```
procedure Set_Sort_Column
  (Clist              : access Gtk_Clist_Record;
   Column             :      Gint);
```

Indicate the column on which to sort the clist.
This column is relevant when you use Sort or Set_Auto_Sort below. The first column is number 0.

```
function Get_Sort_Column
  (Clist              : access Gtk_Clist_Record)
   return Gint;
```

Return the column on which the clist is sorted.

```
procedure Set_Sort_Type
  (Clist              : access Gtk_Clist_Record;
   Sort_Type          :      Gtk_Sort_Type);
```

Indicate in which order the sort should be done on the clist (ascending or descending).

```
function Get_Sort_Type
  (Clist              : access Gtk_Clist_Record)
   return Gtk_Sort_Type;
```

Return the sort type currently used for the list

```
procedure Sort
  (Clist              : access Gtk_Clist_Record);
```

Sort the lines of the clist, based on the column set by Set_Sort_Column, and in the order set by Set_Sort_Type.

```
procedure Set_Auto_Sort
  (Clist              : access Gtk_Clist_Record;
   Auto_Sort          :      Boolean);
```

If Auto_Sort is true, then the clist will be automatically sorted every time a new line is inserted into the clist.

```
procedure Set_Compare_Func
  (Clist              : access Gtk_Clist_Record;
   Func               :      Gtk_Clist_Compare_Func);
```

Set the function used when sorting the list. This function takes two rows as its arguments, and should return a Gint indicating in which order the rows are found (-1 if Row1 comes first, 0 if they are equal, 1 if Row2 comes last). Func should be null to restore the default sorting functions.

89.4.4 Columns

```
function Get_Columns
  (Clist              : access Gtk_Clist_Record)
   return Gint;
```

Return the number of columns in the clist.

```
procedure Column_Titles_Hide
  (Clist              : access Gtk_Clist_Record);
```

Hide the column titles for the list.
This is the default behavior if no column titles were given when the list was created.

```
procedure Column_Titles_Show
  (Clist              : access Gtk_Clist_Record);
```

Show the column titles for the list.
This is the default behavior if some column titles were given when the list was created.

```
procedure Column_Title_Active
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint);
```

Set the column title to be an activate title.
In other words, answer all button presses, highlights when the mouse is over it, ...

```
procedure Column_Title_Passive
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint);
```

Set the column title to be passive.
Act just as a title, and do not react to mouse events.

```
procedure Column_Titles_Active
  (Clist              : access Gtk_Clist_Record);
```

Set all column titles to be active.

```
procedure Column_Titles_Passive
  (Clist              : access Gtk_Clist_Record);
```

Set all column titles to be passive.

```
procedure Set_Column_Title
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Title              : in    UTF8_String);
```

Set the text for the button of the column's title.
See Set_Column_Widget if you want to put a pixmap inside the button.

```
function Get_Column_Title
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint)
   return UTF8_String;
```

Return the text used for the title's column.
This is a copy of the title, so you can't modify it to automatically change the column's title.

```
procedure Set_Column_Widget
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Modify the widget used in the Gtk_Button that is the column's title.
By default, this button contains a simple Gtk_Label, which is replaced by Widget. This is the function to use if you want to put a pixmap (or a Gtk_Box that contains both a pixmap and some text) in a column's title.

```
function Get_Column_Widget
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint)
   return Gtk.Widget.Gtk_Widget;
```

Return the child of the button that makes the column's title.
Unless you changed it with Set_Column_Widget, this will return a Gtk_Label. Note also that if this widget was not created in Ada, but transparently by gtk+, you have to 'with' Gtk.Type_Conversion so that the correct type of the widget is created (See the user's guide for more information on type conversion).

```
procedure Set_Column_Justification
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Justification      : in    Gtk.Enums.Gtk_Justification);
```

Change the way the text in the whole column is justified.
This function has no effect on the title if you used Set_Column_Widget before.

```
procedure Set_Column_Visibility
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Visible            : in    Boolean);
```

Modify the visibility of a column.
Note that GtkAda prevents the last remaining visible column to be hidden. Nothing will be done if you try to hide that last column. See the example below for an example how to hide all the columns but one.

```
procedure Set_Column_Resizeable
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Resizeable         : in    Boolean);
```

Set whether the column can be dynamically resized with the mouse.
If Resizeable is true, then the column can be resized by clicking and dragging the lines that separates the column from the next one.

```
procedure Set_Column_Auto_Resize
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Auto_Resize        : in    Boolean);
```

Set whether the column should automatically be resized to the optimal
size (based on its contents). Note that this operation could slow things down a lot if you have a lot of items in your list.

```
function Columns_Autosize
  (Clist              : access Gtk_Clist_Record)
   return Gint;
```

Set all the columns' width to their optimal size.
Return the total width of the clist after this operation.

```
function Optimal_Column_Width
  (Clist              : access Gtk_Clist_Record;
   Column             :      Gint)
   return Gint;
```

Return the optimal width for Column, based on its contents.
This is the maximal cell width in the column.

```
procedure Set_Column_Width
  (Clist              : access Gtk_Clist_Record;
   Column             : in    Gint;
   Width              : in    Gint);
```

Set the column width in pixels.
By default, the column's width is chosen from the column's title.

```
procedure Set_Column_Min_Width
  (Clist              : access Gtk_Clist_Record;
   Column             :      Gint;
   Min_Width          :      Gint);
```

Set the minimal width for the column, in pixels.
if Min_Width is negative, there is no limit on the minimal width for the column.

```
procedure Set_Column_Max_Width
  (Clist              : access Gtk_Clist_Record;
   Column             :      Gint;
   Max_Width          :      Gint);
```

Set the maximal width for the column, in pixels.
If Max_Width is negative, there is no limit on the maximal width for the column.

89.4.5 Rows

```
function Get_Rows
  (Clist              : access Gtk_Clist_Record)
   return Gint;
```

Return the number of rows in the clist.

```
procedure Set_Row_Height
  (Clist              : access Gtk_Clist_Record;
   Height             :      Gint);
```

Set the height of the rows, in pixels.
if Height is 0, the chosen height will be the current's font height.

```
function Row_Is_Visible
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint)
   return Gtk.Enums.Gtk_Visibility;
```

Return the visibility status of the row.

```
procedure Set_Foreground
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Color              : in    Gdk.Color.Gdk_Color);
```

Set the foreground color for the row.
The color must already be allocated. If no such row exists in the list, nothing is done.

```
procedure Set_Background
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Color              : in    Gdk.Color.Gdk_Color);
```

Set the background color for the row.
The color must already be allocated. If no such row exists in the list, nothing is done.

```
procedure Set_Row_Style
  (Clist              : access Gtk_Clist_Record;
   Row                :      Gint;
   Style              : in    Gtk.Style.Gtk_Style);
```

Set the default style for the cells in the row. This can be
overridden for each cell with Set_Cell_Style.

```
function Get_Row_Style
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint)
   return Gtk.Style.Gtk_Style;
```

Return the default style used for the row.

```
procedure Set_Selectable
  (Clist              : access Gtk_Clist_Record;
   Row                :      Gint;
   Selectable         :      Boolean);
```

Indicate whether the row can be selected or not.
The default value is True.

```
function Get_Selectable
  (Clist              : access Gtk_Clist_Record;
   Row                :      Gint)
   return Boolean;
```

Return the selectable status of the row.

```
procedure Select_Row
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint);
```

Emit the signal "select_row". This simulates the user pressing
the mouse on Row, Column on the clist.

```
procedure Unselect_Row
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint);
```

Emit the signal "unselect_row", as if the user had clicked on
Row, Column on the clist.

```
procedure Undo_Selection
  (Clist              : access Gtk_Clist_Record);
```

Undo the last select/unselect operation.

```
procedure Get_Selection_Info
  (Clist              : access Gtk_Clist_Record;
   X                  : in    Gint;
   Y                  : in    Gint;
   Row                : out   Gint;
   Column             : out   Gint;
   Is_Valid           : out   Boolean);
```

Return the Row/Column corresponding to the coordinates X,Y in the
Row column. The coordinates X,Y are relative to the clist window (ie 0,0 is the top left corner of the clist). The result is valid only if Is_Valid is true

```
procedure Select_All
  (Clist              : access Gtk_Clist_Record);
```

Select all the rows in the clist. This only works if the selection
mode allows for multiple rows selected at the same time (extended or multiple).

```
procedure Unselect_All
  (Clist              : access Gtk_Clist_Record);
```

Deselect all the rows in the clist. If the selection mode is
Browse, then only the current line is deselected.

```
function Get_Focus_Row
  (Clist              : access Gtk_Clist_Record)
   return Gint;
```

Return the number of the line that currently has the focus.

```
function Get_Row_List
  (Clist              : access Gtk_Clist_Record)
   return Row_List.Glist;
```

Return the list of all the rows in the clist. This might speed up
the access to the rows a little. You can then use the function Set_Cell_Contents to modify the cells in the row, and Get_Text or Get_Pixmap to get its contents.

```
function Get_Selection
  (Widget             : access Gtk_Clist_Record)
   return Gtk.Enums.Gint_List.Glist;
```

Return the list of selected rows, by number.

89.4.6 Cells

```
function Get_Cell_Type
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint)
   return Gtk_Cell_Type;
```

Return the type of the cell at Row/Column.
This indicates which of the functions Get_Text. Get_Pixmap, etc. below you can use.

```
procedure Set_Text
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Text               : in    UTF8_String);
```

Set the cell's text, replacing its current contents.
This changes the type of the cell to Cell_Text. The pixmap (if any) will no longer be displayed.

```
function Get_Text
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint)
   return UTF8_String;
```

Return the text contained in cell. The type of the cell should be
either Cell_Text or Cell_Pixtext. If there was a problem, a null-length string is returned. The problem might appear in case the row or the column are invalid, or if the cell does not contain any text.

```
function Get_Text
  (Clist              : access Gtk_Clist_Record;
   Row                :      Gtk_Clist_Row;
   Column             : in    Gint)
   return UTF8_String;
```

Return the text contained in cell. The Row can be obtained from
Get_Row_List, this function speeds up the access a little compared to the other Get_Text above.

```
procedure Set_Pixmap
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Pixmap             : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask               : in    Gdk.Bitmap.Gdk_Bitmap);
```

Set the cell's pixmap, replacing its current contents.
The type of the cell becomes Cell_Pixmap, and the text is no longer displayed.

```
procedure Get_Pixmap
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap;
   Is_Valid           : out   Boolean);
```

Return the pixmap contained in a cell. The type of the cell should
be Cell_Pixmap. The result is meaningful only if Is_Valid is True. If the Cell did not contain a pixmap, Is_Valid is set to False

```
procedure Get_Pixmap
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gtk_Clist_Row;
   Column             : in    Gint;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap;
   Is_Valid           : out   Boolean);
```

Return the pixmap contained in a cell. Row can be obtained directly with
Get_Row_List, and speeds up the access a little compared to the previous Get_Pixmap function.

```
procedure Set_Pixtext
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Text               : in    UTF8_String;
   Spacing            : in    Guint8;
   Pixmap             : in    Gdk.Pixmap.Gdk_Pixmap;
```

```
   Mask               : in    Gdk.Bitmap.Gdk_Bitmap);
```

Set both the text and the pixmap for the cell.
Replace its current contents. The type of the cell becomes Cell_Pixtext, and both the text and the pixmap are displayed.

```
procedure Get_Pixtext
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Spacing            : out   Guint8;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap;
   Is_Valid           : out   Boolean);
```

The result is not meaningful if Is_Valid is False.
The only way to get the string is to use Get_Text, since a String is an unconstrained type in Ada and is not really convenient to use as an out parameter.

```
procedure Set_Cell_Style
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Style              : in    Gtk.Style.Gtk_Style);
```

Set the style (font, color, ...) used for the cell.
This overrides the row's style.

```
function Get_Cell_Style
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint)
   return Gtk.Style.Gtk_Style;
```

Return the style of the cell.

```
procedure Set_Shift
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Vertical           : in    Gint;
   Horizontal         : in    Gint);
```

Set a horizontal and vertical shift for drawing the content of the cell.
Both shifts can be either positive or negative. This is particularly useful for indenting items in a columns.

```
procedure Set_Cell_Contents
  (Clist              : access Gtk_Clist_Record;
   Row                :        Gtk_Clist_Row;
   Column             :        Gint;
   Cell_Type          :        Gtk_Cell_Type;
   Text               :        UTF8_String;
   Spacing            :        Guint8;
   Pixmap             :        Gdk.Pixmap.Gdk_Pixmap;
   Mask               :        Gdk.Bitmap.Gdk_Bitmap);
```

Modify the contents and type of a cell.
Cell_Type indicates what should be displayed in the cell. Note that if you do not want any string, you should pass an empty string "". You get Row from Get_Row_List.

89.4.7 Reordering the list

```
procedure Set_Reorderable
  (Clist              : access Gtk_Clist_Record;
   Reorderable        :      Boolean);
```

Set whether the list can be dynamically reordered by the user.
(using a simple drag-n-drop protocol).

```
procedure Set_Use_Drag_Icons
  (Clist              : access Gtk_Clist_Record;
   Use_Icons          :      Boolean);
```

Set whether drag icons are shown while the user is reordering the list.
The default value is True.

```
procedure Set_Button_Actions
  (Clist              : access Gtk_Clist_Record;
   Button             :      Guint;
   Button_Action      :      Gtk_Button_Action);
```

Set the action for a specific button on the list.
The default if for the left mouse button to select or drag and item, the other buttons are ignored. The Button_Expands action has no effect on a clist.

```
procedure Moveto
  (Clist              : access Gtk_Clist_Record;
   Row                : in    Gint;
   Column             : in    Gint;
   Row_Align          : in    Gfloat;
   Col_Align          : in    Gfloat);
```

Scroll the list so that Row/Column is visible.
If Row is -1, the clist is not scrolled vertically. If Column is -1, the clist is not scrolled horizontally. The new location of Row/Column depends on the value of Row_Align and Col_Align (from 0.0x0.0 (top-left) to 1.0x1.0 (bottom-right), all intermediate values are possible).

89.4.8 Row_Data

You can associate one private data with each row in the clist. If you@* want to store multiple values, you should create a record type that contains all the values, and associate with value with the relevant line in the clist. This package is the equivalent of Gtk.Widget.User_Data for the Clists.

This is your responsibility to use the Get and Set functions from the same generic package. However, you can use different packages for different lines (although this will definitely make things harder to use!)

Note also that an internal copy of the Data is done, therefore the "find" functions found in gtk+ have no equivalent in GtkAda, although it would be enough to write one by iterating over the Row numbers.

```
function Get
  (Object             : access Gtk_Clist_Record'Class;
   Row                : in    Gint)
   return Data_Type;
```

Get the data associated to a specific row.

```
function Get
  (Object             : access Gtk_Clist_Record'Class;
```

```
   Row                : in    Gtk_Clist_Row)
   return Data_Type;
```

Same as above, but acts directly on a row obtained through Get_Row_List. This is faster for big lists.

```
procedure Set
  (Object             : access Gtk_Clist_Record'Class;
   Row                : in    Gint;
   Data               : in    Data_Type);
```

Modify the data associated with a row

```
procedure Set
  (Object             : access Gtk_Clist_Record'Class;
   Row                : in    Gtk_Clist_Row;
   Data               : in    Data_Type);
```

Same as above but acts directly on a row obtained through Get_Row_List. This is faster for big lists.

89.5 Example

```
--  The procedure below shows how you can hide all the columns but one
--  in the clist.
--  Since Gtk_Clist prevents you to hide the last visible column, the fol-
lowing
--  code does not work:
--
--     --  Hide all the columns
--     for J in 0 .. Get_Columns (Clist) loop
--        Set_Column_Visibility (Clist, J, False);
--     end loop;
--
--     --  Show the one you want
--     Set_Column_Visibility (Clist, New_Column, True);
--
--  The following code should be used instead:

package body Clist is

   procedure Hide_All_But_One (Clist : access Gtk_Clist_Record'Class;
                               New_Column : Gint)
   is
   begin
      --  Make sure that at least one column is visible
      Set_Column_Visibility (Clist, New_Column, True);

      --  Hide all the other columns.
      for J in 0 .. Get_Columns (Clist) loop
         if J /= New_Column then
```

```
            Set_Column_Visibility (Clist, J, False);
         end if;
      end loop;
   end Hide_All_But_One;

end Clist;
```

90 Package Gtk.Color_Button

The Gtk_Color_Button is a button which displays the currently selected color an allows to open a color selection dialog to change the color. It is suitable widget for selecting a color in a preference dialog.

90.1 Signals

- **"color_set"**

  ```
  procedure Handler (Button : access Gtk_Color_Button_Record'Class);
  ```

  The color-set signal is emitted when the user selects a color. When handling this signal, use Get_Color and Get_Alpha to find out which color was just selected. Note that this signal is only emitted when the user changes the color. If you need to react to programmatic color changes as well, use the notify::color signal.

90.2 Subprograms

```
function Get_Type                return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Color_Button.

```
procedure Gtk_New
  (Button              : out   Gtk_Color_Button);

procedure Gtk_New_With_Color
  (Button              : out   Gtk_Color_Button;
   Color               :       Gdk.Color.Gdk_Color);

procedure Set_Color
  (Button              : access Gtk_Color_Button_Record;
   Color               :       Gdk.Color.Gdk_Color);

function Get_Color
  (Button              : access Gtk_Color_Button_Record)
   return Gdk.Color.Gdk_Color;
```

Sets the current color to be Color.

```
procedure Set_Alpha
  (Button              : access Gtk_Color_Button_Record;
   Alpha               :       Guint16);

function Get_Alpha
  (Button              : access Gtk_Color_Button_Record)
   return Glib.Guint16;
```

Sets the current opacity to be Alpha (0 to 65_535).

```
procedure Set_Use_Alpha
  (Button              : access Gtk_Color_Button_Record;
   Use_Alpha           :       Boolean);

function Get_Use_Alpha
  (Button              : access Gtk_Color_Button_Record)
   return Boolean;
```

Sets whether or not the color button should use the alpha channel.

```
procedure Set_Title
  (Button              : access Gtk_Color_Button_Record;
   Title               :       String);
```

```
function Get_Title
  (Button              : access Gtk_Color_Button_Record)
   return String;
```

Sets the title for the color selection dialog.

91 Package Gtk.Color_Selection

A Gtk_Color_Selection widget is a complex dialog that allows the user to select a color based either on its (Red, Green, Blue) or its (Hue, Saturation, Value). An additional field is provided to select the opacity of the color (this is usually called the alpha channel).

See Gtk.Color_Selection_Dialog for a version of this widget that comes with its own dialog.

See Gtk.Extra.Color_Combo for a different way to select colors.

91.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Box           (Package Gtk.Box)
            \___ Gtk_Color_Selection (Package Gtk.Color_Selection)
```

91.2 Signals

- **"color_changed"**

  ```
  procedure Handler
  (Selection : access Gtk_Color_Selection_Record'Class);
  ```

  Called every time a new color is selected in the dialog

91.3 Types

type Color_Array **is array** (Color_Index) **of** Gdouble;

Array that indicates the currently selected color. All the values are between 0.0 and 1.0 (a percentage value). They should be converted to absolute values before using them to create a new color, with the following piece of code: Absolute := To_Absolute (Color_Array (Index))

type Color_Index **is**
```
(Red, Green, Blue, Opacity);
```

Used as an index to the table used to set and get the currently selected color.

type Gtk_Color_Selection_Change_Palette_With_Screen_Func **is access procedure**
```
(Screen : Gdk.Gdk_Screen;
 Colors : Gdk.Color.Gdk_Color_Array);
```

This function should save the new palette contents, and update the Gtk_Settings property "gtk-color-palette" so all Gtk_Color_Selection widgets will be modified, including the current one. For instance, you would do: Set_String_Property (Get_Default, Gtk_Color_Palette, Palette_To_String (Colors), "Foo");

91.4 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Color_Selection);
function Get_Type              return Glib.GType;
```

Return the internal value associated with a Gtk_Color_Selection.

```
procedure Set_Current_Color
  (Colorsel           : access Gtk_Color_Selection_Record;
   Color              :      Gdk.Color.Gdk_Color);
procedure Get_Current_Color
  (Colorsel           : access Gtk_Color_Selection_Record;
   Color              : out   Gdk.Color.Gdk_Color);
```

Set the current color of the Colorsel. When called for the first time, the original color will the set to Color as well.

```
procedure Set_Previous_Color
  (Colorsel           : access Gtk_Color_Selection_Record;
   Color              :      Gdk.Color.Gdk_Color);
procedure Get_Previous_Color
  (Colorsel           : access Gtk_Color_Selection_Record;
   Color              : out   Gdk.Color.Gdk_Color);
```

Set the previous color. This procedure should not be called without analysis, as it might seem confusing to see that color change. Calling Set_Current_Color for the first time will also set this color.

```
function Is_Adjusting
  (Colorsel           : access Gtk_Color_Selection_Record)
   return Boolean;
```

Get the current state of the Colorsel.
Return TRue if the user is currently dragging a color around, False if the selection has stopped.

```
function To_Absolute
  (Color              :      Gdouble)
   return Gushort;
```

Convert from a percentage value as returned by Get_Color to an absolute value as can be used with Gdk_Color.

```
function To_Percent
  (Color              :      Gushort)
   return Gdouble;
```

Convert from an absolute value as used in Gdk_Color to a percentage value as used in Set_Color.

91.4.1 Opacity

The color selection widget allows you optionally to select the opacity@* of the color

```
procedure Set_Has_Opacity_Control
  (Colorsel           : access Gtk_Color_Selection_Record;
   Has_Opacity        :      Boolean);
function Get_Has_Opacity_Control
  (Colorsel           : access Gtk_Color_Selection_Record)
   return Boolean;
```

Set the Colorsel to use or not use opacity. An additional field is displayed to select the opacity if needed.

```
procedure Set_Previous_Alpha
  (Colorsel           : access Gtk_Color_Selection_Record;
   Alpha              :      Guint16);
function Get_Previous_Alpha
  (Colorsel           : access Gtk_Color_Selection_Record)
   return Guint16;
```

Set the previous opacity to Alpha. This procedure should not be called without analysis, as it might seem confusing to see that value change.

```
procedure Set_Current_Alpha
  (Colorsel           : access Gtk_Color_Selection_Record;
   Alpha              :      Guint16);
function Get_Current_Alpha
  (Colorsel           : access Gtk_Color_Selection_Record)
   return Guint16;
```

Set the current opacity to be Alpha. When called for the first time, the original opacity will be set too.

91.4.2 Palette

The color selection widget can optionally display a palette, which the@* user can change dynamically. This palette helps selecting colors for the user, who can chose faster among a limited set of colors.

```
procedure Set_Has_Palette
  (Colorsel           : access Gtk_Color_Selection_Record;
   Has_Palette        :      Boolean);
function Get_Has_Palette
  (Colorsel           : access Gtk_Color_Selection_Record)
   return Boolean;
```

If Has_Palette is True, then set the Colorsel to show the palette. Hide the palette otherwise.

```
function Palette_From_String
  (Str                :      String)
   return Gdk.Color.Gdk_Color_Array;
```

Parses a color palette string; this string is a colon-separated list of color names readable by Gdk.Color.Parse. An empty array is returned if Str couldn't be parsed

```
function Palette_To_String
  (Colors             :      Gdk.Color.Gdk_Color_Array)
   return String;
```

Encodes a palette as a string, useful for persistent storage.

```
function Set_Change_Palette_With_Screen_Hook
  (Func               :      Gtk_Color_Selection_Change_Palette_With_Screen_Func)
   return Gtk_Color_Selection_Change_Palette_With_Screen_Func;
```

Installs a global function to be called whenever the user tries to modify the palette in a color selection. Return value: the previous change palette hook (that was replaced).

92 Package Gtk.Color_Selection_Dialog

The Gtk_Color_Selection_Dialog provides a standard dialog which allows the user to select a color much like the Gtk_File_Selection provides a standard dialog for file selection.

92.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Window     (Package Gtk.Window)
               \___ Gtk_Dialog (Package Gtk.Dialog)
                  \___ Gtk_Color_Selection_Dialog
                          (Package Gtk.Color_Selection_Dialog)
```

92.2 Subprograms

```
procedure Gtk_New
  (Color_Selection_Dialog  : out   Gtk_Color_Selection_Dialog;
   Title               :      UTF8_String);
```

Create a new Color_Selection_Dialog with a specified title.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Menu.

92.2.1 Functions to get the fields of the dialog

```
function Get_Colorsel
  (Color_Selection_Dialog  : access Gtk_Color_Selection_Dialog_Record)
   return Gtk.Color_Selection.Gtk_Color_Selection;
```

Get the Gtk_Color_Selection widget contained within the dialog.
Use this widget and its Gtk.Color_Selection.Get_Color function to gain access to the selected color. Connect a handler for this widget's color_changed signal to be notified when the color changes.

```
function Get_OK_Button
  (Color_Selection_Dialog  : access Gtk_Color_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Get the OK button widget contained within the dialog.

```
function Get_Cancel_Button
  (Color_Selection_Dialog  : access Gtk_Color_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Get the cancel button widget contained within the dialog.

```
function Get_Help_Button
  (Color_Selection_Dialog  : access Gtk_Color_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Get the help button widget contained within the dialog.

93 Package Gtk.Combo

The Gtk_Combo widget consists of a single-line text entry field and a drop-down list. The drop-down list is displayed when the user clicks on a small arrow button to the right of the entry field.

The drop-down list is a Gtk_List widget and can be accessed using the list member of the Gtk_Combo. List elements can contain arbitrary widgets, but if an element is not a plain label, then you must use the Gtk_List.Set_Item_String function. This sets the string which will be placed in the text entry field when the item is selected.

By default, the user can step through the items in the list using the arrow (cursor) keys, though this behaviour can be turned off with Set_Use_Arrows.

Normally the arrow keys are only active when the contents of the text entry field matches one of the items in the list. If the contents of the entry field do not match any of the list items, then pressing the arrow keys does nothing. However, by calling Set_Use_Arrows_Always you can specify that the arrow keys are always active. If the contents of the entry field does not match any of the items in the list, then pressing the up or down arrow key will set the entry field to the last or first item in the list, respectively.

93.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Box         (Package Gtk.Box)
            \___ Gtk_Combo    (Package Gtk.Combo)
```

93.2 Subprograms

```
procedure Gtk_New
  (Combo_Box          : out   Gtk_Combo);
```

Create a new Gtk_Combo.

```
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Combo.

```
procedure Set_Value_In_List
  (Combo_Box          : access Gtk_Combo_Record;
   Val                :      Boolean := True;
   Ok_If_Empty        :      Boolean := False);
```

Specify whether the value entered in the text entry field must match one of the values in the list. If this is set then the user will not be able to perform any other action until a valid value has been entered. If an empty field is acceptable, the Ok_If_Empty parameter should be True. If the value entered must match one of the values in the list, val should be True.

```
procedure Set_Use_Arrows
  (Combo_Box          : access Gtk_Combo_Record;
   Val                :      Boolean := True);
```

Specify if the arrow (cursor) keys can be used to step through the items in the list. This is on by default.

```
procedure Set_Use_Arrows_Always
  (Combo_Box          : access Gtk_Combo_Record;
   Val                :      Boolean := True);
```

Specify if the arrow keys will still work even if the current contents of the Gtk_Entry field do not match any of the list items.

```
procedure Set_Case_Sensitive
  (Combo_Box          : access Gtk_Combo_Record;
   Val                :      Boolean := True);
```

Specify whether the text entered into the Gtk_Entry field and the text in the list items are case sensitive. This may be useful, for example, when you have called Set_Value_In_List to limit the values entered, but you are not worried about differences in case.

```
procedure Set_Item_String
  (Combo_Box          : access Gtk_Combo_Record;
   Item               :      Gtk.Item.Gtk_Item;
   Item_Value         :      UTF8_String);
```

Set the string to place in the Gtk_Entry field when a particular list item is selected. This is needed if the list item is not a simple label.

```
procedure Set_Popdown_Strings
  (Combo_Box          : access Gtk_Combo_Record;
   Strings            :      String_List.Glist);
```

Set all the items in the popup list.

```
procedure Disable_Activate
  (Combo_Box          : access Gtk_Combo_Record);
```

Disable the standard handler for the <return> key in the entry field. The default behavior is to popdown the combo box list, so that the user can choose from it. However, if you want to add your own callback for the return key, you need to call this subprogram, and connect a handler to the "activate" signal for the entry.

```
procedure Set_Entry
  (Combo_Box          : access Gtk_Combo_Record;
   GEntry             :      Gtk.GEntry.Gtk_Entry);

function Get_Entry
  (Combo_Box          : access Gtk_Combo_Record)
   return Gtk.GEntry.Gtk_Entry;
```

Set the entry field for the combo box.

```
function Get_List
  (Combo_Box          : access Gtk_Combo_Record)
   return Gtk.List.Gtk_List;
```

Return the list of items associated with a Combo_Box. Add (Gtk.Container.Add) Gtk_List_Items to this list to insert new entries in the popdown menu.

```
function Get_Popup_Window
  (Combo_Box          : access Gtk_Combo_Record)
   return Gtk.Window.Gtk_Window;
```

Return the popup window associated with a Combo_Box.

93.3 Example

Creating a Gtk_Combo widget **with** simple text items.

```
Combo : Gtk_Combo;
Items : String_List.Glist;

String_List.Append (Items, "First Item");
String_List.Append (Items, "Second Item");
String_List.Append (Items, "Third Item");
String_List.Append (Items, "Fourth Item");
String_List.Append (Items, "Fifth Item");

Gtk_New (Combo);
Set_Popdown_Strings (Combo, Items);
Free (Items);
```

94 Package Gtk.Combo_Box

A Gtk_Combo_Box is a widget that allows the user to choose from a list of valid choices. The Gtk_Combo_Box displays the selected choice. When activated, the Gtk_Combo_Box displays a popup which allows the user to make new choice. The style in which the selected value is displayed, and the style of the popup is determined by the current theme. It may be similar to a Gtk_Option_Menu, or similar to a Windows-style combo box.

Unlike its predecessors Gtk.Combo.Gtk_Combo and Gtk.Option_Menu.Gtk_Option_Menu, the Gtk_Combo_Box uses the model-view pattern; the list of valid choices is specified in the form of a tree model, and the display of the choices can be adapted to the data in the model by using cell renderers, as you would in a tree view. This is possible since Gtk_Combo_Box implements the Gtk_Cell_Layout interface. The tree model holding the valid choices is not restricted to a flat list, it can be a real tree, and the popup will reflect the tree structure.

In addition to the model-view API, Gtk_Combo_Box offers a simple API which is suitable for text-only combo boxes, and hides the complexity of managing the data in a model.

94.1 Signals

- **"changed"**

  ```
  procedure Handler (Combo : access Gtk_Combo_Box_Record'Class);
  ```

  Emitted when the active item is changed. The can be due to the user selecting a different item from the list, or due to a call to Set_Active_Iter. It will also be emitted while typing into a Gtk_Combo_Box_Entry, as well as when selecting an item from the Gtk_Combo_Box_Entry's list.

94.2 Subprograms

```
procedure Gtk_New
  (Combo               : out   Gtk_Combo_Box);
procedure Gtk_New_With_Model
  (Combo               : out   Gtk_Combo_Box;
   Model               : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class);
function Get_Type                return Glib.GType;
```

Returns the internal value used for Gtk_Combo_Box widgets

```
procedure Set_Model
  (Combo_Box           : access Gtk_Combo_Box_Record;
   Model               :      Gtk.Tree_Model.Gtk_Tree_Model
                        := null);
function Get_Model
  (Combo_Box           : access Gtk_Combo_Box_Record)
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Sets the model used by Combo_Box to be Model. Will unset a previously set model (if applicable). If model is null, then it will unset the model. Note that this function does not clear the cell renderers, you have to call Gtk.Cell_Layout.Clear yourself if you need to set up different cell renderers for the new model.

```
procedure Set_Active
  (Combo_Box           : access Gtk_Combo_Box_Record;
   Index               :      Gint);
```

```
function Get_Active
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gint;
```

Returns the index of the currently active item, or -1 if there's no
active item. If the model is a non-flat treemodel, and the active item is not an immediate child of the root of the tree, this function returns Gtk.Tree_Model.Get_Indices (Path)[0], where Path is the Gtk_Tree_Path of the active model.

```
procedure Set_Active_Iter
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter);

function Get_Active_Iter
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gtk.Tree_Model.Gtk_Tree_Iter;
```

Sets the current active item to be the one referenced by Iter.
Iter must correspond to a path of depth one.

```
procedure Set_Wrap_Width
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Width              :      Gint);

function Get_Wrap_Width
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gint;
```

Returns the wrap width which is used to determine the number
of columns for the popup menu. If the wrap width is larger than 1, the combo box is in table mode. This can be used for instance to display a matrix of color (a color palette to choose from). See also Set_Column_Span_Column

```
procedure Set_Add_Tearoffs
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Add_Tearoffs       :      Boolean);

function Get_Add_Tearoffs
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Boolean;
```

Sets whether the popup menu should have a tearoff menu item.
Clicking on this menu will detach the combo into a floating window that the user can put anywhere on the screen.

```
procedure Set_Column_Span_Column
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Column_Span        :      Gint);

function Get_Column_Span_Column
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gint;
```

Sets the column with column span information for Combo_Box to be
Column_Span. The column span column contains integers which indicate how many columns an item should span. This applies to grid combos, see also Set_Wrap_Width.

```
procedure Set_Row_Span_Column
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Row_Span           :      Gint);

function Get_Row_Span_Column
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gint;
```

Sets the column with row span information for Combo_Box to be Row_Span.
The row span column contains integers which indicate how many rows an item should span.

```
procedure Set_Focus_On_Click
  (Combo              : access Gtk_Combo_Box_Record;
   Focus_On_Click     :      Boolean);

function Get_Focus_On_Click
  (Combo              : access Gtk_Combo_Box_Record)
   return Boolean;
```

Sets whether the combo box will grab focus when it is clicked with
the mouse. Making mouse clicks not grab focus is useful in places like toolbars where you don't want the keyboard focus removed from the main area of the application.

```
procedure Set_Row_Separator_Func
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Func               :      Gtk.Tree_View.Gtk_Tree_View_Row_Separator_Func;
   Data               :      System.Address;
   Destroy            :      Glib.G_Destroy_Notify_Address
                      := null);

function Get_Row_Separator_Func
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return Gtk.Tree_View.Gtk_Tree_View_Row_Separator_Func;
```

Sets the row separator function, which is used to determine
whether a row should be drawn as a separator. If the row separator function is null, no separators are drawn. This is the default value.

94.2.1 Text-only combo boxes

If your combo box only contains text, you do not necessarily have to go@* through the more complex use of a Gtk_Tree_Model.

```
procedure Gtk_New_Text
  (Combo              : out   Gtk_Combo_Box);

procedure Append_Text
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Text               :      String);

procedure Prepend_Text
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Text               :      String);

procedure Insert_Text
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Position           :      Gint;
   Text               :      String);
```

Adds Text to the list of strings stored in Combo_Box. Note that
you can only use this function with combo boxes constructed with Gtk_New_Text.

```
procedure Remove_Text
  (Combo_Box          : access Gtk_Combo_Box_Record;
   Position           :      Gint);
```

Removes the string at Position from Combo_Box. Note that you can only
use this function with combo boxes constructed with Gtk_New_Text.

```
function Get_Active_Text
  (Combo_Box          : access Gtk_Combo_Box_Record)
   return String;
```

Returns the currently active string in Combo_Box or "" if none
is selected. Note that you can only use this function with combo boxes constructed with Gtk_New_Text.

94.2.2 Programmatic Control

```
procedure Popdown
  (Combo_Box          : access Gtk_Combo_Box_Record);
procedure Popup
  (Combo_Box          : access Gtk_Combo_Box_Record);
```

Hides or pops up the menu or dropdown list of Combo_Box.
This function is mostly intended for use by accessibility technologies; applications should have little use for it.

94.2.3 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Cell_Layout" This interface should be used to add new renderers to the view, to render various columns of the model @item "Gtk_Cell_Editable" This interface should be used to edit the contents of a tree model cell @end itemize

```
function "+"
  (Box                : access Gtk_Combo_Box_Record'Class)
   return Gtk.Cell_Layout.Gtk_Cell_Layout;
function "-"
  (Layout             :      Gtk.Cell_Layout.Gtk_Cell_Layout)
   return Gtk_Combo_Box;
```

Converts to and from the Gtk_Cell_Layout interface

```
function "+"
  (Box                : access Gtk_Combo_Box_Record'Class)
   return Gtk.Cell_Editable.Gtk_Cell_Editable;
function "-"
  (Editable           :      Gtk.Cell_Editable.Gtk_Cell_Editable)
   return Gtk_Combo_Box;
```

Converts to and from the Gtk_Cell_Editable interface

95 Package Gtk.Combo_Box_Entry

A Gtk_Combo_Box_Entry is a widget that allows the user to choose from a list of valid choices or enter a different value. It is very similar to Gtk_Combo_Box, but it displays the selected value in an entry to allow modifying it.

In contrast to a Gtk_Combo_Box, the underlying model of a Gtk_Combo_Box_Entry must always have a text column (see Set_Text_Column), and the entry will show the content of the text column in the selected row. To get the text from the entry, use Gtk.Combo_Box.Get_Active_Text.

The changed signal will be emitted while typing into a Gtk_Combo_Box_Entry, as well as when selecting an item from the Gtk_Combo_Box_Entry's list. Use Gtk.Combo_Box.Get_Active or Gtk.Combo_Box.Get_Active_Iter to discover whether an item was actually selected from the list.

Connect to the activate signal of the Gtk_Entry (use Gtk.Bin.Get_Child) to detect when the user actually finishes entering text.

The convenience API to construct simple text-only Gtk_Combo_Box can also be used with Gtk_Combo_Box_Entry which have been constructed with Gtk_New_Text.

95.1 Subprograms

```
procedure Gtk_New
  (Combo               : out   Gtk_Combo_Box_Entry);

procedure Gtk_New_Text
  (Combo               : out   Gtk_Combo_Box_Entry);

procedure Gtk_New_With_Model
  (Combo               : out   Gtk_Combo_Box_Entry;
   Model               : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;
   Text_Column         :      Gint);

function Get_Type               return Glib.GType;
```

Returns the internal value used for Gtk_Combo_Box_Entry widgets

```
procedure Set_Text_Column
  (Entry_Box           : access Gtk_Combo_Box_Entry_Record;
   Text_Column         :      Gint);

function Get_Text_Column
  (Entry_Box           : access Gtk_Combo_Box_Entry_Record)
   return Gint;
```

Sets the model column which Entry_Box should use to get strings from
to be Text_Column.

95.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Cell_Layout" This interface should be used to add new renderers to the view, to render various columns of the model @item "Gtk_Cell_Editable" This interface should be used to edit the contents of a tree model cell @end itemize

```
function "+"
  (Box                 : access Gtk_Combo_Box_Entry_Record'Class)
   return Gtk.Cell_Layout.Gtk_Cell_Layout;
```

```
function "-"
  (Layout              :      Gtk.Cell_Layout.Gtk_Cell_Layout)
   return Gtk_Combo_Box_Entry;
```

Converts to and from the Gtk_Cell_Layout interface

```
function "+"
  (Box                 : access Gtk_Combo_Box_Entry_Record'Class)
   return Gtk.Cell_Editable.Gtk_Cell_Editable;
function "-"
  (Editable            :      Gtk.Cell_Editable.Gtk_Cell_Editable)
   return Gtk_Combo_Box_Entry;
```

Converts to and from the Gtk_Cell_Editable interface

96 Package Gtk.Container

Base class for widgets that have children.

When writing your own container widgets, you need to fully handle the size_allocate event, by also resizing all the children (based on their size requisition). The size_allocate event will always be sent to the parent when a child calls Gtk.Widget.Queue_Resize.

96.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
```

96.2 Signals

- **"add"**

  ```
  procedure Handler (Container : access Gtk_Container_Record'Class;
  Widget    : access Gtk_Widget_Record'Class);
  ```

  A new widget is added to the container

- **"check_resize"**

  ```
  procedure Handler (Container : access Gtk_Container_Record'Class);
  ```

  Called every time the Container needs resizing. Upon receiving this signal, Container should check whether it needs to be resized, and if it does should queue a resize request.

- **"focus"**

  ```
  procedure Handler (Container : access Gtk_Container_Record'Class;
  Direction : Gtk_Direction_Type);
  ```

  Moves the current selection to a new widget.

- **"remove"**

  ```
  procedure Handler (Container : access Gtk_Container_Record'Class;
  Widget    : access Gtk_Widget_Record'Class);
  ```

  A widget is removed from the container

- **"set-focus-child"**

  ```
  procedure Handler (Container : access Gtk_Container_Record'Class;
  Widget    : access Gtk_Widget_Record'Class);
  ```

  Emitted when a new widget gains the focus.

96.3 Types

```
type Gtk_Callback is access procedure
        (Item : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Function that can be call for each child of a container. This is called automatically by the Forall subprogram below.

96.4 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Container.

```
procedure Set_Border_Width
  (Container          : access Gtk_Container_Record;
   Border_Width       :      Guint);
function Get_Border_Width
  (Container          : access Gtk_Container_Record)
   return Guint;
```

Modify the size of the frame that surrounds the widget.
The exact visual impact depends on the specific widget class.

```
procedure Add
  (Container          : access Gtk_Container_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add a new child to the container.
Note that some containers can have only one child. Nothing is done if there is already a child. This basically sends the "add" signal (see below)

```
procedure Remove
  (Container          : access Gtk_Container_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Removes Widget from Container. Widget must be inside Container.
Note that Container will own a reference to Widget, and that this may be the last reference held; so removing a widget from its container can destroy that widget. If you want to use Widget again, you need to add a reference to it while it's not inside a container, using Glib.Object.Ref. If you don't want to use Widget again it's usually more efficient to simply destroy it directly using Gtk.Widget.Destroy since this will remove it from the container and help break any circular reference count cycles.

```
procedure Set_Resize_Mode
  (Container          : access Gtk_Container_Record;
   Resize_Mode        :      Gtk.Enums.Gtk_Resize_Mode);
function Get_Resize_Mode
  (Container          : access Gtk_Container_Record)
   return Gtk.Enums.Gtk_Resize_Mode;
```

Change the resizing behavior for the Container.
The default value is Resize_Parent.

```
function Get_Children
  (Container          : access Gtk_Container_Record)
   return Gtk.Widget.Widget_List.Glist;
```

Return a list of all the children of the container.
The caller must free the returned list.

```
procedure Propagate_Expose
  (Container          : access Gtk_Container_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Event              :      Gdk.Event.Gdk_Event_Expose);
```

When a container receives an expose event, it must send synthetic
expose events to all children that don't have their own Gdk_Window. This function provides a convenient way of doing this. A container, when it receives an expose event, Propagate_Expose once for each child, passing in the event the container received.

Propagate_Expose takes care of deciding whether an expose event needs to be sent to the child, intersecting the event's area with the child area, and sending the event.

In most cases, a container can simply either simply inherit the expose implementation from Gtk_Container, or, do some drawing and then chain to the expose implementation from Gtk_Container.

96.4.1 Focus

```
procedure Set_Focus_Chain
  (Container          : access Gtk_Container_Record;
   Focusable_Widgets  :      Gtk.Widget.Widget_List.Glist);
```

Set the chain of widgets that can take the focus for a given Container.
The list should be freed by the user. This list indicates in which order the widgets will get the focus when the user presses tab or the arrow keys to move from one widget to the next.

```
procedure Get_Focus_Chain
  (Container          : access Gtk_Container_Record;
   Focusable_Widgets  : out   Gtk.Widget.Widget_List.Glist;
   Success            : out   Boolean);
```

Retrieves the focus chain of the container, if one has been
set explicitly. If no focus chain has been explicitly set, GTK+ computes the focus chain based on the positions of the children. In that case, GTK+ stores null in Focusable_Widgets and returns FALSE. The returned list must be freed by the user.

```
procedure Unset_Focus_Chain
  (Container          : access Gtk_Container_Record);
```

Undoes the effect of Set_Focus_Chain

```
procedure Set_Focus_Vadjustment
  (Container          : access Gtk_Container_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);

function Get_Focus_Vadjustment
  (Container          : access Gtk_Container_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Set the focus to the vertical adjustment.
Adjustment should have been created and displayed at some other place in your application. Container will make sure that Adjustment always matches the range for the focus widget's position (y .. y + height).

```
procedure Set_Focus_Hadjustment
  (Container          : access Gtk_Container_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);

function Get_Focus_Hadjustment
  (Container          : access Gtk_Container_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Set the focus to the horizontal adjustment.
Adjustment should have been created and displayed at some other place in your application. Container will make sure that Adjustment always matches the range for the focus widget's position (x .. x + width).

```
procedure Set_Focus_Child
  (Container          : access Gtk_Container_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class);

function Get_Focus_Child
  (Container          : access Gtk_Container_Record)
```

```
   return Gtk.Widget.Gtk_Widget;
```

Emit a "set_focus_child" signal, to set the child that currently has the keyboard focus.

96.4.2 Properties

```
procedure Child_Set_Property
  (Container          : access Gtk_Container_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Property_Name      :      String;
   Value              :      Glib.Values.GValue);
procedure Child_Get_Property
  (Container          : access Gtk_Container_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Property_Name      :      String;
   Value              : out   Glib.Values.GValue);
```

Sets or Gets the value of a child property for Child and Container. This is property set at the container level, and that applies to all children of that container. These are special type of properties, different from the properties associated with each type of widget. See also Gtk.Widget.Child_Notify You should use Glib.Property_Name to get the name from the property declaration in each of the GtkAda packages

```
function Class_Find_Child_Property
  (Cclass             :      Glib.Object.GObject_Class;
   Property_Name      :      String)
   return Glib.Param_Spec;
```

Finds a child property of a container class by name. The returned value describes the property (type, allowed range, description,...) You should use Glib.Property_Name to get the name from the property declaration in each of the GtkAda packages

```
procedure Class_Install_Child_Property
  (Cclass             :      Glib.Object.GObject_Class;
   Property_Id        :      Guint;
   Pspec              :      Glib.Param_Spec);
```

Installs a child property on a container class.
The Property_Id is an custom id that you choose for your class. It will be used in signals that set or get the property, instead of passing around a string.

```
function Class_List_Child_Properties
  (Cclass             :      Glib.Object.GObject_Class)
   return Glib.Param_Spec_Array;
```

Returns all child properties of a container class.

96.4.3 Forall functions

```
procedure Forall
  (Container          : access Gtk_Container_Record;
   Func               :      Gtk_Callback);
```

Invokes Func on each child of Container, including children that are considered "internal" (implementation details of the container). "Internal" children generally weren't added by the user of the container, but were added by the container implementation itself. See Gtk.Widget.Set_Composite_Name. Most applications should use gtk_container_foreach(), rather than gtk_container_forall(). See also the generic package Forall_Pkg if you want to pass some extra data to Func.

```
procedure Foreach
  (Container          : access Gtk_Container_Record;
   Func               :      Gtk_Callback);
```

Invokes Func on each non-internal child of Container. See Forall for details on what constitutes an "internal" child.

96.4.4 Widget-level methods

```
procedure Set_Reallocate_Redraws
  (Container          : access Gtk_Container_Record;
   Needs_Redraws      :      Boolean := False);
```

Set the "needs_redraws" field.
If Needs_Redraws is True, then a "draw" signal is emitted for the Container whenever one is emitted for a child.

```
function Child_Type
  (Container          : access Gtk_Container_Record)
   return Gtk.Gtk_Type;
```

Return the type of the children in Container.
If Container can contain any type of widget, Gtk_Type_None is returned.

```
procedure Resize_Children
  (Container          : access Gtk_Container_Record);
```

The container hasn't changed size but one of its children queued a resize request. Which means that the allocation is not sufficient for the requisition of some child. Run through the list of widgets and reallocate their size appropriately.

96.4.5 Signals emission

```
procedure Check_Resize
  (Container          : access Gtk_Container_Record);
```

Emit the "check_resize" signal

97 Package Gtk.Ctree

This widget is deprecated. Use Gtk.Tree_View instead.

This widget is similar to Gtk_Clist but it displays a tree with expandable nodes instead of a simple list. Gtk_Tree is a more flexible tree widget (it can have arbitrary widgets in the tree cells), but it is less efficient and is limited to 32768 pixels.

If you need horizontal or vertical scrolling, you mustn't put this widget in a Gtk_Viewport, and then in a Gtk_Scrolled_Window. Put it directly into a Gtk_Scrolled_Window, or horizontal scrolling will be disabled, and the column headers will disappear when scrolling vertically.

97.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Clist       (Package Gtk.Clist)
            \___ Gtk_Ctree    (Package Gtk.Ctree)
```

97.2 Signals

- **"tree_collapse"**

  ```
  procedure Handler (Ctree  : access Gtk_Clist_Record'Class;
  Node   : Gtk_Ctree_Node);
  ```

  Emitted when the subtree associated with a Node is collapsed.

- **"tree_expand"**

  ```
  procedure Handler (Ctree  : access Gtk_Clist_Record'Class;
  Node   : Gtk_Ctree_Node);
  ```

  Emitted when the subtree associated with a Node is expanded.

- **"tree_move"**

  ```
  procedure Handler (Ctree       : access Gtk_Clist_Record'Class;
  Node        : Gtk_Ctree_Node);
  ```

  New_Parent : Gtk_Ctree_Node); New_Sibling : Gtk_Ctree_Node);

  Emitted when a Node is moved (e.g its parent and/or its sibling changed).

- **"tree_select_row"**

  ```
  procedure Handler (Ctree  : access Gtk_Ctree_Record'Class;
  Node   : Gtk_Ctree_Node;
  Column : Gint);
  ```

  Emitted to request the selection of a node. Column is the column number where the user clicked.

- **"tree_unselect_row"**

  ```
  procedure Handler (Ctree  : access Gtk_Ctree_Record'Class;
  Node   : Gtk_Ctree_Node;
  Column : Gint);
  ```

  Emitted to request the unselection of a node.

97.3 Types

```
type Gcompare_Func is access function
          (A, B : in Data_Type) return Boolean;
```

Function used to compare data types in the functions Find_[All] By_Row_Data_Custom.

```
type Gtk_Ctree_Compare_Drag_Func is access function
          (Ctree        : in Gtk_Ctree;
           Source_Node  : in Gtk_Ctree_Node;
           New_Parent   : in Gtk_Ctree_Node;
           New_Sibling  : in Gtk_Ctree_Node) return Boolean;
```

Function type used in Set_Drag_Compare_Func.

```
type Gtk_Ctree_Func is access procedure
          (Ctree : access Gtk_Ctree_Record'Class;
           Node  : in     Gtk_Ctree_Node;
           Data  : in     Data_Type_Access);
```

Function used by Post/Pre_Recursive functions below.

type Gtk_Ctree_Node **is new** Gdk.C_Proxy;

This type represents a node inside a Ctree.

type Gtk_Ctree_Row **is new** Gtk.Clist.Gtk_Clist_Row;

97.4 Subprograms

97.4.1 Creation, insertion, deletion

Elements inside a Gtk_Ctree are not ordered from the top to the bottom@* as is the case for Gtk_Clist. Instead, they are put in the ctree by indicating where in the tree they should be placed. The position of an element (called a node) is defined by a parent node and a sibling node. The node will be attached in the parent subtree, on top of the sibling node.

```
procedure Gtk_New
  (Widget             : out   Gtk_Ctree;
   Columns            : in    Gint;
   Tree_Column        : in    Gint := 0);
```

Create a ctree with Columns columns.
Tree_Column indicates in which column the tree will be displayed.

```
procedure Gtk_New
  (Widget             : out   Gtk_Ctree;
   Titles             : in    Chars_Ptr_Array;
   Tree_Column        : in    Gint := 0);
```

Create a ctree with Titles'Length columns.
Titles gives the title of each column. Tree_Column indicates in which column the tree will be displayed.

```
function Get_Type                return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Ctree.

```
function Insert_Node
  (Ctree               : access Gtk_Ctree_Record;
   Parent              : in    Gtk_Ctree_Node;
   Sibling             : in    Gtk_Ctree_Node;
   Text                : in    Chars_Ptr_Array;
   Spacing             : in    Guint8;
   Pixmap_Closed       : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask_Closed         : in    Gdk.Bitmap.Gdk_Bitmap;
   Pixmap_Opened       : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask_Opened         : in    Gdk.Bitmap.Gdk_Bitmap;
   Is_Leaf             : in    Boolean;
   Expanded            : in    Boolean)
   return Gtk_Ctree_Node;
```

Insert a new node in the Ctree.
Parent is the parent node. If null, the new node is part of the root. The new node will be inserted right on top of Sibling. If Sibling is null, then it will be the first node in the subtree. Text contains the text for each cell of the node. Note that Insert_Node expects the length of the Text parameter to be equal to the number of columns of the Ctree. Spacing is the number of pixels between the lines of the tree and the text in the same column. If Is_Leaf is True, then the node won't contain any subtree. If False, the newly created node can be used as the Parent for further node creation. In this case, Expanded indicates whether the subtree associated with this node should be initially visible. In addition to the "+" or "-" sign indicating whether the subtree is expanded or not, it is possible to put a pixmap giving this information. Pixmap_Closed and Mask_Closed represent the image and the mask used when the subtree is closed; similarly, Pixmap_Opened and Mask_Opened represent the image and the mask used when the subtree is opened.

```
procedure Remove_Node
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Remove Node from Ctree.

97.4.2 Tree, Node and Row basic manipulation

```
function Get_Tree_Column
  (Widget              : access Gtk.Ctree.Gtk_Ctree_Record'Class)
   return Gint;
```

Return the Tree_Column attribute of a given Node.
Tree_Column indicates in which column the tree will be displayed.

```
function Get_Node_List
  (Ctree               : access Gtk_Ctree_Record)
   return Node_List.Glist;
```

Return the list of nodes associated with a given Ctree.
Note: you need to extract the nodes with Node_List.Get_Gpointer.

```
function Get_Row_List
  (Ctree               : access Gtk_Ctree_Record)
   return Row_List.Glist;
```

Return the list of rows associated with a given Ctree.

```
function Get_Selection
  (Ctree               : access Gtk_Ctree_Record)
```

```
   return Node_List.Glist;
```

Return the list of nodes currently selected.
Extract the nodes with Node_List.Get_Data

```
function Node_Get_Row
  (Node               : in    Gtk_Ctree_Node)
   return Gtk_Ctree_Row;
```

Return the row of a given Node.

```
function Row_Get_Children
  (Row                : in    Gtk_Ctree_Row)
   return Gtk_Ctree_Node;
```

Return the children node of a given Row.

```
function Row_Get_Expanded
  (Row                : in    Gtk_Ctree_Row)
   return Boolean;
```

Return the expanded attribute of a given Row.
Note that Expanded can also be retrieved via Get_Node_Info, this function is just a quick accessor.

```
function Row_Get_Is_Leaf
  (Row                : in    Gtk_Ctree_Row)
   return Boolean;
```

Return the leaf attribute of a given Row.
Note that Is_Leaf can also be retrieved via Get_Node_Info, this function is just a quick accessor.

```
function Row_Get_Parent
  (Row                : in    Gtk_Ctree_Row)
   return Gtk_Ctree_Node;
```

Return the parent node of a given Row.

```
function Row_Get_Sibling
  (Row                : in    Gtk_Ctree_Row)
   return Gtk_Ctree_Node;
```

Return the sibling node of a given Row.

```
function Is_Created
  (Node               : in    Gtk_Ctree_Node)
   return Boolean;
```

Return True if Node is different from Null_Ctree_Node

97.4.3 Querying / finding tree information

```
function Is_Viewable
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node)
   return Boolean;
```

Return True if Node is viewable.
A Node is viewable if all the trees and subtrees containing it are expanded.

```
function Last
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node)
   return Gtk_Ctree_Node;
```

Return the last node of a given subtree.
Starting at Node, this function will recursively look for the last sibling of the last child. Return an empty node is Node is empty.

```
function Find_Node_Ptr
  (Ctree              : access Gtk_Ctree_Record;
   Ctree_Row          : in    Gtk_Ctree_Row)
   return Gtk_Ctree_Node;
```

Return the node corresponding to a given row.

```
function Node_Nth
  (Ctree              : access Gtk_Ctree_Record;
   Row                : in    Guint)
   return Gtk_Ctree_Node;
```

Return the Node corresponding to the nth row of a given Ctree.
This can be used to retrieve the root node of the tree, by passing 0 for Row.

```
function Find
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Child              : in    Gtk_Ctree_Node)
   return Boolean;
```

Recursively search for a given Child in a given subtree.
the subtree is determined by Node. If Node is empty, the search will occur on the whole tree. Return True if Child is found, False otherwise.

```
function Is_Ancestor
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Child              : in    Gtk_Ctree_Node)
   return Boolean;
```

Indicate whether Node is an ancestor of Child.
It is assumed that Node is not empty.

```
function Is_Hot_Spot
  (Ctree              : access Gtk_Ctree_Record;
   X                  : in    Gint;
   Y                  : in    Gint)
   return Boolean;
```

Return True if the Ctree is centered on (x,y)

97.4.4 Tree signals: move, expand, collapse, (un)select

```
procedure Move
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   New_Parent         : in    Gtk_Ctree_Node;
   New_Sibling        : in    Gtk_Ctree_Node);
```

Move a node in a Ctree.
After its creation, a node can be moved. New_Parent points to the new parent node that will contain Node. If null, Node will be attached to the root. New_Sibling indicates under which node Node will be inserted. If New_Sibling is null, the new node will be the lowest in its branch.

```
procedure Expand
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node);
```

Expand the first level of the subtree associated with Node.

```
procedure Expand_Recursive
  (Ctree              : access Gtk_Ctree_Record;
```

```
   Node                : in    Gtk_Ctree_Node := null);
```

Expand the whole subtree associated with Node.

```
procedure Expand_To_Depth
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node := null;
   Depth               : in    Gint);
```

Expand the subtree associated with Node and its descendants until Depth levels of subtrees have been reached.

```
procedure Collapse
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Collapse the first level of the subtree associated with Node.

```
procedure Collapse_Recursive
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node := null);
```

Collapse the whole subtree associated with Node.

```
procedure Collapse_To_Depth
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node := null;
   Depth               : in    Gint);
```

Collapse the subtree associated with Node and its descendants until Depth levels of subtrees have been reached.

```
procedure Toggle_Expansion
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Change the state of the Ctree from expanded to collapsed and the other way around on one level.

```
procedure Toggle_Expansion_Recursive
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Change the state of the Ctree from expanded to collapsed and the other way around for the whole subtree.

```
procedure Gtk_Select
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Select a specified Node, and only this one.

```
procedure Select_Recursive
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node := null);
```

Select a specified Node, and its whole subtree.

```
procedure Unselect
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node);
```

Unselect a specified Node, and only this one.

```
procedure Unselect_Recursive
  (Ctree               : access Gtk_Ctree_Record;
   Node                : in    Gtk_Ctree_Node := null);
```

Unselect a specified Node, and its whole subtree.

```
procedure Real_Select_Recursive
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node := null;
   Do_Select          : in    Boolean);
```

Similar to Select_Recursive or Unselect_Recursive.
If Do_Select is True, equivalent to Select_Recursive. If Do_Select is False, equivalent to Unselect_Recursive.

97.4.5 Analogs of Gtk_Clist functions

```
procedure Node_Set_Text
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Text               : in    UTF8_String);
```

Set the cell's text, replacing its current contents.
This changes the type of the cell to Cell_Text. The pixmap (if any) will no longer be displayed.

```
function Node_Get_Text
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint)
   return UTF8_String;
```

Return the text contained in cell.
An empty string is returned if Column is invalid or if the Cell did not contain any text (only a pixmap)

```
procedure Node_Set_Pixmap
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Pixmap             : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask               : in    Gdk.Bitmap.Gdk_Bitmap);
```

Set the cell's pixmap, replacing its current contents.
The type of the cell becomes Cell_Pixmap, and the text is no longer displayed.

```
procedure Node_Get_Pixmap
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap;
   Success            : out   Boolean);
```

Return the Pixmap contained in a cell.
The type of the cell should be Cell_Pixmap. The result is meaningful only if Success is True. If the Cell did not contain a pixmap, Success is set to False.

```
procedure Node_Set_Pixtext
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Text               : in    UTF8_String;
   Spacing            : in    Guint8;
   Pixmap             : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask               : in    Gdk.Bitmap.Gdk_Bitmap);
```

Set both the Text and the Pixmap for the cell.
Replace its current contents. The type of the cell becomes Cell_Pixtext, and both the text and the pixmap are displayed.

```
procedure Node_Get_Pixtext
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Text               : out   Interfaces.C.Strings.chars_ptr;
   Spacing            : out   Guint8;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap;
   Success            : out   Boolean);
```

Return the Text and the Pixmap for the cell.
The result is not meaningful if Success is False.

```
procedure Node_Set_Shift
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Vertical           : in    Gint;
   Horizontal         : in    Gint);
```

Set a horizontal and vertical shift for drawing the content of the cell.
Both shifts can be either positive or negative. This is particularly useful for indenting items in a columns.

```
procedure Set_Node_Info
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Text               : in    UTF8_String;
   Spacing            : in    Guint8;
   Pixmap_Closed      : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask_Closed        : in    Gdk.Bitmap.Gdk_Bitmap;
   Pixmap_Opened      : in    Gdk.Pixmap.Gdk_Pixmap;
   Mask_Opened        : in    Gdk.Bitmap.Gdk_Bitmap;
   Is_Leaf            : in    Boolean;
   Expanded           : in    Boolean);
```

Set all the info related to a specific Node.

```
procedure Get_Node_Info
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Text               : out   Interfaces.C.Strings.chars_ptr;
   Spacing            : out   Guint8;
   Pixmap_Closed      : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask_Closed        : out   Gdk.Bitmap.Gdk_Bitmap;
   Pixmap_Opened      : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask_Opened        : out   Gdk.Bitmap.Gdk_Bitmap;
   Is_Leaf            : out   Boolean;
   Expanded           : out   Boolean;
   Success            : out   Boolean);
```

Return all the info related to a specific Node.

```
procedure Node_Set_Selectable
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Selectable         : in    Boolean := True);
```

Indicate whether the Node can be selected or not.
The default value is True.

```
function Node_Get_Selectable
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node)
   return Boolean;
```

Return the selectable status of the Node.

```
procedure Node_Set_Row_Style
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Style              : in    Gtk.Style.Gtk_Style);
```

Set the default style for the cells in the Node.
This can be overridden for each cell with Node_Set_Cell_Style.

```
function Node_Get_Row_Style
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node)
   return Gtk.Style.Gtk_Style;
```

Return the default style used for the Node.

```
procedure Node_Set_Cell_Style
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Style              : in    Gtk.Style.Gtk_Style);
```

Set the style (font, color, ...) used for the cell.
This overrides the Node's style.

```
function Node_Get_Cell_Style
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint)
   return Gtk.Style.Gtk_Style;
```

Return the style of the cell.

```
procedure Node_Set_Foreground
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Color              : in    Gdk.Color.Gdk_Color);
```

Set the foreground color for the Node.
The color must already be allocated. If no such Node exists in the tree, nothing is done.

```
procedure Node_Set_Background
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Color              : in    Gdk.Color.Gdk_Color);
```

Set the background color for the Node.
The color must already be allocated. If no such Node exists in the tree, nothing is done.

```
function Node_Get_Cell_Type
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint)
   return Gtk.Clist.Gtk_Cell_Type;
procedure Node_Moveto
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node;
   Column             : in    Gint;
   Row_Align          : in    Gfloat := 0.5;
   Col_Align          : in    Gfloat := 0.5);
```

Make a Node visible.
Column indicates which column of the Node should be visible, if not all columns can be displayed. Row_Align and Col_Align are parameters between 0.0 and 1.0, and specify how the Node and the Column will be centered in the Ctree window. 0.0 means a Node on the top, and a Column on the left.

```
function Node_Is_Visible
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node)
   return Gtk_Visibility;
```

Indicate the visibility of a Node.
Return Visibility_None if the Node is not visible in the Ctree window; Visibility_Partial if the Node is partially visible; Visibility_Full if the Node is entirely visible. This function ignores the fact that Node is in an expanded or collapsed subtree.

97.4.6 Ctree specific functions

```
procedure Set_Indent
  (Ctree              : access Gtk_Ctree_Record;
   Indent             : in    Gint := 20);
```

Change the indentation of the Ctree.
Each different level of a subtree is indented by a number of pixels. By default, the indentation is 20 pixels, and can be changed using this procedure.

```
function Get_Indent
  (Widget             : access Gtk.Ctree.Gtk_Ctree_Record'Class)
   return Gint;
```

Return the indentation of a Ctree.

```
procedure Set_Spacing
  (Ctree              : access Gtk_Ctree_Record;
   Spacing            : in    Gint := 5);
```

Set the spacing between the tree's icon and the additional pixmap.
The additional pixmap indicates whether the subtree is opened or closed. The default value is 5 pixels.

```
function Get_Spacing
  (Widget             : access Gtk.Ctree.Gtk_Ctree_Record'Class)
   return Gint;
```

Return the spacing between the tree's icon and the additional pixmap.

```
procedure Set_Show_Stub
  (Ctree              : access Gtk_Ctree_Record;
   Show_Stub          : in    Boolean);
```

Set the Show_Stub attribute of Ctree.

```
function Get_Show_Stub
  (Ctree              : access Gtk_Ctree_Record)
   return Boolean;
```

Return the Show_Stub attribute of Ctree.

```
procedure Set_Line_Style
  (Ctree              : access Gtk_Ctree_Record;
   Line_Style         : in    Gtk_Ctree_Line_Style
                       := Ctree_Lines_Solid);
```

Change the style of the lines representing the tree of a given Ctree.
By default, solid lines are used. See the description of Gtk_Ctree_Line_Style for more details of the possible values.

```
function Get_Line_Style
  (Ctree              : access Gtk_Ctree_Record)
   return Gtk_Ctree_Line_Style;
```

return the style of the lines representing the tree of a given Ctree.

```
procedure Set_Expander_Style
  (Ctree              : access Gtk_Ctree_Record;
   Expander_Style     : in    Gtk_Ctree_Expander_Style
                       := Ctree_Expander_Square);
```

Set the way a given Ctree can be expanded.
To expand a subtree, you can either double-click on a node, or click on the "+/-" icon. This icon is by default included in a square pixmap. This procedure can change the form of this pixmap. See the description of Gtk_Ctree_Expander_Style for more details.

```
function Get_Expander_Style
  (Ctree              : access Gtk_Ctree_Record)
   return Gtk_Ctree_Expander_Style;
```

Return the way a given Ctree can be expanded.

```
procedure Set_Drag_Compare_Func
  (Ctree              : access Gtk_Ctree_Record;
   Cmp_Func           : in    Gtk_Ctree_Compare_Drag_Func);
```

Set the drag compare function of a given Ctree.
This function is used when the Ctree receives a dragged data.

97.4.7 Tree sorting functions

```
procedure Sort_Node
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node);
```

Sort the nodes of a given Ctree.
This procedure only sorts the first level of the tree.

```
procedure Sort_Recursive
  (Ctree              : access Gtk_Ctree_Record;
   Node               : in    Gtk_Ctree_Node := null);
```

Sort the nodes of a given Ctree recursively.
This procedure sorts the whole tree and subtrees associated with Ctree. Set Node to null if you want to sort the whole tree starting from its root.

97.4.8 Row_Data handling

```
procedure Node_Set_Row_Data
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Data               : in    Data_Type);
```

Associate a Data with a Node.

```
function Node_Get_Row_Data
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node)
   return Data_Type;
```

Retrieve a data associated with a Node.
Error Handling: Gtkada.Types.Data_Error is raised when trying to retrieve the data from a Node for which no data has been set (using Node_Set_Row_Data).

```
function Find_By_Row_Data
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Data               : in    Data_Type)
   return Gtk_Ctree_Node;
```

Find the first node containing a specified Data.
Node is the starting point of the search. If null, the search will start from the root. Return the first Node whose associated data is Data, null if none can be found.

```
function Find_All_By_Row_Data
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Data               : in    Data_Type)
   return Node_List.Glist;
```

Find all nodes containing a specified Data.
Node is the starting point of the search. If null, the search will start from the root.

```
function Find_By_Row_Data_Custom
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Data               : in    Data_Type;
   Func               : in    Gcompare_Func)
   return Gtk_Ctree_Node;
```

Find the first node containing a specified Data.
Similar to Find_By_Row_Data but Func is used to allow a more flexible (user defined) method to compare two nodes.

```
function Find_All_By_Row_Data_Custom
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Data               : in    Data_Type;
   Func               : in    Gcompare_Func)
   return Node_List.Glist;
```

Find all the nodes containing a specified Data.
Similar to Find_All_By_Row_Data but Func is used to allow a more flexible (user defined) method to compare two nodes.

```
procedure Post_Recursive
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Func               : in    Gtk_Ctree_Func;
   Data               : in    Data_Type_Access);
```

Apply Func to each node of a subtree.
Node designates the root of the subtree. Data will be passed as a parameter to Func. This procedure will first apply Func to the children nodes.

```
procedure Post_Recursive_To_Depth
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Depth              : in    Gint;
   Func               : in    Gtk_Ctree_Func;
   Data               : in    Data_Type_Access);
```

Apply Func to each node of a subtree until a specified Depth.
Node designates the root of the subtree. Data will be passed as a parameter to Func. This function is similar to Post_Recursive except that it stop at a specified subtree depth.

```
procedure Pre_Recursive
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Func               : in    Gtk_Ctree_Func;
   Data               : in    Data_Type_Access);
```

Apply Func to each node of a subtree.
Similar to Post_Recursive but will apply Func to the parent before applying it to its children.

```
procedure Pre_Recursive_To_Depth
  (Ctree              : access Gtk_Ctree_Record'Class;
   Node               : in    Gtk_Ctree_Node;
   Depth              : in    Gint;
   Func               : in    Gtk_Ctree_Func;
   Data               : in    Data_Type_Access);
```

Apply Func to each node of a subtree until a specific Depth.
Similar to Post_Recursive_To_Depth but will apply Func to the parent before applying it to its children.

98 Package Gtk.Curve

The Gtk_Curve widget allows the user to edit a curve covering a range of values. It is typically used to fine-tune color balances in graphics applications like the Gimp.

The Gtk_Curve widget has 3 modes of operation: spline, linear and free. In spline mode the user places points on the curve which are automatically connected together into a smooth curve. In linear mode the user places points on the curve which are connected by straight lines. In free mode the user can draw the points of the curve freely, and they are not connected at all.

98.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Drawing_Area   (Package Gtk.Drawing_Area)
         \___ Gtk_Curve       (Package Gtk.Curve)
```

98.2 Signals

- **"curve-type-changed"**

  ```
  procedure Handler (Curve : access Gtk_Curve_Record'Class);
  ```

  Emitted when the curve type has been changed. The curve type can be changed explicitly with a call to Set_Curve_Type. It is also changed as a side-effect of calling Reset or Set_Gamma.

98.3 Types

```
type Property_Gtk_Curve_Type is new Curve_Type_Properties.Property;
```

98.4 Subprograms

```
procedure Gtk_New
  (Curve              : out   Gtk_Curve);
```

Create a new Curve.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Curve.

```
procedure Reset
  (Curve              : access Gtk_Curve_Record);
```

Reset the curve.

Reset to a straight line from the minimum x & y values to the maximum x & y values (i.e. from the bottom-left to the top-right corners). The curve type is not changed.

```
procedure Set_Gamma
  (Curve              : access Gtk_Curve_Record;
   Gamma              :      Gfloat);
```

Recompute the entire curve using the given gamma value.

A gamma value of 1.0 results in a straight line. Values greater than 1.0 result in a curve

above the straight line. Values less than 1.0 result in a curve below the straight line. The curve type is changed to Curve_Type_Free.

```
procedure Set_Range
  (Curve              : access Gtk_Curve_Record;
   Min_X              :      Gfloat;
   Max_X              :      Gfloat;
   Min_Y              :      Gfloat;
   Max_Y              :      Gfloat);
```

Set the minimum and maximum x & y values of the curve.
The curve is also reset with a call to Reset.

```
procedure Set_Vector
  (Curve              : access Gtk_Curve_Record;
   Vector             :      Gfloat_Array);
procedure Get_Vector
  (Curve              : access Gtk_Curve_Record;
   Vector             : out   Gfloat_Array);
```

Set the vector of points on the curve.
The curve type is set to Curve_Type_Free.

```
procedure Set_Curve_Type
  (Curve              : access Gtk_Curve_Record;
   Curve_Type         :      Gtk_Curve_Type);
```

Set the type of the curve.
The curve will remain unchanged except when changing from a free curve to a linear or spline curve, in which case the curve will be changed as little as possible.

99 Package Gtk.Dialog

Dialog boxes are a convenient way to prompt the user for a small amount of input, eg. to display a message, ask a question, or anything else that does not require extensive effort on the user's part.

Gtkada treats a dialog as a window split horizontally. The top section is a Gtk_Vbox, and is where widgets such as a Gtk_Label or a Gtk_Entry should be packed. The second area is known as the action_area. This is generally used for packing buttons into the dialog which may perform functions such as cancel, ok, or apply. The two areas are separated by a Gtk_Hseparator.

If 'dialog' is a newly created dialog, the two primary areas of the window can be accessed using Get_Vbox and Get_Action_Area as can be seen from the example, below.

A 'modal' dialog (that is, one which freezes the rest of the application from user input), can be created by calling Set_Modal on the dialog.

See Gtkada.Dialogs for a higher level dialog interface.

99.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Window   (Package Gtk.Window)
               \___ Gtk_Dialog (Package Gtk.Dialog)
```

99.2 Signals

- **"close"**

  ```
  procedure Handler (Dialog : access Gtk_Dialog_Record'Class);
  ```

  Emit this signal to force a closing of the dialog.

- **"response"**

  ```
  procedure Handler
  (Dialog      : access Gtk_Dialog_Record'Class;
  Response_Id : Gint);
  ```

  Emitted when an action widget is clicked, the dialog receives a delete event, or the application programmer calls Response. On delete event, the response ID is GTK_RESPONSE_NONE. Otherwise, it depends on which action widget was clicked.

99.3 Types

type Gtk_Dialog_Flags **is mod** 8;

type Gtk_Response_Type **is new** Gint;

Type used for Response_Id's. Positive values are totally user-interpreted. GtkAda will sometimes return Gtk_Response_None if no Response_Id is available. Typical usage is: if Gtk.Dialog.Run (Dialog) = Gtk_Response_Accept then blah; end if;

```
type Response_Type_Array is array (Natural range <>) of Gtk_Response_Type;
```

99.4 Subprograms

99.4.1 Subprograms

```
procedure Gtk_New
  (Dialog             : out   Gtk_Dialog);
```

Create a new dialog.
Widgets should not be packed into this widget directly, but into the vbox and action_area, as described above.

```
procedure Gtk_New
  (Dialog             : out   Gtk_Dialog;
   Title              :       UTF8_String;
   Parent             :       Gtk.Window.Gtk_Window;
   Flags              :       Gtk_Dialog_Flags);
```

Create a new dialog with a specific title, and specific attributes.
Parent is the transient parent for the dialog (ie the one that is used for reference for the flag Destroy_With_Parent, or to compute the initial position of the dialog).

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Dialog.

```
function Get_Action_Area
  (Dialog             : access Gtk_Dialog_Record)
   return Gtk.Box.Gtk_Box;
```

Return the action area box associated with a Dialog.

```
function Get_Vbox
  (Dialog             : access Gtk_Dialog_Record)
   return Gtk.Box.Gtk_Box;
```

Return the vertical box associated with a Dialog.

```
procedure Add_Action_Widget
  (Dialog             : access Gtk_Dialog_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Response_Id        :       Gtk_Response_Type);
```

Add an activatable widget to the action area of Dialog.
When the widget is activated (ie emits the "activate" signal), Dialog will emit the "response" signal with Response_Id.

```
function Add_Button
  (Dialog             : access Gtk_Dialog_Record;
   Text               :       UTF8_String;
   Response_Id        :       Gtk_Response_Type)
   return Gtk.Widget.Gtk_Widget;
```

Add a button with the given text to the dialog. Note that you can also
pass one of the constants defined in Gtk.Stock for the predefined buttons. When the button is clicked, Dialog will emit the "response" signal. The button widget is returned.

```
function Get_Response_For_Widget
  (Dialog             : access Gtk_Dialog_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
```

```
   return Gint;
```

Gets the response id of a widget in the action area of a dialog, or Gtk_Response_None if Widget doesn't have a response Id set

```
procedure Set_Alternative_Button_Order_From_Array
  (Dialog             : access Gtk_Dialog_Record;
   New_Order          :      Response_Type_Array);
```

Sets an alternative button order. If the gtk-alternative-button-order setting is set to %TRUE, the dialog buttons are reordered according to the order of the response ids passed to this function.

By default, GTK+ dialogs use the button order advocated by the Gnome Human Interface Guidelines with the affirmative button at the far right, and the cancel button left of it. But the builtin GTK+ dialogs and message dialogs' do provide an alternative button order, which is more suitable on some platforms, e.g. Windows.

Use this function after adding all the buttons to your dialog.

```
function Gtk_Alternative_Dialog_Button_Order
  (Screen             :      Gdk.Gdk_Screen := null)
   return Boolean;
```

Returns True if dialogs are expected to use an alternative button order on the given screen (or current screen if null) . See Set_Alternative_Button_Order_From_Array for more details about alternative button order.

If you need to use this function, you should probably connect to the ::notify:gtk-alternative-button-order signal on the Gtk_Settings object associated to Screen, in order to be notified if the button order setting changes.

Returns: Whether the alternative button order should be used

```
procedure Set_Response_Sensitive
  (Dialog             : access Gtk_Dialog_Record;
   Response_Id        :      Gtk_Response_Type;
   Setting            :      Boolean);
```

Call Gtk.Widget.Set_Sensitive for all the buttons in the dialog associated with Response_Id.

```
procedure Set_Default_Response
  (Dialog             : access Gtk_Dialog_Record;
   Response_Id        :      Gtk_Response_Type);
```

Set the last widget in the dialog's action area with the given Response_Id as the default widget for Dialog. Pressing Enter will activate this default widget.

```
procedure Set_Has_Separator
  (Dialog             : access Gtk_Dialog_Record;
   Setting            :      Boolean);

function Get_Has_Separator
  (Dialog             : access Gtk_Dialog_Record)
   return Boolean;
```

Set whether the dialog has a separator above the buttons.

```
function Run
  (Dialog             : access Gtk_Dialog_Record)
   return Gtk_Response_Type;
```

Block in a recursive main loop until Dialog emits the "response"
signal, or is destroyed. If the dialog is destroyed, Gtk_Response_None is returned. Otherwise, the response_id from the "response" signal is returned. Run will call Show on the dialog automatically. However, it is your responsability to call Show for any child you have inserted in the dialog. The dialog is automatically set to modal when this function is called. You can exit at any time from this function by emitting the "response" signal directly. When Run returns, you are responsible for hiding or destroying the dialog if necessary.

99.4.2 Signals

```
procedure Response
  (Dialog             : access Gtk_Dialog_Record;
   Response_Id        :      Gtk_Response_Type);
```

Emit the "response" signal

100 Package Gtk.Dnd

Like all modern GUI toolkits, GtkAda has a full support for drag-and-drop operations. This is a mechanism for interactively transferring data between two widgets, either in the same application or in two different applications. The user clicks on a widget (called a "drag source"), and, while keeping the mouse button pressed, moves it to another widget, where the mouse button is released (this other widget is called a "drop site"). As a result, and if both widgets can handle the same type of data, some data is either copied or moved to this new widget.

This is a very intuitive way, in some cases, to enhance the usability of your application, although you should carefully consider whether this should be used or not.

GtkAda supports several drag-and-drop protocols, so as to be able to communicate with the maximum number of applications. These protocols are Xdnd and Motif.

Below is a summary of what is needed to add drag-and-drop capabilities to your application. We highly recommend that you look at, and understand, the example in testgtk (create_dnd.adb), before using these features in your own application.

See also the package Gtk.Selection, that contains some lower subprograms and data types that are used when implementing drag-and-drop.

@itemize @bullet @item Defining a widget as a possible drag source

@end itemize You need to call Source_Set, specifying which mouse buttons can activate the drag, which types of data will be given, and which kind of action will be performed. You then need to connect to the signal "drag_data_get", that will be emitted when the user has dropped the item and GtkAda needs to find the data. You must call Selection_Data_Set in the handler to set the actual data. You can also connect the widget to "drag_data_delete", which will be called whenever the data set for the selection is no longer required and should be deleted. The signal will be emitted only if the drop site requests it, or if the selected action for the drag-and-drop operation was Action_Move. It will not be called automatically for an Action_Copy. Note that the callback might be called several times, if for instance this was an Action_Move, and the drop site requires explicitly to delete the data in its call to Finish.

@itemize @bullet @item Defining a widget as a possible drop site

@end itemize You need to call Dest_Set, specifying which types of Data are accepted by the widget, which actions are recognized, and whether you accept drops from external applications. You also need to connect to "drag_data_received", that will be emitted when the user has dropped some data on the widget. The handler should call Finish, to warn the source widget that the drag and drop operation is finished, and whether it was successful or not.

100.1 Signals

- **"drag_begin"**

  ```
  procedure Handler (Widget  : access Gtk_Widget_Record'Class;
  Context : Drag_Context);
  ```

  A new drag-and-drop operation has just been started from Widget. This callback can be used for instance to modify the visual aspect of the widget, so as to give a visual clue as to what widget is the source.

- **"drag_data_delete"**

```
procedure Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context);
```

  This handler is called whenever the drop site of a drag-and-drop operation has decided that the data should be deleted, or automaticallyif the selected action was Action_Move. Widget is the drag source.

- **"drag_data_get"**

```
procedure Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context;
Data    : Selection_Data;
Info    : Guint;
Time    : Guint);
```

  This should be connected to every drag source. This is used to request the actual data to be transfered to the drop site once the drop has been done. Info is the type of the expected Data, and is in fact the third field of the Target_Entry record, whose value you have define yourself. Data should be modified to include a pointer or a copy of the data, through Selection_Data_Set.

- **"drag_data_received"**

```
procedure Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context;
X       : Gint;
Y       : Gint;
Data    : Selection_Data;
Info    : Guint;
Time    : Guint);
```

  This signal should be connected to every drop site. The handler is called every time some new data has been dropped onto Widget. (X, Y) are the mouse coordinates, relative to the widget's window, where the data was dropped. Info is the type of the data, has set in the third field of the Target_Entry record, and Data contains a pointer to the actual data.

- **"drag_drop"**

```
function Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context;
X       : Gint;
Y       : Gint;
Time    : Guint)
return Boolean;
```

  This is called whenever a drop is about to be performed on the widget. Note that this is called even if no common target type has been found between the drag source and the drop site. Thus, you will need to analyze the result of Get_Targets (Context) to find the possible targets. The data is sent separately through the "drag_data_received" signal, and might not even be available when "drag_drop" is emitted. This signal is mostly used if you have chosen not to use any of the default behavior when calling Dest_Set. Otherwise, everything is already handled directly by GtkAda.

  This handler should return True if Widget acknowledges that it is a possible drop site for the particular targets provided by the drag source.

- **"drag_end"**

```
procedure Handler (Widget  : access Gtk_Widget_Record'Class;
```

```
Context : Drag_Context);
```

The drag-and-drop operation that was started from the widget has been completed, and the standard set of the widget can be restored.

- **"drag_leave"**

```
procedure Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context;
Time    : Guint);
```

Signal emitted whenever a drag-and-drop operation is being performed, and the mouse has just left the area covered by a widget on the screen. This can be used to restore the default visual aspect of the widget. This is also emitted when the drop has been performed on the widget.

- **"drag_motion"**

```
function Handler (Widget  : access Gtk_Widget_Record'Class;
Context : Drag_Context;
X       : Gint;
Y       : Gint;
Time    : Guint)
return Boolean;
```

This is called every time the user is doing a dnd operation, and the mouse is currently over Widget (but not released yet). This can be used to change the visual aspect of Widget to provide visual clues to the user. The "opposite" signal is drag_leave.

The return value is ignored if Dest_Default_Motion was set when Source_Set was called. This handler should return True if Widget acknowledges that it is a possible drop site for the particular targets provided by the drag source.

100.2 Types

type Dest_Defaults **is new** Integer;

Specify the various types of action that will be taken on behalf of the user for a drag destination site.

100.3 Subprograms

100.3.1 Setting up a widget as a destination

```
procedure Dest_Set
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Flags              :      Dest_Defaults
                       := Dest_No_Default;
   Targets            :      Target_Entry_Array
                       := Any_Target_Entry;
   Actions            :      Drag_Action := Action_Any);
```

Set a widget as a potential drop destination.

Flags specifies what action GtkAda should take on behalf of a widget for drops onto that widget. The Targets and Actions fields are used only if Dest_Default_Motion or Dest_Default_Drop are given.

Targets indicates the drop types that Widget accepts. If no item from Targets matches the list of targets emitted by the source (as set in Source_Set), then the drop will be considered illegal and refused.

Actions is a bitmask of possible actions for a drop onto Widget. At least of the actions must be in common with what was set for the source in Source_Set, or the drop is considered illegal.

```
procedure Dest_Set_Proxy
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Proxy_Window       :      Gdk.Window.Gdk_Window;
   Protocol           :      Drag_Protocol;
   Use_Coordinates    :      Boolean);
```

Set this widget as a proxy for drops to another window.
All drag events on Widget will be forwarded to Proxy_Window. Protocol is the drag protocol that Proxy_Window accepts. You can use Gdk.Drag.Get_Protocol to determine this. If Use_Coordinates is True, send the same coordinates to the destination because it is an embedded subwindow.

```
procedure Dest_Unset
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Clear information about a drop destination set with Dest_Set. The
widget will no longer receive notification of drags.

```
procedure Dest_Set_Target_List
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Target_List        :      Gtk.Selection.Target_List);

function Dest_Get_Target_List
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Target_List;
```

Sets the target types that this widget can accept from drag-and-drop.
The widget must first be made into a drag destination with Dest_Set.

```
procedure Dest_Add_Image_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);

procedure Dest_Add_Text_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);

procedure Dest_Add_Uri_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add the image/text/URI targets supported by Gtk_Selection to the target
list of the drag destination. The targets are added with Info = 0. If you need another value, use Gtk.Selection.Target_List_Add_*_Targets, and Dest_Set_Target_List

```
function Dest_Find_Target
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Context            :      Gdk.Dnd.Drag_Context;
   Target_List        :      Gtk.Selection.Target_List)
   return Gdk.Types.Gdk_Atom;
```

Looks for a match between the targets set for context and the
Target_List, returning the first matching target, otherwise returning GDK_NONE. Target_List should usually be the return value from Dest_Get_Target_List, but some widgets may have different valid targets for different parts of the widget; in that case, they will have to implement a drag_motion handler that passes the correct target list to this function.

100.3.2 Setting up a widget as a source

```
procedure Source_Set
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Start_Button_Mask  :      Gdk.Types.Gdk_Modifier_Type;
   Targets            :      Target_Entry_Array;
   Actions            :      Drag_Action);
```

Set up a widget so that GtkAda will start a drag operation when the user clicks and drags on the widget. The widget must have a window.

Targets is the list of targets that the drag can provide. The first possible target accepted by the drop site will be used. For instance, if Targets contains "text/plain" and "text/url", and the drop site only accepts "text/url", this will be the one used. However, if the drop site also accepts "text/plain", the latter will be prefered.

Widget needs to be able to convert the data to any of the types in Target, as any of them might be requested by the drop site.

Actions is a list of possible actions for drags from Widget. At least one of the actions must be in common with the drop site for the drag-and-drop operation to succeed.

```
procedure Source_Unset
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Undo the effects of Source_Set

```
procedure Source_Set_Target_List
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Target_List        :      Gtk.Selection.Target_List);

function Source_Get_Target_List
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Target_List;
```

Changes the target types that this widget offers for drag-and-drop. The widget must first be made into a drag source with Source_Set.

```
procedure Source_Add_Image_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);

procedure Source_Add_Text_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);

procedure Source_Add_Uri_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add the writable image/text/URI targets supported by Gtk_Selection to the target list of the drag source. The targets are added with Info = 0. If you need another value, use Gtk.Selection.Target_List_Add_*_Targets, and Source_Set_Target_List Widget: a #GtkWidget that's is a drag source

```
procedure Source_Set_Icon
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Colormap           :      Gdk.Color.Gdk_Colormap;
   Pixmap             :      Gdk.Pixmap.Gdk_Pixmap;
   Mask               :      Gdk.Bitmap.Gdk_Bitmap);

procedure Source_Set_Icon_Pixbuf
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf);

procedure Source_Set_Icon_Stock
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Stock_Id           :      String);
```

```
procedure Source_Set_Icon_Name
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Icon_Name          :      String);
```

Set the icon that will be used for drags from a particular widget.
GtkAda retains a reference count for the arguments, and will release them when they are no longer needed.

100.3.3 The drag-and-drop operation

```
procedure Finish
  (Context            :      Drag_Context;
   Success            :      Boolean;
   Del                :      Boolean;
   Time               :      Guint32 := 0);
```

Inform the drag source that the drop is finished, and that the data of
the drag will no longer be required. Success should indicate whether the drop was successful. Del should be set to True if the source should delete the original data (this should be True for a move).

```
procedure Get_Data
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Context            :      Drag_Context;
   Target             :      Gdk.Types.Gdk_Atom;
   Time               :      Guint32 := 0);
```

Get the data associated with a drag. When the data is received or the
retrieval fails, GtkAda will emit a "drag_data_received" signal. Failure of the retrieval is indicated by the length field of the selection_data signal parameter being negative. However, when Get_Data is called implicitely because the Drag_Default_Drop was set, then the widget will not receive notification of failed drops.

Target is the target (form of the data) to retrieve. Time is a timestamp to retrive the data, and will be given to "drag_data_motion" or "drag_data_drop" signals.

```
function Get_Source_Widget
  (Context            :      Drag_Context)
   return Gtk.Widget.Gtk_Widget;
```

Determine the source widget for a drag.
If the drag is occuring within a single application, this function returns the source widget. Otherwise, it returns null.

```
procedure Highlight
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Draw a highlight around a widget.

```
procedure Unhighlight
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Remove a highlight set by Highlight.

```
function Drag_Begin
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Targets            :      Target_List;
   Actions            :      Drag_Action;
   Button             :      Gint;
   Event              :      Gdk.Event.Gdk_Event)
   return Drag_Context;
```

Initiate a drag on the source side. The function only needs to be used
when the application is starting drags itself, and is not needed when Source_Set is used.

Targets is the list of targets (data formats) in which the source can provide the data. Actions is a bitmask of the allowed drag actions for this drag. Button is the button the user clicked to start the drag. Event is the event that triggered the start of the drag.

```
function Check_Threshold
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Start_X            :      Gint;
   Start_Y            :      Gint;
   Current_X          :      Gint;
   Current_Y          :      Gint)
   return Boolean;
```

Checks to see if a mouse drag starting at (Start_X, Start_Y) and ending
at (Current_X, Current_Y) has passed the GTK drag threshhold, and thus should trigger the beginning of a drag-and-drop operation. Return True if the drag threshold has been passed.

100.3.4 Icons

```
procedure Set_Icon_Widget
  (Context            :      Drag_Context;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Hot_X              :      Gint;
   Hot_Y              :      Gint);
```

Change the icon for a drag.
GtkAda will not destroy the icon, so if you don't want it to persist, you should connect to the "drag_end" signal and destroy it yourself. Context is the reference to the current drag operation. Widget is the toplevel window to use as an icon. (Hot_X, Hot_Y) is the coordinates of the hot point (that will be just under the mouse) within Widget.

```
procedure Set_Icon_Pixmap
  (Context            :      Drag_Context;
   Colormap           :      Gdk.Color.Gdk_Colormap;
   Pixmap             :      Gdk.Pixmap.Gdk_Pixmap;
   Mask               :      Gdk.Bitmap.Gdk_Bitmap;
   Hot_X              :      Gint;
   Hot_Y              :      Gint);
```

Sets a given pixmap as the icon for a given drag. GtkAda retains a
reference count for the arguments, and will release them when they are no longer needed. (Hot_X, Hot_Y) is the coordinates of the hotspot within Pixmap.

```
procedure Set_Icon_Default
  (Context            :      Drag_Context);
```

Set the icon for a particular drag to the default icon.
This must be called with a context for the source side of a drag.

```
procedure Set_Icon_Pixbuf
  (Context            :      Drag_Context;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf;
   Hot_X              :      Gint;
   Hot_Y              :      Gint);
```

Sets Pixbuf as the icon for a given drag.
Context: the context for a drag. (This must be called with a context for the source side of a drag) Pixbuf: the Gdk_Pixbuf to use as the drag icon. Hot_x: the X offset within the pixbuf of the hotspot. Hot_y: the Y offset within the pixbuf of the hotspot.

```
procedure Set_Icon_Stock
  (Context            :      Drag_Context;
   Stock_Id           :      String;
   Hot_X              :      Gint;
   Hot_Y              :      Gint);
```

Sets the icon for a given drag from a stock ID
Context: the context for a drag. (This must be called with a context for the source side of a drag) Stock: the ID of the stock icon to use for the drag. Hot_x: the X offset within the icon of the hotspot. Hot_y: the Y offset within the icon of the hotspot.

```
procedure Set_Icon_Name
  (Context            :      Drag_Context;
   Icon_Name          :      String;
   Hot_X              :      Gint;
   Hot_Y              :      Gint);
```

Sets the icon for a given drag from a named themed icon. See
the docs for Gtk_Icon_Theme for more details. Note that the size of the icon depends on the icon theme (the icon is loaded at the symbolic size GTK_ICON_SIZE_DND), thus Hot_X and Hot_Y have to be used with care.

101 Package Gtk.Drawing_Area

This widget provides an empty canvas on which the application can draw anything. Note that this widget is simply an empty space, and that you need to connect it to events to make it useful. For instance, you might want to do one of the following :

* Connect it to "expose_event": The handlers are called every time the widget needs to be redrawn. You can then draw anything you want on the canvas, after getting its associated window with a call to Gtk.Widget.Get_Window. Note that the event mask is automatically set up to accept expose_events.

* Connect it to "button_press_event" and "button_release_event" events, when you want it to react to user input. Note that you need to set up the event mask with a call to Gtk.Widget.Set_Events.

See also the Double_Buffer widget provided in the GtkAda examples for an advanced example that demonstrates how to use double buffering, to avoid flickering in your drawings.

101.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Drawing_Area   (Package Gtk.Drawing_Area)
```

101.2 Subprograms

```
procedure Gtk_New
  (Drawing_Area       : out   Gtk_Drawing_Area);
```

Create a new blank Drawing_Area.
Note that the background of the widget is uninitialized, and that you have to draw on it yourself.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Drawing_Area.

102 Package Gtk.Editable

This widget is an abstract widget designed to support the common functionalities of all widgets for editing text. It provides general services to manipulate an editable widget, a large number of action signals used for key bindings, and several signals that an application can connect to to modify the behavior of a widget.

102.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Editable       (Package Gtk.Editable)
```

102.2 Signals

- **"changed"**

  ```
  procedure Handler (Widget : access Gtk_Editable_Record'Class);
  ```

  Called when the contents of Widget has changed

- **"delete_text"**

  ```
  procedure Handler (Widget    : access Gtk_Editable_Record'Class;
  Start_Pos : in Gint;
  End_Pos   : in Gint);
  ```

  Emitted when some text is deleted by the user. As for the "insert-text" handler, it is possible to override the default behavior by connecting a handler to this signal, and then stopping the signal.

- **"insert_text"**

  ```
  procedure Handler (Widget   : access Gtk_Editable_Record'Class;
  Text     : in UTF8_String;
  Length   : in Gint;
  Position : in Gint_Access);
  ```

  Emitted when some text is inserted inside the widget by the user. The default handler inserts the text into the widget. By connecting a handler to this signal, and then by stopping the signal with Gtk.Handlers.Emit_Stop_By_Name, it is possible to modify the inserted text, or even prevent it from being inserted. Position.all should be modified by the callback, and indicates the new position of the cursor after the text has been inserted.

102.3 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Editable.

```
procedure Select_Region
  (Editable           : access Gtk_Editable_Record;
   Start              :      Gint;
   The_End            :      Gint := -1);
```

Select the region of text from Start to The_End.
The characters that are selected are those characters at positions from Start up to, but not including The_End. If The_End_Pos is negative, then the characters selected will be those characters from Start to the end of the text.

```
procedure Get_Selection_Bounds
  (Widget             : access Gtk_Editable_Record;
   Success            : out   Boolean;
   Start_Pos          : out   Guint;
   End_Pos            : out   Guint);
```

Return the position of the start and end of the current selection.
If success is false, Start_Pos and End_Pos are not modified.

```
procedure Insert_Text
  (Editable           : access Gtk_Editable_Record;
   New_Text           :      UTF8_String;
   Position           : in out Gint);
```

Insert the given string at the given position.
Position is set to the new cursor position. If Position is -1, the text is appended at the end.

```
procedure Delete_Text
  (Editable           : access Gtk_Editable_Record;
   Start_Pos          :      Gint := 0;
   End_Pos            :      Gint := -1);
```

Delete the characters from Start_Pos to End_Pos.
If End_Pos is negative, the characters are deleted from Start_Pos to the end of the text.

```
function Get_Chars
  (Editable           : access Gtk_Editable_Record;
   Start_Pos          :      Gint := 0;
   End_Pos            :      Gint := -1)
   return UTF8_String;
```

Get the text from Start_Pos to End_Pos.
If End_Pos is negative, the text from Start_Pos to the end is returned.

```
procedure Cut_Clipboard
  (Editable           : access Gtk_Editable_Record);
```

Copy the characters in the current selection to the clipboard.
The selection is then deleted.

```
procedure Copy_Clipboard
  (Editable           : access Gtk_Editable_Record);
```

Copy the characters in the current selection to the clipboard.

```
procedure Paste_Clipboard
  (Editable           : access Gtk_Editable_Record);
```

The contents of the clipboard is pasted into the given widget at
the current cursor position.

```
procedure Delete_Selection
  (Editable           : access Gtk_Editable_Record);
```

Disclaim and delete the current selection.

```
procedure Set_Position
  (Editable           : access Gtk_Editable_Record;
   Position           :      Gint);
function Get_Position
  (Editable           : access Gtk_Editable_Record)
   return Gint;
```

Change the position of the cursor in the entry.
The cursor is displayed before the character with the given index in the widget (the first character has index 0). The value must be less than or equal to the number of characters in

the widget. A value of -1 indicates that the position should be set after the last character in the entry. Note that this position is in characters, not in bytes.

```
procedure Set_Editable
  (Widget             : access Gtk_Editable_Record;
   Editable           :      Boolean := True);
function Get_Editable
  (Editable           : access Gtk_Editable_Record)
   return Boolean;
```

Set the editable status of the entry.
If Editable is False, the user can not modify the contents of the entry. This does not affect the user of the insertion functions above.

103 Package Gtk.Entry_Completion

This widget provides completion functionality for Gtk.Gentry.Gtk_Entry.

"Completion functionality" means that when the user modifies the text in the entry, GtkEntryCompletion checks which rows in the model match the current content of the entry, and displays a list of matches. By default, the matching is done by comparing the entry text case-insensitively against the text column of the model (see Set_Text_Column), but this can be overridden with a custom match function (see Set_Match_Func).

When the user selects a completion, the content of the entry is updated. By default, the content of the entry is replaced by the text column of the model, but this can be overridden by connecting to the ::match-selected signal and updating the entry in the signal handler. Note that you should return TRUE from the signal handler to suppress the default behaviour.

To add completion functionality to an entry, use Gtk.Entry.Set_Completion.

In addition to regular completion matches, which will be inserted into the entry when they are selected, GtkEntryCompletion also allows to display "actions" in the popup window. Their appearance is similar to menuitems, to differentiate them clearly from completion strings. When an action is selected, the ::action-activated signal is emitted.

103.1 Signals

- **"action_activated"**

  ```
  procedure Handler
  (Completion : access Gtk_Entry_Completion_Record'Class;
  Index      : Gint);
  ```

  Gets emitted when an action is activated.

- **"insert_prefix"**

  ```
  procedure Handler
  (Completion : access Gtk_Entry_Completion_Record'Class;
  Prefix     : String);
  ```

  Gets emitted when the inline autocompletion is triggered. The default behaviour is to make the entry display the whole prefix and select the newly inserted part. Applications may connect to this signal in order to insert only smaller part of the Prefix into the entry - e.g. the entry used in the #GtkFileChooser inserts only the part of the prefix up to the next '/'. Return value: %TRUE if the signal has been handled

- **"match_selected"**

  ```
  procedure Handler
  (Completion : access Gtk_Entry_Completion_Record'Class;
  Model      : Gtk_Tree_Model;
  Iter       : Gtk_Tree_Iter);
  ```

  Gets emitted when a match from the list is selected. The default behaviour is to replace the contents of the entry with the contents of the text column in the row pointed to by Iter. Return value: %TRUE if the signal has been handled

103.2 Types

type Data_Type **is private**;

type Destroy_Notify **is access procedure**
(Data : **in out** Data_Type);

type Gtk_Entry_Completion_Match_Func **is access function**
(Completion : **access** Gtk_Entry_Completion_Record'Class;
Key : String;
Iter : Gtk.Tree_Model.Gtk_Tree_Iter;
User_Data : Data_Type) **return** Boolean;

103.3 Subprograms

procedure Gtk_New
(Completion : **out** Gtk_Entry_Completion);

function Get_Type **return** Glib.GType;

Return the internal type used for this object

procedure Complete
(Completion : **access** Gtk_Entry_Completion_Record);

Requests a completion operation, or in other words a refiltering of the current list with completions, using the current key. The completion list view will be updated accordingly.

procedure Delete_Action
(Completion : **access** Gtk_Entry_Completion_Record;
Index : Gint);

Deletes the action at index from completion's action list.

function Get_Entry
(Completion : **access** Gtk_Entry_Completion_Record)
return Gtk.Widget.Gtk_Widget;

Gets the entry completion has been attached to.

procedure Set_Inline_Completion
(Completion : **access** Gtk_Entry_Completion_Record;
Inline_Completion : Boolean);

function Get_Inline_Completion
(Completion : **access** Gtk_Entry_Completion_Record)
return Boolean;

Returns whether the common prefix of the possible completions should be automatically inserted in the entry. This text appears greyed out, and is removed when the user types some text not compatible with the possible completions

procedure Set_Minimum_Key_Length
(Completion : **access** Gtk_Entry_Completion_Record;
Length : Gint);

```
function Get_Minimum_Key_Length
  (Completion         : access Gtk_Entry_Completion_Record)
   return Gint;
```

Requires the length of the search key for completion to be at least
length. This is useful for long lists, where completing using a small key takes a lot of time and will come up with meaningless results anyway (ie, a too large dataset). This is the minimal number of characters the user must start typing before any completion is attempted

```
procedure Set_Model
  (Completion         : access Gtk_Entry_Completion_Record;
   Model              :      Gtk.Tree_Model.Gtk_Tree_Model);
function Get_Model
  (Completion         : access Gtk_Entry_Completion_Record)
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Returns the model the completion is using as data source.
Returns null if the model is unset (setting it to null unsets the current model)

```
procedure Set_Popup_Completion
  (Completion         : access Gtk_Entry_Completion_Record;
   Popup_Completion   :      Boolean);
function Get_Popup_Completion
  (Completion         : access Gtk_Entry_Completion_Record)
   return Boolean;
```

Returns whether the completions should be presented in a popup window.
This is to be used in addition to, or instead of, Get_Inline_Completion.

```
procedure Set_Popup_Set_Width
  (Completion         : access Gtk_Entry_Completion_Record;
   Popup_Set_Width    :      Boolean);
function Get_Popup_Set_Width
  (Completion         : access Gtk_Entry_Completion_Record)
   return Boolean;
```

Returns whether the completion popup window will be resized to the
width of the entry.

```
procedure Set_Popup_Single_Match
  (Completion         : access Gtk_Entry_Completion_Record;
   Popup_Single_Match  :      Boolean);
function Get_Popup_Single_Match
  (Completion         : access Gtk_Entry_Completion_Record)
   return Boolean;
```

Returns whether the completion popup window will appear even if there is
only a single match. You may want to set this to False if you are using inline completion.

```
procedure Set_Text_Column
  (Completion         : access Gtk_Entry_Completion_Record;
   Column             :      Gint);
function Get_Text_Column
  (Completion         : access Gtk_Entry_Completion_Record)
   return Gint;
```

Convenience function for setting up the most used case of this code: a
completion list with just strings. This function will set up completion to have a list displaying all (and just) strings in the completion list, and to get those strings from column in the model of completion.

This functions creates and adds a #GtkCellRendererText for the selected column. If you need to set the text column, but don't want the cell renderer, use Set_Property to set the ::text_column property directly.

```
procedure Insert_Action_Markup
  (Completion         : access Gtk_Entry_Completion_Record;
   Index              :      Gint;
   Markup             :      String);
```

Inserts an action in ccompletion's action item list at position index with the given markup. Markup can be used to represent bold text, for instance with "bold text"

```
procedure Insert_Action_Text
  (Completion         : access Gtk_Entry_Completion_Record;
   Index              :      Gint;
   Text               :      String);
```

Inserts an action in completion's action item list at position index with text Text. If you want the action item to have markup, use Insert_Action_Markup.

```
procedure Insert_Prefix
  (Completion         : access Gtk_Entry_Completion_Record);
```

Requests a prefix insertion.

```
procedure Set_Match_Func
  (Completion         : access Gtk_Entry_Completion_Record;
   Func               :      Gtk_Entry_Completion_Match_Func;
   Func_Data          :      Data_Type;
   Func_Notify        :      Destroy_Notify);
```

Sets the match function for completion to be Func. The match function is used to determine if a row should or should not be in the completion list.

104 Package Gtk.Enums

This package contains a number of types that are shared by several widgets in GtkAda.

104.1 Types

```
type Gtk_Anchor_Type is
          (Anchor_Center,
           Anchor_North,
           Anchor_North_West,
           Anchor_North_East,
           Anchor_South,
           Anchor_South_East,
           Anchor_South_West,
           Anchor_West,
           Anchor_East);
```

Gtk_Anchor_Type indicates the exact location of the widget on its side. Note that not all anchors are relevant for each side. For instance, if you put a widget on Side_Right, with an anchor of Anchor_North, Anchor_North_West or Anchor_North_East, the widget will in fact appear on the upper right side of the remaining space in the container. Thus, if a previous child was added on Side_North, then the new child will only appear on the second line in the container. The order the children are inserted into the container is important.

```
type Gtk_Arrow_Type is
          (Arrow_Up,
           Arrow_Down,
           Arrow_Left,
           Arrow_Right);
```

The various types of arrows that can be represented by GtkAda

```
type Gtk_Attach_Options is new Glib.Guint32;
```

The various options used for attaching widgets to tables

```
type Gtk_Button_Box_Style is
          (Buttonbox_Default_Style,
           Buttonbox_Spread,
           Buttonbox_Edge,
           Buttonbox_Start,
           Buttonbox_End);
```

The style for button boxes (see gtk-button_box.ads)

```
type Gtk_Corner_Type is
          (Corner_Top_Left,
           Corner_Bottom_Left,
           Corner_Top_Right,
           Corner_Bottom_Right);
```

Type used by Set_Placement below to determine the location of the child widget with respect to the scrollbars. Corner_Top_Left means the child is in the top left, with the scrollbars

underneath and to the right.

```
type Gtk_Curve_Type is
          (Curve_Type_Linear,       --  Linear interpolation
           Curve_Type_Spline, -- Spline interpolation
           Curve_Type_Free); -- Free form curve
```

The curve types that can be used in gtk-curve.ads

```
type Gtk_Delete_Type is
          (Delete_Chars,
           Delete_Word_Ends,
           Delete_Words,
           Delete_Display_Lines,
           Delete_Display_Line_Ends,
           Delete_Paragraph_Ends,
           Delete_Paragraphs,
           Delete_Whitespace);
```

The deletion modes used in the text editor. Delete_Word_Ends will delete only the portion of the word to the left/right of the cursor if we are in the middle of a word. Delete_Paragraph_Ends acts like c-k in Emacs: it deletes the text until, but not including, the end of line. Delete_Paragraphs acts like c-k in pico: it deletes the whole line. Delete_Whitespace acts like M-\ in Emacs, and removes all white spaces surrounding the cursor.

```
type Gtk_Direction_Type is
          (Dir_Tab_Forward,
           Dir_Tab_Backward,
           Dir_Up,
           Dir_Down,
           Dir_Left,
           Dir_Right);
```

Focus movement types

```
type Gtk_Expander_Style is
          (Expander_Collapsed,
           Expander_Semi_Collapsed,
           Expander_Semi_Expanded,
           Expander_Expanded);
```

Expander styles, as seen in trees

```
type Gtk_Icon_Size is new Gint;
```

```
type Gtk_Justification is
          (Justify_Left,
           Justify_Right,
           Justify_Center,
```

```
          Justify_Fill);
```

Within a paragraph, text can be justified in various ways: aligned on the left, aligned on the right, centered, or justified (in which case the width of the spaces might vary so that the text is aligned on both sides).

```
type Gtk_Menu_Direction_Type is
          (Menu_Dir_Parent,
           Menu_Dir_Child,
           Menu_Dir_Next,
           Menu_Dir_Prev);
```

Direction where to move the selection.

```
type Gtk_Metric_Type is
          (Pixels, Inches, Centimeters);
```

The unit to use when you display a ruler at the top of a drawing area.

```
type Gtk_Movement_Step is
          (Movement_Logical_Positions, --  move by forw/back graphemes
           Movement_Visual_Positions, -- move by left/right graphemes
           Movement_Words, -- move by forward/back words
           Movement_Display_Lines, -- move up/down lines (wrapped lines)
```

```
type Gtk_Orientation is
          (Orientation_Horizontal, Orientation_Vertical);
```

Orientation of widgets. Most widgets have no such notion, but for instance toolbars can display different kind of information depending on their current orientation

```
type Gtk_Pack_Direction is
          (Pack_Direction_LTR,
           Pack_Direction_RTL,
           Pack_Direction_TTB,
           Pack_Direction_BTT);
```

The direction in which children should be packed in their parents (Left-to-Right, Right-To-Left, Top-To-Bottom or Bottom-To-Top)

```
type Gtk_Pack_Type is
          (Pack_Start, Pack_End);
```

Whether items should be added at the start or at the end of the list of children for a widget. This impacts the visual rendering of containers

```
type Gtk_Path_Priority_Type is mod 2 ** 32;
```

Priorities for path lookups

```
type Gtk_Path_Type is
          (Path_Widget, Path_Widget_Class, Path_Class);
```

Widget path types

```
type Gtk_Policy_Type is
          (Policy_Always, Policy_Automatic, Policy_Never);
```

When should scrollbars be made visible in Gtk_Scrolled_Window

```
type Gtk_Position_Type is
          (Pos_Left,
           Pos_Right,
           Pos_Top,
           Pos_Bottom);
```

Use to define the position of children within a container

```
type Gtk_Relief_Style is
          (Relief_Normal, Relief_Half, Relief_None);
```

Explains how the border of widgets should be displayed

```
type Gtk_Resize_Mode is
          (Resize_Parent,     --  Pass request to the parent
           Resize_Queue, -- Queue resizes on this widget
           Resize_Immediate); -- Perform the resizes now
```

The resizing of widgets is generally done asynchronously, for efficiency reasons. This can have some impact on the visual rendering of the widget which might be an issue in some cases. This type is only used when you are writting your own containers.

```
type Gtk_Scroll_Step is
          (Scroll_Steps,
           Scroll_Pages,
           Scroll_Ends,
           Scroll_Horizontal_Steps,
           Scroll_Horizontal_Pages,
           Scroll_Horizontal_Ends);
```

The behavior of scrollbars for editors

```
type Gtk_Scroll_Type is
          (Scroll_None,
           Scroll_Jump,
           Scroll_Step_Backward,
           Scroll_Step_Forward,
           Scroll_Page_Backward,
           Scroll_Page_Forward,
```

```
           Scroll_Step_Up,
           Scroll_Step_Down,
           Scroll_Page_Up,
           Scroll_Page_Down,
           Scroll_Step_Left,
           Scroll_Step_Right,
           Scroll_Page_Left,
           Scroll_Page_Right,
           Scroll_Start,
           Scroll_End);
```

How clists should be scrolled

```
type Gtk_Selection_Mode is
          (Selection_None,
           Selection_Single,
           Selection_Browse,
           Selection_Multiple);
```

Indicates what selection is allowed in a tree (no selection allowed, a single line, a single line when the mouse is released, or multiple lines).

```
type Gtk_Shadow_Type is
          (Shadow_None,
           Shadow_In,
           Shadow_Out,
           Shadow_Etched_In,
           Shadow_Etched_Out);
```

The type of shadows that can be drawn around widgets

```
type Gtk_Sort_Type is
          (Sort_Ascending,
           Sort_Descending);
```

How to sort

```
type Gtk_State_Type is
          (State_Normal,
           State_Active,
           State_Prelight,
           State_Selected,
           State_Insensitive);
```

Widgets can be in various states. This impacts their visual rendering, but can also impact whether they react to events or not (they do not when in State_Insensitive mode).

```
type Gtk_Text_Direction is
          (Text_Dir_None,
           Text_Dir_Ltr,
           Text_Dir_Rtl);
```

The directory in which text should be written (left to right or right to left).

```
type Gtk_Text_Window_Type is
           (Text_Window_Private,
            Text_Window_Widget,
            Text_Window_Text,
            Text_Window_Left,
            Text_Window_Right,
            Text_Window_Top,
            Text_Window_Bottom);
```

The various components of a Gtk.Text_View widget

```
type Gtk_Toolbar_Style is
           (Toolbar_Icons,
            Toolbar_Text,
            Toolbar_Both,
            Toolbar_Both_Horiz);
```

The style of toolbars. Toolbar_Both_Horiz indicates that both icon and text should be displayed, arranged horizontally.

```
type Gtk_Update_Type is
           (Update_Continuous,
            Update_Discontinuous,
            Update_Delayed);
```

For some widgets, this indicates how often they should be updated

```
type Gtk_Visibility is
           (Visibility_None,
            Visibility_Partial,
            Visibility_Full);
```

Generic visibility flags. This indicate how visible a window currently is.

```
type Gtk_Window_Position is
           (Win_Pos_None,
            Win_Pos_Center,
            Win_Pos_Mouse,
            Win_Pos_Center_Always,
            Win_Pos_Center_On_Parent);
```

The position at which a new window should be initially displayed on the screen.

```
type Gtk_Window_Type is
           (Window_Toplevel,
            Window_Popup);
```

GtkAda supports multiple types of windows. They all act as top-level containers, but the amount of decoration is different. A popup window has no title bar for instance.

```
type Gtk_Wrap_Mode is
           (Wrap_None,
            Wrap_Char,
            Wrap_Word,
            Wrap_Word_Char);
```

Text wrapping algorithm. This indicates where a text widget is allowed to break its contents to display multiple lines when a line doesn't fit on the screen.

type Property_Gtk_Arrow_Type **is new** Arrow_Type_Properties.Property;

type Property_Gtk_Button_Box_Style **is new** Button_Box_Style_Properties.Property;

type Property_Gtk_Justification **is new** Justification_Properties.Property;

type Property_Gtk_Orientation **is new** Orientation_Properties.Property;

type Property_Gtk_Policy_Type **is new** Policy_Properties.Property;

type Property_Gtk_Position_Type **is new** Position_Type_Properties.Property;

type Property_Gtk_Relief_Style **is new** Relief_Style_Properties.Property;

type Property_Gtk_Resize_Mode **is new** Resize_Mode_Properties.Property;

type Property_Gtk_Shadow_Type **is new** Shadow_Type_Properties.Property;

type Property_Gtk_Text_Direction **is new** Text_Direction_Properties.Property;

type Property_Gtk_Toolbar_Style **is new** Toolbar_Style_Properties.Property;

type Property_Gtk_Update_Type **is new** Update_Type_Properties.Property;

```
type Property_Gtk_Window_Position is new Window_Position_Properties.Property;

type Property_Gtk_Window_Type is new Window_Type_Properties.Property;

type Property_Gtk_Wrap_Mode is new Wrap_Mode_Properties.Property;

type Property_Metric_Type is new Metric_Type_Properties.Property;

type Property_Pack_Direction is new Pack_Direction_Properties.Property;

type Property_Pack_Type is new Pack_Type_Properties.Property;

type Property_Sort_Type is new Sort_Type_Properties.Property;
```

104.2 Subprograms

104.2.1 Some Glib instantiations

```
function Convert
  (S                  :      String)
   return System.Address;
function Convert
  (S                  :      System.Address)
   return String;
function Convert_I
  (I                  :      Gint)
   return System.Address;
function Convert_A
  (S                  :      System.Address)
   return Gint;
procedure Free_String_List
  (List               : in out String_List.Glist);
procedure Free_String_List
  (List               : in out String_SList.GSlist);
```

Free the memory occupied by all the strings in the list, as well as the memory occupied by the list itself.

```
function Convert_UI
  (I                  :      Guint)
   return System.Address;
```

```
function Convert_UA
  (S                  :      System.Address)
   return Guint;
```

105 Package Gtk.Event_Box

This widget is a container that catches events for its child when its child does not have its own window (like a Gtk_Scrolled_Window or a Gtk_Label for instance). Some widgets in GtkAda do not have their own window, and thus can not directly get events from the server. The Gtk_Event_Box widget can be used to force its child to receive events anyway.

For instance, this widget is used internally in a Gtk_Combo_Box so that the application can change the cursor when the mouse is in the popup window. In that case, it contains a frame, that itself contains the scrolled window of the popup.

105.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Event_Box (Package Gtk.Event_Box)
```

105.2 Subprograms

```
procedure Gtk_New
  (Event_Box          : out   Gtk_Event_Box);
```

Create a new box.
The box's child can then be set using the Gtk.Container.Add function.

```
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Event_Box.

```
procedure Set_Visible_Window
  (Event_Box          : access Gtk_Event_Box_Record;
   Visible_Window     :      Boolean);
function Get_Visible_Window
  (Event_Box          : access Gtk_Event_Box_Record)
   return Boolean;
```

Set whether the event box uses a visible or invisible child window. The
default is to use visible windows Except if you want to explicitly change the background, or explicitly draw on it, you should make the event box invisible.

```
procedure Set_Above_Child
  (Event_Box          : access Gtk_Event_Box_Record;
   Above_Child        :      Boolean);
function Get_Above_Child
  (Event_Box          : access Gtk_Event_Box_Record)
   return Boolean;
```

Set whether the event box window is positioned above the windows of its
child, as opposed to below it. If the window is above, all events inside the event box will go to the event box. If the window is below, events in windows of child widgets will first go to that widget, and then to its parent. The default is to keep the window below the child.

106 Package Gtk.Expander

A container which can hide its child.

106.1 Signals

- **"activate"**

  ```
  procedure Handler (Expander : access Gtk_Expander_Record'Class);
  ```

  Send this signal if you want to toggle the state of the expander, as if the user had clicked on it. This is mostly useful when associated with a keybinding

- **"notify::expanded"**

106.2 Subprograms

```
procedure Gtk_New
  (Expander           : out   Gtk_Expander;
   Label              :       String);
procedure Gtk_New_With_Mnemonic
  (Expander           : out   Gtk_Expander;
   Label              :       String);
function Get_Type               return Glib.GType;
```

Returns the internal value used for an expander

```
procedure Set_Expanded
  (Expander           : access Gtk_Expander_Record;
   Expanded           :       Boolean);
function Get_Expanded
  (Expander           : access Gtk_Expander_Record)
   return Boolean;
```

Sets the state of the expander. Set to True, if you want
the child widget to be revealed, and False if you want the child widget to be hidden.

```
procedure Set_Label
  (Expander           : access Gtk_Expander_Record;
   Label              :       String);
function Get_Label
  (Expander           : access Gtk_Expander_Record)
   return String;
```

Sets the text of the label of the expander to Label.

```
procedure Set_Label_Widget
  (Expander           : access Gtk_Expander_Record;
   Label_Widget       : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Label_Widget
  (Expander           : access Gtk_Expander_Record)
   return Gtk.Widget.Gtk_Widget;
```

Set the label widget for the expander. This is the widget
that will appear embedded alongside the expander arrow.

```
procedure Set_Spacing
  (Expander           : access Gtk_Expander_Record;
   Spacing            :       Gint);
```

```
function Get_Spacing
  (Expander           : access Gtk_Expander_Record)
   return Gint;
```

Sets the spacing field of Expander, which is the number of pixels to place between expander and the child.

```
procedure Set_Use_Markup
  (Expander           : access Gtk_Expander_Record;
   Use_Markup         :      Boolean);
function Get_Use_Markup
  (Expander           : access Gtk_Expander_Record)
   return Boolean;
```

Sets whether the text of the label contains markup in Pango's text markup language. See Gtk.Label.Set_Markup.

```
procedure Set_Use_Underline
  (Expander           : access Gtk_Expander_Record;
   Use_Underline      :      Boolean);
function Get_Use_Underline
  (Expander           : access Gtk_Expander_Record)
   return Boolean;
```

If true, an underline in the text of the expander label indicates the next character should be used for the mnemonic accelerator key.

107 Package Gtk.File_Chooser

This package provides an interface implemented by Gtk.File_Chooser_Widget and Gtk.File_Chooser_Button, and by your own file selection widgets if you wish to expose a standard interface.

Gtk_File_Chooser allows for shortcuts to various places in the filesystem. In the default implementation these are displayed in the left pane. It may be a bit confusing at first that these shortcuts come from various sources and in various flavours, so lets explain the terminology here: @itemize @bullet @item Bookmarks are created by the user, by dragging folders from the right pane to the left pane, or by using the "Add". Bookmarks can be renamed and deleted by the user. @item Shortcuts can be provided by the application or by the underlying filesystem abstraction (e.g. both the gnome-vfs and the Windows filesystems provide "Desktop" shortcuts). Shortcuts cannot be modified by the user. @item Volumes are provided by the underlying filesystem abstraction. They are the "roots" of the filesystem.

@end itemize File Names and Encodings

When the user is finished selecting files in a Gtk_File_Chooser, your program can get the selected names either as filenames or as URIs. For URIs, the normal escaping rules are applied if the URI contains non-ASCII characters. However, filenames are always returned in the character set specified by the G_FILENAME_ENCODING environment variable. Please see the Glib documentation for more details about this variable.

Important: This means that while you can pass the result of Get_Filename to low-level file system primitives , you will need to convert it to UTF-8 before using it in a Gtk_Label for instance. Conversion is done through Glib.Convert.Filename_To_UTF8.

You can add a custom preview widget to a file chooser and then get notification about when the preview needs to be updated. To install a preview widget, use gtk_file_chooser_set_preview_widget(). Then, connect to the GtkFileChooser::update-preview signal to get notified when you need to update the contents of the preview.

Preview widgets

You can add a custom preview widget to a file chooser and then get notification about when the preview needs to be updated. To install a preview widget, use Set_Preview_Widget. Then, connect to the GtkFileChooser::update-preview signal to get notified when you need to update the contents of the preview.

Your callback should use Get_Preview_Filename to see what needs previewing. Once you have generated the preview for the corresponding file, you must call Set_Preview_Widget_Active with a boolean flag that indicates whether your callback could successfully generate a preview.

Adding Extra Widgets

You can add extra widgets to a file chooser to provide options that are not present in the default design. For example, you can add a toggle button to give the user the option to open a file in read-only mode. You can use Set_Extra_Widget to insert additional widgets in a file chooser.

If you want to set more than one extra widget in the file chooser, you can a container such as a GtkVBox or a GtkTable and include your widgets in it. Then, set the container as the whole extra widget.

Key bindings

The following default key bindings are provided, but you can use a gtkrc file to override them if you need (see gtk-rc.ads). Signal name | Key binding location-popup | Control-L (empty path) | / (path of "/") | ~ (home directory) up-folder | Alt-Up or backspace down-folder | Alt-Down home-folder | Alt-Home desktop-folder | Alt-D quick-bookmark | Alt-1 through Alt-0

107.1 Signals

- **"confirm-overwrite"**

  ```
  function Handler
  (Chooser : Gtk_File_Chooser) return File_Chooser_Confirmation;
  ```

  This signal gets emitted whenever it is appropriate to present a confirmation dialog when the user has selected a file name that already exists. The signal only gets emitted when the file chooser is in Action_Save mode. Most applications just need to turn on the do-overwrite-confirmation property (or call the Set_Do_Overwrite_Confirmation function), and they will automatically get a stock confirmation dialog. Applications which need to customize this behavior should do that, and also connect to the confirm-overwrite signal. A signal handler for this signal must return a value which indicates the action to take. If the handler determines that the user wants to select a different filename, it should return Confirmation_Select_Again. If it determines that the user is satisfied with his choice of file name, it should return Confirmation_Accept_Filename. On the other hand, if it determines that the stock confirmation dialog should be used, it should return Confirmation_Confir.

- **"current-folder-changed"**

  ```
  procedure Handler (Chooser : Gtk_File_Chooser);
  ```

  This signal is emitted when the current folder in a Gtk_File_Chooser changes. This can happen due to the user performing some action that changes folders, such as selecting a bookmark or visiting a folder on the file list. It can also happen as a result of calling a function to explicitly change the current folder in a file chooser. Normally you do not need to connect to this signal, unless you need to keep track of which folder a file chooser is showing.

- **"file-activated"**

  ```
  procedure Handler (Chooser : Gtk_File_Chooser);
  ```

  This signal is emitted when the user "activates" a file in the file chooser. This can happen by double-clicking on a file in the file list, or by pressing Enter. Normally you do not need to connect to this signal. It is used internally by Gtk_File_Chooser_Dialog to know when to activate the default button in the dialog.

- **"selection-changed"**

  ```
  procedure Handler (Chooser : Gtk_File_Chooser);
  ```

  This signal is emitted when there is a change in the set of selected files in a Gtk_File_Chooser. This can happen when the user modifies the selection with the mouse or the keyboard, or when explicitly calling functions to change the selection. Normally you do not need to connect to this signal, as it is easier to wait for the file chooser to finish running, and then to get the list of selected files using the functions mentioned below.

- **"update-preview"**

  ```
  procedure Handler (Chooser : Gtk_File_Chooser);
  ```

  This signal is emitted when the preview in a file chooser should be regenerated. For example, this can happen when the currently selected file changes. You should use this signal if you want your file chooser to have a preview widget. Once you have installed a preview widget with Set_Preview_Widget, you should update it when this signal is emitted. You can use the functions Get_Preview_Filename or Get_Preview_Uri to get the name of the file to preview. Your widget may not be able to preview all kinds of files; your callback must call Set_Preview_Wiget_Active to inform the file chooser about whether the preview was generated successfully or not.

107.2 Types

```
type File_Chooser_Action is
          (Action_Open,
           Action_Save,
           Action_Select_Folder,
           Action_Create_Folder);
```

Describes whether a Gtk_File_Chooser is being used to open existing files or to save to a possibly new file. Action_Open: Will only let the user select existing file Action_Save: Select existing file or enter new filename Action_Select_Folder: Only pick an existing folder Action_Create_Folder: Select existing folder or enter new name

```
type File_Chooser_Confirmation is
          (Confirmation_Confirm,
           Confirmation_Accept_Filename,
           Confirmation_Select_Again);
```

Used as a return value of handlers for the confirm-overwrite signal of a Gtk_File_Chooser. This value determines whether the file chooser will present the stock confirmation dialog, accept the user's choice of a filename, or let the user choose another filename. Confirmation_Confirm: Ask confirmation about overwriting existing file Confirmation_Accept_Filename: Accept the user's choice Confirmation_Select_Again: Let the user select another file

```
type File_Chooser_Error is
          (Error_Non_Existent,
           Error_Bad_Filename);
```

Identify the various errors that can occur while calling Gtk_File_Chooser functions

```
type Gtk_File_Chooser is new Glib.Types.GType_Interface;
```

107.3 Subprograms

```
function Get_Type                  return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_File_Chooser

107.3.1 Configuration

```
procedure Set_Action
  (Chooser            :      Gtk_File_Chooser;
   Action             :      File_Chooser_Action);

function Get_Action
  (Chooser            :      Gtk_File_Chooser)
   return File_Chooser_Action;
```

Sets the type of operation that the chooser is performing; the
user interface is adapted to suit the selected action. For example, an option to create a new folder might be shown if the action is Action_Save, but not if the action is Action_Open.

```
procedure Set_Local_Only
  (Chooser            :      Gtk_File_Chooser;
   Local_Only         :      Boolean := True);

function Get_Local_Only
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether only local files can be selected in the file selector. If
Local_Only is True (the default), then the selected file are files are guaranteed to be accessible through the operating systems native file file system and therefore the application only needs to worry about the filename functions in Gtk_File_Chooser, like Get_Filename, rather than the URI functions like Get_Uri,

```
procedure Set_Select_Multiple
  (Chooser            :      Gtk_File_Chooser;
   Select_Multiple    :      Boolean);

function Get_Select_Multiple
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether multiple files can be selected in the file selector. This
is only relevant if the action is set to be Action_Open or Action_Save. It cannot be set with either of the folder actions.

```
procedure Set_Show_Hidden
  (Chooser            :      Gtk_File_Chooser;
   Show_Hidden        :      Boolean);

function Get_Show_Hidden
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether hidden files and folders are displayed in the file selector

```
procedure Set_Do_Overwrite_Confirmation
  (Chooser            :      Gtk_File_Chooser;
   Do_Overwrite_Confirmation  :     Boolean);

function Get_Do_Overwrite_Confirmation
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether a file chooser in Action_Save mode will present a
confirmation dialog if the user types a file name that already exists. This is False by default. Regardless of this setting, Chooser will emit the "confirm-overwrite" signal when appropriate. If all you need is the stock confirmation dialog, set this property to True You

can override the way confirmation is done by actually handling the "confirm-overwrite" signal; please refer to its documentation for the details.

```
procedure Set_Current_Name
  (Chooser            :      Gtk_File_Chooser;
   Name               :      UTF8_String);
```

Sets the current name in the file selector, as if entered by the user.
Note that the name passed in here is a UTF-8 string rather than a filename. This function is meant for such uses as a suggested name in a "Save As..." dialog. If you want to preselect a particular existing file, you should use Set_Filename or Set_Uri instead.

107.3.2 Filename manipulation

```
function Get_Filename
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the filename for the currently selected file in the file selector.
If multiple files are selected, one of the filenames will be returned at random. If the file chooser is in folder mode, this function returns the selected folder. Return value: The currently selected filename, or "" if no file is selected, or the selected file can't be represented with a local filename.

```
function Get_Filenames
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Enums.String_SList.GSlist;
```

Lists all the selected files and subfolders in the current folder of
Chooser. The returned names are full absolute paths. If files in the current folder cannot be represented as local filenames they will be ignored. (See Get_Uris). The returned value must be freed by the caller, through Gtk.Enums.Free_String_List.

```
function Set_Filename
  (Chooser            :      Gtk_File_Chooser;
   Filename           :      String)
   return Boolean;
```

Sets Filename as the current filename for the file chooser, by changing
to the file's parent folder and actually selecting the file in list. If the Chooser is in Action_Save mode, the file's base name will also appear in the dialog's file name entry. If the file name isn't in the current folder of Chooser, then the current folder of Chooser will be changed to the folder containing Filename. This is equivalent to a sequence of Unselect_All followed by Select_Filename. Note that the file must exist, or nothing will be done except for the directory change. If you are implementing a "File/Save As..." dialog, you should use this function if you already have a file name to which the user may save; for example, when the user opens an existing file and then does "File/Save As..." on it. If you don't have a file name already — for example, if the user just created a new file and is saving it for the first time, do not call this function. Instead, use Set_Current_Name.

Returns True if both the folder could be changed and the file was selected successfully.

```
function Select_Filename
  (Chooser            :      Gtk_File_Chooser;
   Filename           :      String)
   return Boolean;
procedure Unselect_Filename
  (Chooser            :      Gtk_File_Chooser;
```

```
   Filename           :      String);
```

Selects a filename. If the file name isn't in the current
folder of Chooser, then the current folder of Chooser will be changed to the folder containing Filename. Returns True if both the folder could be changed and the file was selected successfully.

```
procedure Select_All
  (Chooser            :      Gtk_File_Chooser);
procedure Unselect_All
  (Chooser            :      Gtk_File_Chooser);
```

Selects all the files in the current folder of a file chooser.

```
function Set_Current_Folder
  (Chooser            :      Gtk_File_Chooser;
   Filename           :      String)
   return Boolean;
```

Sets the current folder for Chooser from a local filename. The user will
be shown the full contents of the current folder, plus user interface elements for navigating to other folders. Filename is an absolute file name. Returns True if the folder could be changed successfully.

```
function Get_Current_Folder
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the current folder of Chooser as a local filename.
Returns "" if the file chooser was unable to load the last folder that was requested from it, for instance with a call to Set_Current_Folder with an invalid directory. Also returns "" if the selected path cannot be represented as a local filename.

107.3.3 URI manipulation

```
function Set_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String)
   return Boolean;
```

Sets the file referred to by Uri as the current file for the file
chooser, by changing to the URI's parent folder and actually selecting the URI in the list. If the Chooser is Action_Save mode, the URI's base name will also appear in the dialog's file name entry. If the URI isn't in the current folder of Chooser, then the current folder of Chooser will be changed to the folder containing Uri. This is equivalent to a sequence of Unselect_All followed by Select_Uri. Note that the URI must exist, or nothing will be done except for the directory change. See also Set_Filename Return True if both the folder could be changed and the URI was selected successfully.

```
function Get_Uri
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the URI for the currently selected file in the file selector. If
multiple files are selected, one of the filenames will be returned at random. If the file chooser is in folder mode, this function returns the selected folder.

```
function Get_Uris
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Enums.String_SList.GSlist;
```

Lists all the selected files and subfolders in the current folder of
Chooser. The returned names are full absolute URIs. The returned value must be freed by
the caller

```
function Select_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String)
   return Boolean;

procedure Unselect_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String);
```

Selects the file by Uri. If the URI doesn't refer to a file in the
current folder of Chooser, then the current folder of Chooser will be changed to the folder
containing Uri. Return True if both the folder could be changed and the URI was selected
successfully.

```
function Set_Current_Folder_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String)
   return Boolean;
```

Sets the current folder for Chooser from an URI. The user will be shown
the full contents of the current folder, plus user interface elements for navigating to other
folders. Return True if the folder could be changed successfully.

```
function Get_Current_Folder_Uri
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the current folder of Chooser as an URI, or "" if the chooser was
unable to load the last folder set by Set_Current_Folder_Uri.

107.3.4 Preview widget

```
procedure Set_Preview_Widget
  (Chooser            :      Gtk_File_Chooser;
   Preview_Widget     : access Gtk.Widget.Gtk_Widget_Record'Class);

function Get_Preview_Widget
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Widget.Gtk_Widget;
```

Sets an application-supplied widget to use to display a custom preview
of the currently selected file. To implement a preview, after setting the preview widget, you connect to the "update-preview" signal, and call Get_Preview_Filename() or Get_Preview_Uri on each change. If you can display a preview of the new file, update your widget and set the preview active using Set_Preview_Widget_Active. Otherwise, set the preview inactive. When there is no application-supplied preview widget, or the application-supplied preview widget is not active, the file chooser may display an internally generated preview of the current file or it may display no preview at all.

```
procedure Set_Preview_Widget_Active
  (Chooser            :      Gtk_File_Chooser;
   Active             :      Boolean);

function Get_Preview_Widget_Active
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether the preview widget set by Set_Preview_Widget should be shown for the current filename. When Active is set to false, the file chooser may display an internally generated preview of the current file or it may display no preview at all.

```
procedure Set_Use_Preview_Label
  (Chooser            :      Gtk_File_Chooser;
   Use_Label          :      Boolean);

function Get_Use_Preview_Label
  (Chooser            :      Gtk_File_Chooser)
   return Boolean;
```

Sets whether the file chooser should display a stock label with the name of the file that is being previewed; the default is True. Applications that want to draw the whole preview area themselves should set this to False and display the name themselves in their preview widget.

```
function Get_Preview_Filename
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the filename that should be previewed in a custom preview widget. See Set_Preview_Widget. Return "" if no file is selected or it can't be represented as a local filename.

```
function Get_Preview_Uri
  (Chooser            :      Gtk_File_Chooser)
   return String;
```

Gets the URI that should be previewed in a custom preview widget. See Set_Preview_Widget. Return "" if no file is selected.

107.3.5 Extra widget

```
procedure Set_Extra_Widget
  (Chooser            :      Gtk_File_Chooser;
   Extra_Widget       : access Gtk.Widget.Gtk_Widget_Record'Class);

function Get_Extra_Widget
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Widget.Gtk_Widget;
```

Sets an application-supplied widget to provide extra options to the user

107.3.6 Filters

```
procedure Add_Filter
  (Chooser            :      Gtk_File_Chooser;
   Filter             : access Gtk.File_Filter.Gtk_File_Filter_Record'Class);

procedure Remove_Filter
  (Chooser            :      Gtk_File_Chooser;
   Filter             : access Gtk.File_Filter.Gtk_File_Filter_Record'Class);
```

Adds or Removes Filter to the list of filters that the user can select between. When a filter is selected, only files that are passed by that filter are displayed. Note that the Chooser takes ownership of the filter, so you have to ref and sink it if you want to keep a reference.

```
procedure Set_Filter
  (Chooser            :      Gtk_File_Chooser;
   Filter             : access Gtk.File_Filter.Gtk_File_Filter_Record'Class);
```

```
function Get_Filter
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.File_Filter.Gtk_File_Filter;
```

Sets the current filter; only the files that pass the filter will be displayed. If the user-selectable list of filters is non-empty, then the filter should be one of the filters in that list. Setting the current filter when the list of filters is empty is useful if you want to restrict the displayed set of files without letting the user change it.

```
function List_Filters
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Object.Object_SList.GSlist;
```

Lists the current set of user-selectable filters. The list contains Gtk_File_Filter instances. Do not free the contents of list, still owned by gtk+, but you must free the list itself with Gtk.Object.Object_SList.Free.

107.3.7 Shorcut folders

```
function Add_Shortcut_Folder
  (Chooser            :      Gtk_File_Chooser;
   Folder             :      String)
   return Glib.Error.GError;
function Remove_Shortcut_Folder
  (Chooser            :      Gtk_File_Chooser;
   Folder             :      String)
   return Glib.Error.GError;
```

Adds a folder to be displayed with the shortcut folders in a file chooser. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "/usr/share/mydrawprogram/Clipart" folder to the volume list. Return null if the folder could be added successfully, or the code of the error otherwise.

```
function List_Shortcut_Folders
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Enums.String_SList.GSlist;
```

Queries the list of shortcut folders in the file chooser, as set by Add_Shortcut_Folder. The returned value must be freed by the user.

```
function Add_Shortcut_Folder_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String)
   return Glib.Error.GError;
function Remove_Shortcut_Folder_Uri
  (Chooser            :      Gtk_File_Chooser;
   Uri                :      String)
   return Glib.Error.GError;
```

Adds a folder URI to be displayed with the shortcut folders in a file chooser. Note that shortcut folders do not get saved, as they are provided by the application. For example, you can use this to add a "file:///usr/share/mydrawprogram/Clipart" folder to the volume list. Return null if the folder could be added successfully, or the code of the error otherwise.

```
function List_Shortcut_Folder_Uris
  (Chooser            :      Gtk_File_Chooser)
   return Gtk.Enums.String_SList.GSlist;
```

Queries the list of shortcut folders in the file chooser, as set by Add_Shortcut_Folder_Uri. The returned value must be freed by the user.

108 Package Gtk.File_Chooser_Button

The Gtk_File_Chooser_Button is a widget that lets the user select a file. It implements the Gtk_File_Chooser interface. Visually, it is a file name with a button to bring up a Gtk_File_Chooser_Dialog. The user can then use that dialog to change the file associated with that button. This widget does not support setting the "select-multiple" property to TRUE.

The Gtk_File_Chooser_Button supports the File_Chooser_Actions Action_Open and Action_Select_Folder.

The Gtk_File_Chooser_Button will ellipsize the label, and thus will request little horizontal space. To give the button more space, you should call Gtk.Widget.Size_Request, Set_Width_Chars, or pack the button in such a way that other interface elements give space to the widget.

108.1 Subprograms

```
function Get_Type                 return GType;
```

Return the internal value associated with a Gtk_File_Chooser_Button

```
procedure Gtk_New
  (Button             : out   Gtk_File_Chooser_Button;
   Title              :       String;
   Action             :       Gtk.File_Chooser.File_Chooser_Action);

procedure Gtk_New_With_Backend
  (Button             : out   Gtk_File_Chooser_Button;
   Title              :       String;
   Action             :       Gtk.File_Chooser.File_Chooser_Action;
   Backend            :       String);

procedure Gtk_New_With_Dialog
  (Button             : out   Gtk_File_Chooser_Button;
   Dialog             : access Gtk_File_Chooser_Dialog_Record'Class);

procedure Set_Title
  (Button             : access Gtk_File_Chooser_Button_Record;
   Title              :       String);

function Get_Title
  (Button             : access Gtk_File_Chooser_Button_Record)
   return String;
```

Modifies the Title of the browse dialog used by Button.

```
procedure Set_Width_Chars
  (Button             : access Gtk_File_Chooser_Button_Record;
   N_Chars            :       Gint);

function Get_Width_Chars
  (Button             : access Gtk_File_Chooser_Button_Record)
   return Gint;
```

Sets the width (in characters) that Button will use.

108.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_File_Chooser" @end itemize

```
function "+"
  (Button              : access Gtk_File_Chooser_Button_Record'Class)
   return Gtk.File_Chooser.Gtk_File_Chooser;
function "-"
  (File                :      Gtk.File_Chooser.Gtk_File_Chooser)
   return Gtk_File_Chooser_Button;
```

Converts to and from the Gtk_File_Chooser interface

109 Package Gtk.File_Chooser_Dialog

Gtk_File_Chooser_Dialog is a dialog box suitable for use with "File/Open" or "File/Save as" commands. This widget works by putting a Gtk_File_Chooser_Widget inside a Gtk_Dialog. It exposes the Gtk_File_Chooser interface, so you can use all of the Gtk_File_Chooser functions on the file chooser dialog as well as those for GtkDialog.

Note that Gtk_File_Chooser_Dialog does not have any methods of its own. Instead, you should use the functions that work on a Gtk_File_Chooser.

109.1 Subprograms

```
function Get_Type                 return GType;
```

Return the internal value associated with a Gtk_File_Chooser_Button

```
procedure Gtk_New
  (Dialog             : out   Gtk_File_Chooser_Dialog;
   Title              :       String;
   Parent             : access Gtk.Window.Gtk_Window_Record'Class;
   Action             :       Gtk.File_Chooser.File_Chooser_Action);
procedure Gtk_New_With_Backend
  (Dialog             : out   Gtk_File_Chooser_Dialog;
   Title              :       String;
   Parent             : access Gtk.Window.Gtk_Window_Record'Class;
   Action             :       Gtk.File_Chooser.File_Chooser_Action;
   Backend            :       String);
```

109.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_File_Chooser" @end itemize

```
function "+"
  (Dialog             : access Gtk_File_Chooser_Dialog_Record'Class)
   return Gtk.File_Chooser.Gtk_File_Chooser;
function "-"
  (File               :       Gtk.File_Chooser.Gtk_File_Chooser)
   return Gtk_File_Chooser_Dialog;
```

Converts to and from the Gtk_File_Chooser interface

110 Package Gtk.File_Chooser_Widget

Gtk_File_Chooser_Widget is a widget suitable for selecting files. It is the main building block of a Gtk_File_Chooser_Dialog. Most applications will only need to use the latter; you can use Gtk_File_Chooser_Widget as part of a larger window if you have special needs.

110.1 Subprograms

```
function Get_Type                  return GType;
```

Return the internal value associated with a Gtk_File_Chooser_Widget

```
procedure Gtk_New
  (Widget             : out   Gtk_File_Chooser_Widget;
   Action             :       Gtk.File_Chooser.File_Chooser_Action);
procedure Gtk_New_With_Backend
  (Widget             : out   Gtk_File_Chooser_Widget;
   Action             :       Gtk.File_Chooser.File_Chooser_Action;
   Backend            :       String);
```

110.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_File_Chooser" @end itemize

```
function "+"
  (Widget             : access Gtk_File_Chooser_Widget_Record'Class)
   return Gtk.File_Chooser.Gtk_File_Chooser;
function "-"
  (File               :       Gtk.File_Chooser.Gtk_File_Chooser)
   return Gtk_File_Chooser_Widget;
```

Converts to and from the Gtk_File_Chooser interface

111 Package Gtk.File_Filter

A Gtk_File_Filter can be used to restrict the files being shown in a Gtk_File_Chooser. Files can be filtered based on their name (with Add_Pattern), on their mime type (with Add_Mime_Type), or by a custom filter function (with Add_Custom).

Filtering by mime types handles aliasing and subclassing of mime types; e.g. a filter for text/plain also matches a file with mime type application/rtf, since application/rtf is a subclass of text/plain. Note that Gtk_File_Filter allows wildcards for the subtype of a mime type, so you can e.g. filter for image/*.

Normally, filters are used by adding them to a Gtk_File_Chooser, see Add_Filter, but it is also possible to manually use a filter on a file with Filter.

111.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_File_Filter       (Package Gtk.File_Filter)
```

111.2 Types

type File_Filter_Flags **is mod** 2 ** 8;

type File_Filter_Func **is access function**
```
        (Info : File_Filter_Info) return Gboolean;
```

type File_Filter_Info **is new** Glib.C_Proxy;

An opaque structure that contains information about a file

111.3 Subprograms

```
function Get_Filename
  (Info               :      File_Filter_Info)
   return String;
function Get_Uri
  (Info               :      File_Filter_Info)
   return String;
function Get_Display_Name
  (Info               :      File_Filter_Info)
   return String;
function Get_Mime_Type
  (Info               :      File_Filter_Info)
   return String;
```

Return the various information known about the file. The empty string is returned when the associated information is unknown. Display_Name is the string used to display the file in a file_chooser.

```
function Get_Type                return GType;
```

Return the internal value associated with a Gtk_File_Filter

```
procedure Gtk_New
  (Filter             : out   Gtk_File_Filter);
procedure Set_Name
  (Filter             : access Gtk_File_Filter_Record;
   Name               :      String);
function Get_Name
  (Filter             : access Gtk_File_Filter_Record)
   return String;
```

Sets the human-readable name of the filter; this is the string
that will be displayed in the file selector user interface if there is a selectable list of filters.

```
procedure Add_Mime_Type
  (Filter             : access Gtk_File_Filter_Record;
   Mime_Type          :      String);
```

Adds a rule allowing a given mime type to Filter.

```
procedure Add_Pattern
  (Filter             : access Gtk_File_Filter_Record;
   Pattern            :      String);
```

Adds a rule allowing a shell style glob to a filter.

```
procedure Add_Pixbuf_Formats
  (Filter             : access Gtk_File_Filter_Record);
```

Adds a rule allowing image files in the formats supported
by Gdk_Pixbuf.

```
procedure Add_Custom
  (Filter             : access Gtk_File_Filter_Record;
   Needed             :      File_Filter_Flags;
   Func               :      File_Filter_Func;
   Data               :      System.Address
                      := System.Null_Address;
   Notify             :      G_Destroy_Notify_Address
                      := null);
```

Adds rule to a filter that allows files based on a custom callback
function. The bitfield Needed which is passed in provides information about what sorts of information that the filter function needs; this allows GTK+ to avoid retrieving expensive information when it isn't needed by the filter. Notify is called when Data is no longer needed and should be freed.

112 Package Gtk.File_Selection

A Gtk_File_Selection is a general widget to interactively select file. It displays a dialog in which the user can navigate through directories, select a file, and even manipulate files with operations like removing, renaming,... Currently, only one file can be selected in the dialog.

112.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Window   (Package Gtk.Window)
               \___ Gtk_Dialog (Package Gtk.Dialog)
                  \___ Gtk_File_Selection (Package Gtk.File_Selection)
```

112.2 Subprograms

112.2.1 Operations on the dialog

```
procedure Gtk_New
  (File_Selection     : out   Gtk_File_Selection;
   Title              :      UTF8_String);
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_File_Selection.

```
procedure Set_Filename
  (File_Selection     : access Gtk_File_Selection_Record;
   Filename           :      UTF8_String);
function Get_Filename
  (File_Selection     : access Gtk_File_Selection_Record)
   return UTF8_String;
```

Highlight the given file in the dialog.
Note that this does not close the dialog. You can also use this procedure to select the directory to be displayed in the dialog. Along with Complete, this allows you to set some filters in the dialog.

```
function Get_Selections
  (Filesel            : access Gtk_File_Selection_Record)
   return GNAT.Strings.String_List;
```

Retrieves the list of file selections the user has made in the dialog
box. This function is intended for use when the user can select multiple files in the file list. The filenames are in the GLib file name encoding. To convert to UTF-8, call g_filename_to_utf8() on each string. The returned value must be freed by the caller

```
procedure Complete
  (File_Selection     : access Gtk_File_Selection_Record;
   Pattern            :      UTF8_String);
```

Set the filter used to display the files.
The pattern is displayed in the entry at the bottom of the dialog, and the list of files displayed in the list.

```
procedure Show_Fileop_Buttons
  (File_Selection     : access Gtk_File_Selection_Record);
```

```
procedure Hide_Fileop_Buttons
  (File_Selection     : access Gtk_File_Selection_Record);
```

When this function is called, the dialog includes a series of buttons for file operations (create directory, rename a file, delete a file).

```
procedure Set_Show_File_Op_Buttons
  (File_Selection     : access Gtk_File_Selection_Record;
   Flag               :      Boolean);
```

Choose whether to display or not the file operation buttons. If Flag is true, calls Show_Fileop_Buttons, otherwise calls Hide_Fileop_Buttons.

```
procedure Set_Select_Multiple
  (Filesel            : access Gtk_File_Selection_Record;
   Select_Multiple    :      Boolean);

function Get_Select_Multiple
  (Filesel            : access Gtk_File_Selection_Record)
   return Boolean;
```

Sets whether the user is allowed to select multiple files in the file list. Use Get_selections to get the list of selected files.

112.2.2 Getting the fields

The following functions are provided to access the fields of the@* file selection dialog. This dialog is divided into two main areas, the Action_Area which is the top part that contains the list of files, and the button area which is the bottom part that contains the OK and Cancel buttons.

```
function Get_Action_Area
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Box.Gtk_Box;
```

Return the action area. This is the area that contains the list of files, the filter entry,etc.

```
function Get_Button_Area
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Box.Gtk_Box;
```

Return the button area. This is the area that contains the OK and Cancel buttons.

```
function Get_Dir_List
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the list that display the list of directories.

```
function Get_File_List
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the list that display the list of files in the selected directory

```
function Get_Cancel_Button
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Button.Gtk_Button;
```

Return the Cancel button. To remove this button from the dialog, call Hide on the return value. The callbacks on this button should simply close the dialog, but should ignore the file selected by the user.

```
function Get_Help_Button
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Button.Gtk_Button;
```

Return the Help button.
To remove this button from the dialog, call Hide on the return value. The callbacks on this button should display a new dialog that explain what file the user should select.

```
function Get_Ok_Button
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Button.Gtk_Button;
```

Return the OK button.
The callbacks on this button should close the dialog and do something with the file selected by the user.

```
function Get_History_Pulldown
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the menu that display the history of directories
for easy access by the user.

```
function Get_Selection_Entry
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the entry used to set the filter on the list of directories.
The simplest way to insert text in this entry is to use the Complete procedure above.

```
function Get_Selection_Text
  (File_Selection     : access Gtk_File_Selection_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the text displayed just above the Selection_Entry.

113 Package Gtk.Fixed

The Gtk_Fixed widget is a container which can place child widgets at fixed positions and with fixed sizes, given in pixels.

Note that it is usually bad practice to use the Gtk_Fixed container in GtkAda. Instead, you should consider using one of the other many containers available, that will allow you to handle resizing of your windows, as well as font size changes easily.

113.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Fixed       (Package Gtk.Fixed)
```

113.2 Subprograms

```
procedure Gtk_New
  (Fixed              : out   Gtk_Fixed);
```

Create a new fixed container.

```
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Fixed.

```
procedure Put
  (Fixed              : access Gtk_Fixed_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   X                  :      Gint;
   Y                  :      Gint);
```

Add Widget to a Fixed container at the given position.
X indicates the horizontal position to place the widget at. Y is the vertical position to place the widget at.

```
procedure Move
  (Fixed              : access Gtk_Fixed_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   X                  :      Gint;
   Y                  :      Gint);
```

Move a child of a GtkFixed container to the given position.
X indicates the horizontal position to place the widget at. Y is the vertical position to place the widget at.

```
procedure Set_Has_Window
  (Fixed              : access Gtk_Fixed_Record;
   Has_Window         :      Boolean := False);
function Get_Has_Window
  (Fixed              : access Gtk_Fixed_Record)
   return Boolean;
```

Sets whether a Gtk_Fixed widget is created with a separate
Gdk_Window for or not. (By default, it will be created with no separate Gdk_Window). This function must be called while the widget is not realized, for instance, immediately after the window is created.

114 Package Gtk.Font_Button

The Gtk_Font_Button is a button which displays the currently selected font an allows to open a font selection dialog to change the font. It is suitable widget for selecting a font in a preference dialog.

114.1 Signals

- **"font-set"**

  ```
  procedure Handler (Button : access Gtk_Font_Button_Record'Class);
  ```

  The font-set signal is emitted when the user selects a font. When handling this signal, use Get_Font_Name to find out which font was just selected. Note that this signal is only emitted when the user changes the font. If you need to react to programmatic font changes as well, use the notify::font-name signal.

114.2 Subprograms

```
procedure Gtk_New
  (Font_Button        : out   Gtk_Font_Button);
procedure Gtk_New_With_Font
  (Font_Button        : out   Gtk_Font_Button;
   Fontname           :       String);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Font_Button

```
function Set_Font_Name
  (Font_Button        : access Gtk_Font_Button_Record;
   Fontname           :       String)
   return Boolean;
function Get_Font_Name
  (Font_Button        : access Gtk_Font_Button_Record)
   return String;
```

Sets or updates the currently-displayed font in font picker dialog.
Returns the value of Gtk.Font_Selection.Set_Font_Name if the font selection dialog exists, False otherwise.

```
procedure Set_Show_Size
  (Font_Button        : access Gtk_Font_Button_Record;
   Show_Size          :       Boolean);
function Get_Show_Size
  (Font_Button        : access Gtk_Font_Button_Record)
   return Boolean;
```

If Show_Size is True, the font size will be displayed along with the
name of the selected font.

```
procedure Set_Show_Style
  (Font_Button        : access Gtk_Font_Button_Record;
   Show_Style         :       Boolean);
function Get_Show_Style
  (Font_Button        : access Gtk_Font_Button_Record)
   return Boolean;
```

Returns whether the name of the font style will be shown in the label.

```
procedure Set_Title
  (Font_Button        : access Gtk_Font_Button_Record;
   Title              :      String);
function Get_Title
  (Font_Button        : access Gtk_Font_Button_Record)
   return String;
```

Retrieves the title of the font selection dialog.

```
procedure Set_Use_Font
  (Font_Button        : access Gtk_Font_Button_Record;
   Use_Font           :      Boolean);
function Get_Use_Font
  (Font_Button        : access Gtk_Font_Button_Record)
   return Boolean;
```

If Use_Font is True, the font name will be written using the selected font.

```
procedure Set_Use_Size
  (Font_Button        : access Gtk_Font_Button_Record;
   Use_Size           :      Boolean);
function Get_Use_Size
  (Font_Button        : access Gtk_Font_Button_Record)
   return Boolean;
```

If Use_Size is True, the font name will be written using the selected size.

115 Package Gtk.Font_Selection

This widget provides a nice way for the user of your application to select fonts. It first searches on your system for the list of fonts available, and displays a set of boxes to select them based on their name, their weight, their size, etc. This widget is provided in two forms, one widget that can be embedded in any container, a Gtk_Font_Selection, whereas the other one comes directly in its own separate window (to be popped up as a dialog).

Some filters can be applied to the widget, when you want the user to select only a font only among a specific subset (like bitmap or true-type fonts for instance). There are two kinds of filters: a base filter, set in your application and that the user can not change; a user filter that can be modified interactively by the user.

115.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Window   (Package Gtk.Window)
               \___ Gtk_Dialog (Package Gtk.Dialog)
                  \___ Gtk_Font_Selection_Dialog
                          (Package Gtk.Font_Selection_Dialog)
```

115.2 Subprograms

115.2.1 Font_Selection functions

```
procedure Gtk_New
  (Widget             : out   Gtk_Font_Selection);

function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Font_Selection.

```
function Set_Font_Name
  (Fontsel            : access Gtk_Font_Selection_Record;
   Fontname           :      String)
   return Boolean;

function Get_Font_Name
  (Fontsel            : access Gtk_Font_Selection_Record)
   return String;
```

Set the name and attributes of the selected font in Fontsel.
Fontname should have the format described in Pango.Font. Fontself must have been displayed on the screen already

```
procedure Set_Preview_Text
  (Fontsel            : access Gtk_Font_Selection_Record;
   Text               :      UTF8_String);

function Get_Preview_Text
  (Fontsel            : access Gtk_Font_Selection_Record)
   return UTF8_String;
```

Set or Get the string used to preview the selected font in the dialog.

115.2.2 Font_Selection_Dialog functions

```
procedure Gtk_New
  (Widget             : out   Gtk_Font_Selection_Dialog;
   Title              :       UTF8_String);
function Dialog_Get_Type       return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Font_Selection_Dialog.

```
function Set_Font_Name
  (Fsd                : access Gtk_Font_Selection_Dialog_Record;
   Fontname           :       String)
   return Boolean;
function Get_Font_Name
  (Fsd                : access Gtk_Font_Selection_Dialog_Record)
   return String;
```

Return the name of the font selected by the user.
It returns an empty string if not font is selected. The string has the same format as excepted in the Gdk.Font package. This is also the standard format on X11 systems.

```
procedure Set_Preview_Text
  (Fsd                : access Gtk_Font_Selection_Dialog_Record;
   Text               :       UTF8_String);
function Get_Preview_Text
  (Fsd                : access Gtk_Font_Selection_Dialog_Record)
   return UTF8_String;
```

Return the string used to preview the selected font in the dialog.

```
function Get_Cancel_Button
  (Fsd                : access Gtk_Font_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Return the Id of the cancel button of the dialog.
You can use this to set up a callback on that button. The callback should close the dialog, and ignore any value that has been set in it.

```
function Get_OK_Button
  (Fsd                : access Gtk_Font_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Return the Id of the Ok button.
The callback set on this button should close the dialog if the selected font is valid, and do whatever if should with it.

```
function Get_Apply_Button
  (Fsd                : access Gtk_Font_Selection_Dialog_Record)
   return Gtk.Button.Gtk_Button;
```

Return the Id of the Apply button.
The callback on this button should temporarily apply the font, but should be able to cancel its effect if the Cancel button is selected.

116 Package Gtk.Font_Selection_Dialog

117 Package Gtk.Frame

A Gtk_Frame is a simple border than can be added to any widget or group of widget to enhance its visual aspect. Optionally, a frame can have a title.

This is a very convenient widget to visually group related widgets (like groups of buttons for instance), possibly with a title to explain the purpose of this group.

A Gtk_Frame has only one child, so you have to put a container like for instance a Gtk_Box inside if you want the frame to surround multiple widgets.

117.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Frame    (Package Gtk.Frame)
```

117.2 Subprograms

```
procedure Gtk_New
  (Frame              : out   Gtk_Frame;
   Label              :       UTF8_String := "");
```

Create a new frame.
If Label is not the empty string, the frame will have a title.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Frame.

```
procedure Set_Label
  (Frame              : access Gtk_Frame_Record;
   Label              :       UTF8_String := "");
function Get_Label
  (Frame              : access Gtk_Frame_Record)
   return UTF8_String;
```

Change the label of the frame dynamically.
If Label is the empty string, the frame's label is deleted.

```
procedure Set_Label_Widget
  (Frame              : access Gtk_Frame_Record;
   Label_Widget       : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Label_Widget
  (Frame              : access Gtk_Frame_Record)
   return Gtk.Widget.Gtk_Widget;
```

Set the label widget for the frame.
This is the widget that will appear embedded in the top edge of the frame as a title.

```
procedure Set_Label_Align
  (Frame              : access Gtk_Frame_Record;
   Xalign             :       Gfloat := 0.0;
   Yalign             :       Gfloat := 0.0);
```

Change the alignment of the title in the frame.
Xalign and Yalign are both percents that indicate the exact position of the label relative to the top-left corner of the frame. Note that Yalign is currently ignored, and the label can

only be displayed on the top of the frame (0.0 for Xalign means align the label on the left, 1.0 means align the label on the right).

```
procedure Get_Label_Align
  (Frame              : access Gtk_Frame_Record;
   Xalign             : out   Gfloat;
   Yalign             : out   Gfloat);
```

Return the X and Y alignments of the title in the frame.

```
procedure Set_Shadow_Type
  (Frame              : access Gtk_Frame_Record;
   The_Type           :      Gtk_Shadow_Type);
```

Change the visual aspect of the frame.

```
function Get_Shadow_Type
  (Frame              : access Gtk_Frame_Record)
   return Gtk_Shadow_Type;
```

Return the visual aspect of the frame.

118 Package Gtk.GC

This package provides a convenitn function to create a new graphic context. Such contexts are needed in several places in GtkAda, in particular for the drawing routines, and this function provides a convenient way to create them.

118.1 Subprograms

```
function Get
  (Depth              :      Gint;
   Colormap           :      Gdk.Gdk_Colormap;
   Values             :      Gdk.GC.Gdk_GC_Values;
   Values_Mask        :      Gdk.GC.Gdk_GC_Values_Mask)
   return Gdk_GC;
```

Create a new GC with the matching attributes.
If such a graphic context already exists, it is returned, which is much faster than creating a new one. Creating a new context requires a round-trip to the server (X11 for instance), and can be slow. You shouldn't modify the attributes of the returned context, since that might impact other parts of the code that have queried it.

```
procedure Release
  (Gc                 :      Gdk_GC);
```

Decrease the reference counting for the GC. If it reaches 0, then
calling Get will create a new one the next time it is called with the same attributes.

119 Package Gtk.GEntry

A Gtk_Entry is a single line text editing widget. The text is automatically scrolled if it is longer than can be displayed on the screen, so that the cursor position is visible at all times.

See Gtk_Text_View for a multiple-line text editing widget.

119.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Editable       (Package Gtk.Editable)
         \___ Gtk_Entry       (Package Gtk.GEntry)
```

119.2 Signals

- **"activate"**

  ```
  procedure Handler (Ent : access Gtk_Entry_Record'Class);
  ```

  Called when the entry is activated, for instance when the user presses <enter> while in it

- **"copy_clipboard"**

  ```
  procedure Handler (Ent : access Gtk_Entry_Record'Class);
  ```

  You should emit this signal to request that the current selection be copied into the clipboard. This is mostly used from key bindings.

- **"cut_clipboard"**

  ```
  procedure Handler (Ent : access Gtk_Entry_Record'Class);
  ```

  You should emit this signal to request that the current selection be deleted and copied into the clipboard. This is mostly used from key bindings.

- **"delete_from_cursor"**

  ```
  procedure Handler
  (Ent              : access Gtk_Entry_Record'Class;
  Step             : Gtk_Movement_Step;
  Amount           : Gint);
  ```

  You should emit this signal to request that some text be delete from the cursor position.

- **"insert_at_cursor"**

  ```
  procedure Handler
  (Ent  : access Gtk_Entry_Record'Class;
  Text : String);
  ```

  You should emit this signal to request that some text be inserted at the current cursor location. This is mostly used from key bindings.

- **"move_cursor"**

  ```
  procedure Handler
  (Ent              : access Gtk_Entry_Record'Class;
  Step             : Gtk_Movement_Step;
  Amount           : Gint;
  Extend_Selection : Boolean);
  ```

  You should emit this signal to request that the cursor be moved. This is mostly used when connected to a keybinding, as is done by default for the arrow keys for instance.

- **"paste_clipboard"**

  ```
  procedure Handler (Ent : access Gtk_Entry_Record'Class);
  ```

  You should emit this signal to request that the clipboard be inserted at the current cursor location. This is mostly used from key bindings.

- **"populate_popup"**

  ```
  procedure Handler
  (Ent  : access Gtk_Entry_Record'Class;
  Menu : access Gtk_Menu_Record'Class);
  ```

  ???

- **"toggle_overwrite"**

  ```
  procedure Handler (Ent : access Gtk_Entry_Record'Class);
  ```

  You should emit this signal to request that the insertion mode be changed. This is mostly used from a key binding, as is done by default for the Insert key.

119.3 Types

subtype Gtk_GEntry **is** Gtk_Entry;

119.4 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Entry);
```

Create a new entry with no maximum length for the text

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Entry.

```
procedure Set_Visibility
  (The_Entry          : access Gtk_Entry_Record;
   Visible            :      Boolean);
function Get_Visibility
  (The_Entry          : access Gtk_Entry_Record)
   return Boolean;
```

Set the visibility of the characters in the entry.
If Visible is set to False, the characters will be replaced with the invisible character ('*' by default) in the display, and when the text is copied elsewhere.

```
procedure Set_Invisible_Char
  (The_Entry          : access Gtk_Entry_Record;
   Char               :      Gunichar);
function Get_Invisible_Char
  (The_Entry          : access Gtk_Entry_Record)
   return Gunichar;
```

Set the character to use in place of the actual text when
Set_Visibility has been called to set text visibility to False. i.e. this is the character used in "password mode" to show the user how many characters have been typed. The default invisible char is an asterisk ('*'). If you set the invisible char to 0, then the user will get no feedback at all; there will be no text on the screen as they type. for entries with visisbility set to false. See Set_Invisible_Char.

```
procedure Set_Has_Frame
  (The_Entry          : access Gtk_Entry_Record;
   Setting            :      Boolean := True);

function Get_Has_Frame
  (The_Entry          : access Gtk_Entry_Record)
   return Boolean;
```

Set whether the entry has a beveled frame around it.

```
procedure Set_Max_Length
  (The_Entry          : access Gtk_Entry_Record;
   Max                :      Gint);

function Get_Max_Length
  (The_Entry          : access Gtk_Entry_Record)
   return Gint;
```

Set the maximum length for the text.
The current text is truncated if needed.

```
procedure Set_Activates_Default
  (The_Entry          : access Gtk_Entry_Record;
   Setting            :      Boolean);

function Get_Activates_Default
  (The_Entry          : access Gtk_Entry_Record)
   return Boolean;
```

If Setting is True, pressing Enter in the Entry will activate the
default widget for the window containing the entry. This usually means that the dialog box containing the entry will be closed, since the default widget is usually one of the dialog buttons.

(For experts: if Setting is True, the entry calls Gtk.Window.Activate_Default on the window containing the entry, in the default handler for the "activate" signal.)

```
procedure Set_Width_Chars
  (The_Entry          : access Gtk_Entry_Record'Class;
   Width              :      Gint);

function Get_Width_Chars
  (The_Entry          : access Gtk_Entry_Record'Class)
   return Gint;
```

Number of characters to leave space for in the entry, on the screen.
This is the number of visible characters, not the maximal number of characters the entry can contain

```
procedure Set_Text
  (The_Entry          : access Gtk_Entry_Record;
   Text               :      UTF8_String);

function Get_Text
  (The_Entry          : access Gtk_Entry_Record)
   return UTF8_String;
```

Modify the text in the entry.
The text is cut at the maximum length that was set when the entry was created. The text replaces the current contents.

```
procedure Set_Alignment
  (Ent                : access Gtk_Entry_Record;
   Xalign             :      Gfloat);
```

```
function Get_Alignment
  (Ent                 : access Gtk_Entry_Record)
   return Gfloat;
```

Sets the alignment for the contents of the entry. This controls
the horizontal positioning of the contents when the displayed text is shorter than the width of the entry.

```
procedure Set_Completion
  (Ent                 : access Gtk_Entry_Record;
   Completion          : access Gtk_Entry_Completion_Record'Class);
function Get_Completion
  (Ent                 : access Gtk_Entry_Record)
   return Gtk_Entry_Completion;
```

Sets Completion to be the auxiliary completion object to use with Ent.
All further configuration of the completion mechanism is done on Completion using the Gtk.Entry_Completion API.

```
function Text_Index_To_Layout_Index
  (Ent                 : access Gtk_Entry_Record;
   Text_Index          :      Gint)
   return Gint;
```

Converts from a position in the entry's layout (returned by Get_Layout)
to a position in the entry contents (returned by Get_Text). Returns the byte index into the entry layout text

```
function Layout_Index_To_Text_Index
  (Ent                 : access Gtk_Entry_Record;
   Layout_Index        :      Gint)
   return Gint;
```

Converts from a position in the entry contents (returned
by Get_Text) to a position in the entry's layout (returned by Get_Layout, with text retrieved via pango.layout.Get_Text). Return the byte index into the entry contents

```
procedure Get_Layout_Offsets
  (The_Entry           : access Gtk_Entry_Record;
   X                   : out   Gint;
   Y                   : out   Gint);
```

Obtain the position of the Pango_Layout used to render text
in the entry, in widget coordinates. Useful if you want to line up the text in an entry with some other text, e.g. when using the entry to implement editable cells in a sheet widget.

Also useful to convert mouse events into coordinates inside the Pango_Layout, e.g. to take some action if some part of the entry text is clicked.

Note that as the user scrolls around in the entry the offsets will change; you'll need to connect to the "notify::scroll_offset" signal to track this. Remember when using the Pango_Layout functions you need to convert to and from pixels using Pango_Pixels or Pango_Scale.

```
function Get_Layout
  (The_Entry           : access Gtk_Entry_Record)
   return Pango.Layout.Pango_Layout;
```

Return the widget that manages all the layout of text (left-to-right,
right-to-left, fonts,...). Changing the font used for the entry should be done by changing the font using for this layout. Note that you should also change the font in the Pango_Context

returned by Get_Pango_Context, or the next keypress event in the entry will restore the default initial font.

The layout is useful to e.g. convert text positions to pixel positions, in combination with Get_Layout_Offsets. The returned layout is owned by the entry so need not be freed by the caller.

120 Package Gtk.GLArea

This widget is derived from Gtk_Drawing_Area and provides an area where it is possible to use the openGL API.

120.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Drawing_Area   (Package Gtk.Drawing_Area)
         \___ Gtk_GLArea      (Package Gtk.GLArea)
```

120.2 Types

type Attributes_Array **is array** (Natural **range** <>) **of** GL_Configs;

Note: as opposed to what exists in C, you don't need to have the last element in the array be GDK_GL_NONE. This is done transparently by GtkAda itself.

120.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_GLArea;
   Attr_List          :      Attributes_Array);
```

Make an OpenGL widget, Attr_List is passed to glXChooseVisual GLX call.
Attr_List specifies a list of Boolean attributes and enum/integer attribute/value pairs. See glXChooseVisual man page for more explanation on Attr_List. Widget is created with visual and colormap of the requested type and GLX context is created for this widget. You can't do opengl calls on widget until it has X window. X window is not created until widget is realized.

```
procedure Gtk_New
  (Widget             : out   Gtk_GLArea;
   Attr_List          :      Attributes_Array;
   Share              : access Gtk_GLArea_Record'Class);
```

Same as above.
Share specifies the widget with which to share display lists and texture objects. A non initialized value indicates that no sharing is to take place.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_GLArea.

```
function Make_Current
  (Glarea             : access Gtk_GLArea_Record'Class)
   return Boolean;
```

Must be called before rendering into OpenGL widgets.
Return True if rendering to widget is possible. Rendering is not possible if widget is not Gtk_GLArea widget or widget is not realized.

```
procedure Swap_Buffers
  (Glarea             : access Gtk_GLArea_Record'Class);
```

Promote contents of back buffer of Glarea to front buffer.
The contents of front buffer become undefined.

121 Package Gtk.GRange

This widget provides a low level graphical representation of a range of values. It is used by other widgets such as Gtk_Scale and Gtk_Scrollbar.

121.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Range          (Package Gtk.Grange)
```

121.2 Signals

- **"adjust_bounds"**

  ```
  procedure Handler
  (R     : access Gtk_Range_Record'Class;
  Value : Gdouble);
  ```

- **"change_value"**

  ```
  function Handler
  (R     : access Gtk_Range_Record'Class;
  Typ   : Gtk_Scroll_Type;
  Value : Gdouble) return Gboolean;
  ```

  Emitted when a scroll action is performed on the range. The type of event that occurred and the new value are returned. The application should return True if it has handled the event itself.

- **"move_slider"**

  ```
  procedure Handler (R : access Gtk_Range_Record'Class);
  ```

  Emitted when the slider has changed

- **"value_changed"**

  ```
  procedure Handler (R :  access Gtk_Range_Record'Class);
  ```

  Emitted when the current value of the range has changed

121.3 Types

subtype Gtk_GRange **is** Gtk_Range;

121.4 Subprograms

```
function Get_Type                  return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Range.

```
procedure Set_Update_Policy
  (The_Range          : access Gtk_Range_Record;
   Policy             :      Gtk_Update_Type);
```

Set the update policy for the range.
Update_Continuous means that anytime the range slider is moved, the range value will change and the value_changed signal will be emitted. Update_Delayed means that the

value will be updated after a brief timeout where no slider motion occurs, so updates are spaced by a short time rather than continuous. Update_Discontinuous means that the value will only be updated when the user releases the button and ends the slider drag operation.

```
function Get_Update_Policy
  (The_Range          : access Gtk_Range_Record)
   return Gtk_Update_Type;
```

Return the current update policy.

```
procedure Set_Adjustment
  (The_Range          : access Gtk_Range_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
```

Set the adjustment to be used as the "model" object for this range widget. The adjustment indicates the current range value, the minimum and maximum range values, the step/page increments used for keybindings and scrolling, and the page size. The page size is normally 0 for Gtk_Scale and nonzero for Gtk_Scrollbar, and indicates the size of the visible area of the widget being scrolled. The page size affects the size of the scrollbar slider.

```
function Get_Adjustment
  (The_Range          : access Gtk_Range_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment associated with the range widget.

```
procedure Set_Inverted
  (The_Range          : access Gtk_Range_Record;
   Setting            :      Boolean := True);
```

Ranges normally move from lower to higher values as the slider moves from top to bottom or left to right. Inverted ranges have higher values at the top or on the right rather than on the bottom or left.

```
function Get_Inverted
  (The_Range          : access Gtk_Range_Record)
   return Boolean;
```

Return whether the range is inverted.

```
procedure Set_Increments
  (The_Range          : access Gtk_Range_Record;
   Step               :      Gdouble;
   Page               :      Gdouble);
```

Set the Step and the Page size for the range. The Step size is used when the user clicks on the Gtk_Scrollbar arrows or moves the Gtk_Scale via the arrow keys. The Page size is used when moving by pages via the Page-Up and Page-Down keys for instance.

```
procedure Set_Range
  (The_Range          : access Gtk_Range_Record;
   Min                :      Gdouble;
   Max                :      Gdouble);
```

Set the allowable values in the Gtk_Range, and clamps the range value to the between Min and Max.

```
procedure Set_Value
  (The_Range          : access Gtk_Range_Record;
   Value              :      Gdouble);
```

Set the current value of the given Range. If the value is outside the minimum or the maximum value range, it will be clamped to fit inside the range. Cause the "value_changed" signal to be emitted if the value is different.

```
function Get_Value
  (The_Range           : access Gtk_Range_Record)
   return Gdouble;
```

Return the current value of the range.

122 Package Gtk.Gamma_Curve

The Gtk_Gamma_Curve widget is a child of Gtk_Curve specifically for editing gamma curves, which are used in graphics applications such as the Gimp.

The Gamma_Curve widget shows a curve which the user can edit with the mouse just like a Gtk_Curve widget. On the right of the curve it also displays 5 buttons, 3 of which change between the 3 curve modes (spline, linear and free), and the other 2 set the curve to a particular gamma value, or reset it to a straight line.

122.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Box          (Package Gtk.Box)
            \___ Gtk_Gamma_Curve (Package Gtk.Gamma_Curve)
```

122.2 Subprograms

```
procedure Gtk_New
  (Gamma_Curve        : out   Gtk_Gamma_Curve);
```

Create a new Gtk_Gamma_Curve.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Gamma_Curve.

```
function Get_Curve
  (Gamma_Curve        : access Gtk_Gamma_Curve_Record)
   return Gtk.Curve.Gtk_Curve;
```

Return the Curve widget associated with a Gamma_Curve.

```
function Get_Gamma
  (Gamma_Curve        : access Gtk_Gamma_Curve_Record)
   return Gfloat;
```

Return the Gamma value associated with a Gamma_Curve.

123 Package Gtk.Grange

124 Package Gtk.Handle_Box

The Gtk_Handle_Box widget allows a portion of a window to be "torn off". It is a bin widget which displays its child and a handle that the user can drag to tear off a separate window (the float window) containing the child widget. A thin ghost is drawn in the original location of the handlebox. By dragging the separate window back to its original location, it can be reattached.

When reattaching, the ghost and float window, must be aligned along one of the edges, the snap edge. This either can be specified by the application programmer explicitely, or GtkAda will pick a reasonable default based on the handle position.

To make detaching and reattaching the handlebox as minimally confusing as possible to the user, it is important to set the snap edge so that the snap edge does not move when the handlebox is detached. For instance, if the handlebox is packed at the bottom of a Vbox, then when the handlebox is detached, the bottom edge of the handlebox's allocation will remain fixed as the height of the handlebox shrinks, so the snap edge should be set to Pos_Bottom.

124.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Handle_Box (Package Gtk.Handle_Box)
```

124.2 Signals

- **"child_attached"**

  ```
  procedure Handler
  (Handle_Box : access Gtk_Handle_Box_Record'Class;
  Widget      : access Gtk_Widget_Record'Class);
  ```

  Emitted when the contents of the Handle_Box are reattached to the main window. Widget is the child widget of the Handle_Box. (this argument provides no extra information and is here only for backwards-compatibility)

- **"child_detached"**

  ```
  procedure Handler
  (Handle_Box : access Gtk_Handle_Box_Record'Class;
  Widget      : access Gtk_Widget_Record'Class);
  ```

  Emitted when the contents of the Handle_Box are detached from the main window. See "child-attached" for drtails on the parameters.

124.3 Subprograms

```
procedure Gtk_New
  (Handle_Box         : out   Gtk_Handle_Box);
```

Create a new Handle_Box.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Dialog.

```
procedure Set_Shadow_Type
  (Handle_Box         : access Gtk_Handle_Box_Record;
   Typ                :      Enums.Gtk_Shadow_Type);
function Get_Shadow_Type
  (Handle_Box         : access Gtk_Handle_Box_Record)
   return Enums.Gtk_Shadow_Type;
```

Sets or gets the type of shadow to be drawn around the border of the Handle_Box.

```
procedure Set_Handle_Position
  (Handle_Box         : access Gtk_Handle_Box_Record;
   Position           :      Enums.Gtk_Position_Type);
function Get_Handle_Position
  (Handle_Box         : access Gtk_Handle_Box_Record)
   return Enums.Gtk_Position_Type;
```

Sets or gets the side of the Handle_Box where the handle is drawn.

```
procedure Set_Snap_Edge
  (Handle_Box         : access Gtk_Handle_Box_Record;
   Edge               :      Enums.Gtk_Position_Type);
function Get_Snap_Edge
  (Handle_Box         : access Gtk_Handle_Box_Record)
   return Enums.Gtk_Position_Type;
```

Sets or gets the snap edge of a Handle_Box.
The snap edge is the edge of the detached child that must be aligned with the corresponding edge of the "ghost" left behind when the child was detached to reattach the torn-off window. Usually, the snap edge should be chosen so that it stays in the same place on the screen when the Handle_Box is torn off.

If the snap edge is not set, then an appropriate value will be guessed from the handle position. If the handle position is Pos_Right or Pos_Left, then the snap edge will be Pos_Top, otherwise it will be Pos_Left.

125 Package Gtk.Handlers

The aim of this package is to provide some services to connect a handler to a signal emitted by a Gtk Object. To understand the services provided by this package, some definitions are necessary:

Signal: A signal is a kind of message that an object wants to broadcast. All GObjects can emit signals. These messages are associated to certain events happening during the life of an object. For instance, when a user clicks on a button, the "clicked" signal is emitted by the button.

Handler (or callback): A handler is a function or procedure that the user "connects" to a signal for a particular object. Connecting a handler to a signal means associating this handler to the signal. When the signal is emitted, all connected handlers are called back. Usually, the role of those callbacks is to do some processing triggered by a user action. For instance, when "clicked" signal is emitted by the "OK" button of a dialog, the connected handler can be used to close the dialog or recompute some value.

In GtkAda, the handlers are defined in a form as general as possible. The first argument is always an access to the object it has been connected to. The second object is a table of values (See Glib.Values for more details about this table). It is the responsibility of this handler to extract the values from it, and to convert them to the correct Ada type.

Because such handlers are not very convenient to use, this package also provides some services to connect a marshaller instead. It will then do the extraction work before calling the more programmer-friendly handler, as defined in Gtk.Marshallers (see Gtk.Marshallers for more details).

The subdivision of this package is identical to Gtk.Marshallers; it is made of four generic sub-packages, each representing one of the four possible kinds of handlers: they can return a value or not, and they can have some user specific data associated to them or not. Selecting the right package depends on the profile of the handler. For example, the handler for the "delete_event" signal of a Gtk_Window has a return value, and has an extra parameter (a Gint). All handlers also have a user_data field by default, but its usage is optional. To connect a handler to this signal, if the user_data field is not used, the Return_Callback generic should be instantiated. On the other hand, if the user_data field is necessary, then the User_Return_Callback generic should be used.

Note also that the real handler in Gtk+ should expect at least as many arguments as in the marshaller you are using. If your marshaller has one argument, the C handler must have at least one argument too.

The common generic parameter to all sub-packages is the widget type, which is the basic widget manipulated. This can be Glib.Object.GObject_Record type if you want to reduce the number of instantiations, but the conversion to the original type will have to be done inside the handler.

All sub-packages are organized in the same way.

First, the type "Handler" is defined. It represents the general form of the callbacks supported by the sub-package.

The corresponding sub-package of Gtk.Marshallers is instantiated.

A series of "Connect" procedures and functions is given. All cases are covered: the functions return the Handler_Id of the newly created association, while the procedures just

connect the handler, dropping the Handler_Id; some services allow the user to connect a Handler while some others allow the usage of Marshallers, which are more convenient. Note that more than one handler may be connected to a signal; the handlers will then be invoked in the order of connection.

Some "Connect_Object" services are also provided. Those services never have a user_data. They accept an additional parameter called Slot_Object. When the callback in invoked, the Gtk Object emitting the signal is substituted by this Slot_Object. These callbacks are always automatically disconnected as soon as one of the two widgets involved is destroyed.

There are several methods to connect a handler. For each method, although the option of connecting a Handler is provided, the recommended way is to use Marshallers. Each connect service is documented below, in the first sub-package.

A series of "To_Marshaller" functions are provided. They return some marshallers for the most commonly used types in order to ease the usage of this package. Most of the time, it will not be necessary to use some other marshallers. For instance, if a signal is documented as receiving a single argument, the widget (for instance the "clicked" signal for a Gtk_Button), you will connect to it with: with Gtkada.Handlers; procedure On_Clicked (Button : access Gtk_Widget_Record'Class); ... Widget_Callback.Connect (Button, "clicked", On_Clicked'Access);

The simple form above also applies for most handlers that take one additional argument, for instance the "button_press_event" in gtk-widget.ads. Just declare your subprogram with the appropriate profile and connect it, as in: with Gtkada.Handlers; procedure On_Button (Widget : access Gtk_Widget_Record'Class; Event : Gdk_Event); ... Widget_Callback.Connect (Widget, "button_press_event", On_Button'Access);

More complex forms of handlers exists however in GtkAda, for which no predefined marshaller exists. In this case, you have to use the general form of callbacks. For instance, the "select_row" signal of Gtk.Clist. with Gtkada.Handlers; with Gtk.Arguments; procedure On_Select (Clist : access Gtk_Widget_Record'Class; Args : Glib.Values.GValues) is Row : constant Gint := To_Gint (Args, 1); Column : constant Gint := To_Gint (Args, 2); Event : constant Gdk_Event := To_Event (Args, 3); begin ... end On_Select; ... Widget_Callback.Connect (Clist, "select_row", On_Select'Access);

As for the "To_Marshaller" functions, a series of "Emit_By_Name" procedures are also provided for the same most common types, to allow the user to easily emit signals. These procedures are mainly intended for people building new GObjects.

At the end of this package, some general services related to the management of signals and handlers are also provided. Each one of them is documented individually below.

IMPORTANT NOTE: These packages must be instantiated at library-level

125.1 Types

```
type Handler is access function
        (Widget    : access Widget_Type'Class;
```

```
type Return_Type is
          (<>);
```

```
type Simple_Handler is access function
          (Widget     : access Widget_Type'Class;
```

```
type User_Type is private;
```

```
type Widget_Type is new Glib.Object.GObject_Record with private;
```

125.2 Subprograms

See also the package User_Callback_With_Setup@* ——————————————————————————

```
      procedure Connect
        (Widget              : access Widget_Type'Class;
         Name                :      Glib.Signal_Name;
         Marsh               :      Marshallers.Marshaller;
         User_Data           :      User_Type;
         After               :      Boolean := False);
      procedure Object_Connect
        (Widget              : access Glib.Object.GObject_Record'Class;
         Name                :      Glib.Signal_Name;
         Marsh               :      Marshallers.Marshaller;
         Slot_Object         : access Widget_Type'Class;
         User_Data           :      User_Type;
         After               :      Boolean := False);
      procedure Connect
        (Widget              : access Widget_Type'Class;
         Name                :      Glib.Signal_Name;
         Cb                  :      Simple_Handler;
         User_Data           :      User_Type;
         After               :      Boolean := False);
      procedure Object_Connect
        (Widget              : access Glib.Object.GObject_Record'Class;
         Name                :      Glib.Signal_Name;
         Cb                  :      Simple_Handler;
         Slot_Object         : access Widget_Type'Class;
         User_Data           :      User_Type;
         After               :      Boolean := False);
      procedure Connect
        (Widget              : access Widget_Type'Class;
         Name                :      Glib.Signal_Name;
         Cb                  :      Handler;
         User_Data           :      User_Type;
         After               :      Boolean := False);
```

```
procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False);

function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;

function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;

function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;

function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;

function To_Marshaller
  (Cb                  :      Gint_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Guint_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Event_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Widget_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Marshallers.Void_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Notebook_Page_Marshaller.Handler)
   return Marshallers.Marshaller;
```

```
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gint)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Guint)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gdk.Event.Gdk_Event)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gtk.Notebook.Gtk_Notebook_Page)
   return Return_Type;
```

125.2.1 User_Return_Callback_With_Setup

This package is basically the same as User_Return_Callback, except that@* an extra function (Setup) is called after a handler has been connected. Typical usage is to automatically call Add_Watch (see below) in case the User_Type is (or contains) widgets.

```
procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   User_Data          :      User_Type;
   After              :      Boolean := False);
procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False);
procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   User_Data          :      User_Type;
   After              :      Boolean := False);
procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
```

```
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False);
procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Simple_Handler;
   User_Data           :      User_Type;
   After               :      Boolean := False);
procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Simple_Handler;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False);
function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;
function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False)
   return Handler_Id;
function To_Marshaller
  (Cb                  :      Gint_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Guint_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Event_Marshaller.Handler)
```

```
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Widget_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Internal_Cb.Marshallers.Void_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Notebook_Page_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gint)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Guint)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gdk.Event.Gdk_Event)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name)
   return Return_Type;
function Emit_By_Name
  (Object             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Param              :      Gtk.Notebook.Gtk_Notebook_Page)
   return Return_Type;
```

```
procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   After              :      Boolean := False);
procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   Slot_Object        : access Widget_Type'Class;
   After              :      Boolean := False);
procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
```

```
   After               :      Boolean := False);
procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   After               :      Boolean := False);
procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Simple_Handler;
   After               :      Boolean := False);
procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Simple_Handler;
   Slot_Object         : access Widget_Type'Class;
   After               :      Boolean := False);
function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   After               :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   Slot_Object         : access Widget_Type'Class;
   After               :      Boolean := False)
   return Handler_Id;
function Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   After               :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
   After               :      Boolean := False)
   return Handler_Id;
function To_Marshaller
  (Cb                  :      Gint_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Guint_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Event_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Widget_Marshaller.Handler)
   return Marshallers.Marshaller;
```

```
function To_Marshaller
  (Cb                  :      Marshallers.Void_Marshaller.Handler)
   return Marshallers.Marshaller;

function To_Marshaller
  (Cb                  :      Notebook_Page_Marshaller.Handler)
   return Marshallers.Marshaller;

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gint);

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Guint);

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gdk.Event.Gdk_Event);

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               : access Gtk.Widget.Gtk_Widget_Record'Class);

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name);

procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gtk.Notebook.Gtk_Notebook_Page);
```

See also the package User_Callback_With_Setup@* ————————————————-

```
procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   User_Data           :      User_Type;
   After               :      Boolean := False);

procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False);

procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   User_Data           :      User_Type;
   After               :      Boolean := False);

procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   Slot_Object         : access Widget_Type'Class;
```

```
   User_Data          :      User_Type;
   After              :      Boolean := False);
procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Simple_Handler;
   User_Data          :      User_Type;
   After              :      Boolean := False);
procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Simple_Handler;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False);
function Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;
function Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;
function Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;
function To_Marshaller
  (Cb                 :      Gint_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Guint_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Event_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                 :      Widget_Marshaller.Handler)
```

```
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Marshallers.Void_Marshaller.Handler)
   return Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Notebook_Page_Marshaller.Handler)
   return Marshallers.Marshaller;
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gint);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Guint);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gdk.Event.Gdk_Event);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               : access Gtk.Widget.Gtk_Widget_Record'Class);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gtk.Notebook.Gtk_Notebook_Page);
```

125.2.2 User_Callback_With_Setup

This package is basically the same as User_Callback, except that an@* extra function (Setup) is called after a handler has been connected. Typical usage is to automatically call Add_Watch (see below) in case the User_Type is (or contains) widgets.

```
procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   User_Data           :      User_Type;
   After               :      Boolean := False);
procedure Object_Connect
  (Widget              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name;
   Marsh               :      Marshallers.Marshaller;
   Slot_Object         : access Widget_Type'Class;
   User_Data           :      User_Type;
   After               :      Boolean := False);
procedure Connect
  (Widget              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Cb                  :      Handler;
   User_Data           :      User_Type;
   After               :      Boolean := False);
```

```
procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False);

procedure Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Simple_Handler;
   User_Data          :      User_Type;
   After              :      Boolean := False);

procedure Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Simple_Handler;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False);

function Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;

function Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Marsh              :      Marshallers.Marshaller;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;

function Connect
  (Widget             : access Widget_Type'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;

function Object_Connect
  (Widget             : access Glib.Object.GObject_Record'Class;
   Name               :      Glib.Signal_Name;
   Cb                 :      Handler;
   Slot_Object        : access Widget_Type'Class;
   User_Data          :      User_Type;
   After              :      Boolean := False)
   return Handler_Id;

function To_Marshaller
  (Cb                 :      Gint_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;

function To_Marshaller
  (Cb                 :      Guint_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
```

```
function To_Marshaller
  (Cb                  :      Event_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Widget_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Internal_Cb.Marshallers.Void_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
function To_Marshaller
  (Cb                  :      Notebook_Page_Marshaller.Handler)
   return Internal_Cb.Marshallers.Marshaller;
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gint);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Guint);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gdk.Event.Gdk_Event);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               : access Gtk.Widget.Gtk_Widget_Record'Class);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name);
procedure Emit_By_Name
  (Object              : access Widget_Type'Class;
   Name                :      Glib.Signal_Name;
   Param               :      Gtk.Notebook.Gtk_Notebook_Page);
procedure Add_Watch
  (Id                  :      Handler_Id;
   Object              : access Glib.Object.GObject_Record'Class);
```

Make sure that when Object is destroyed, the handler Id is also destroyed. This function should mostly be used in cases where you use a User_Data that is Object. If you don't destroy the callback at the same time, then the next time the callback is called it will try to access some invalid memory (Object being destroyed), and you will likely get a Storage_Error.

```
procedure Disconnect
  (Object              : access Glib.Object.GObject_Record'Class;
   Id                  : in out Handler_Id);
```

Disconnect the handler identified by the given Handler_Id.

```
procedure Emit_Stop_By_Name
  (Object              : access Glib.Object.GObject_Record'Class;
   Name                :      Glib.Signal_Name);
```

During a signal emission, invoking this procedure will halt the emission.

```
procedure Handler_Block
  (Obj                : access Glib.Object.GObject_Record'Class;
   Id                 :      Handler_Id);
```

Blocks temporily the signal. For each call to this procedure,
a call to Handler_Unblock must be performed in order to really unblock the signal.

```
procedure Handlers_Destroy
  (Obj                : access Glib.Object.GObject_Record'Class);
```

Destroys all the handlers associated to the given object.

```
procedure Handler_Unblock
  (Obj                : access Glib.Object.GObject_Record'Class;
   Id                 :      Handler_Id);
```

See Handler_Block.

125.3 Example

```
--  This example connects the "delete_event" signal to a widget.
--  The handlers for this signal get an extra argument which is
--  the Gdk_Event that generated the signal.

with Gtk.Handlers;    use Gtk.Handlers;
with Gtk.Marshallers; use Gtk.Marshallers;

function My_Cb (Widget : access Gtk_Widget_Record'Class;
                Event  : Gdk.Event.Gdk_Event)
                return Gint;
--  your own function

package Return_Widget_Cb is new Gtk.Handlers.Return_Callback
   (Gtk.Widget.Gtk_Widget_Record, Gint);

Return_Widget_Cb.Connect (W, "delete_event",
   Return_Widget_Cb.To_Marshaller (My_Cb'Access));
```

126 Package Gtk.Hbutton_Box

A Gtk_Hbutton_Box is a specific Gtk_Button_Box that organizes its children horizontally. The beginning of the box (when you add children with Gtk.Box.Pack_Start) is on the left of the box. Its end (for Gtk.Box.Pack_End) is on the right.

126.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Box          (Package Gtk.Box)
            \___ Gtk_Button_Box (Package Gtk.Button_Box)
               \___ Gtk_Hbutton_Box (Package Gtk.Hbutton_Box)
```

126.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Hbutton_Box);
```

Create a new horizontal button box.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_HButton_Box.

127 Package Gtk.Icon_Factory

Browse the available stock icons in the list of stock IDs found here. You can also use the gtk-demo application for this purpose.

An icon factory manages a collection of Gtk_Icon_Set; a Gtk_Icon_Set manages set of variants of a particular icon (i.e. a Gtk_Icon_Set contains variants for different sizes and widget states). Icons in an icon factory are named by a stock ID, which is a simple string identifying the icon. Each Gtk_Style has a list of Gtk_Icon_Factory derived from the current theme; those icon factories are consulted first when searching for an icon. If the theme doesn't set a particular icon, GTK+ looks for the icon in a list of default icon factories, maintained by gtk.icon_factory.add_default and gtk.icon_factory.remove_default. Applications with icons should add default icon factory with their icons, which will allow themes to override the icons for the application.

To display an icon, always use Lookup_Icon_Set on the widget that will display the icon, or the convenience function Gtk.Widget.Render_Icon. These functions take the theme into account when looking up the icon to use for a given stock ID.

127.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Icon_Factory      (Package Gtk.Icon_Factory)
```

127.2 Types

type Gtk_Icon_Set **is new** Glib.C_Proxy;

type Gtk_Icon_Source **is new** Glib.C_Proxy;

127.3 Subprograms

127.3.1 Icon factories

```
procedure Gtk_New
  (Widget             : out   Gtk_Icon_Factory);

function Get_Type             return Glib.GType;
```

Return the internal value associated with a Icon_Factory.

```
procedure Add_Default
  (Factory            : access Gtk_Icon_Factory_Record);
```

Adds an icon factory to the list of icon factories searched by Lookup_Icon_Set. This means that, for example, Gtk.Image.New_From_Stock will be able to find icons in Factory. There will normally be an icon factory added for each library or application that comes with icons. The default icon factories can be overridden by themes.

```
procedure Remove_Default
  (Factory            : access Gtk_Icon_Factory_Record);
```

Removes an icon factory from the list of default icon factories. Not normally used; you might use it for a library that can be unloaded or shut down.

127.3.2 Icon sets

```
function Gtk_New                return Gtk_Icon_Set;
```

Create an empty Gtk_Icon_Set.

```
function Gtk_New
  (Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf)
   return Gtk_Icon_Set;
```

Creates a new icon set with Pixbuf as the default/fallback source image. If you don't add any additional icon sources (see below) to the icon set, all variants of the icon will be created from Pixbuf, using scaling, pixelation, etc. as required to adjust the icon size or make the icon look insensitive/prelighted.

```
procedure Add
  (Factory            : access Gtk_Icon_Factory_Record;
   Stock_Id           :      String;
   Set                :      Gtk_Icon_Set);
```

Adds the given icon set to the icon factory, under the name Stock_Id. Stock_Id should be namespaced for your application, e.g. "myapp-whatever-icon". Normally applications create an icon factory, then add it to the list of default factories with Add_Default. Then they pass the Stock_Id to widgets such as Gtk_Image to display the icon. Themes can provide an icon with the same name (such as "myapp-whatever-icon") to override your application's default icons. If an icon already existed in Factory for Stock_Id, it is unreferenced and replaced with the new icon set.

```
function Lookup_Icon_Set
  (Style              : access Gtk.Style.Gtk_Style_Record'Class;
   Stock_Id           :      String)
   return Gtk_Icon_Set;
```

Retrieve an icon set by its name. The icon might exist in various sizes, that can be manipulated through the result set

```
function Lookup
  (Factory            : access Gtk_Icon_Factory_Record;
   Stock_Id           :      String)
   return Gtk_Icon_Set;
```

Looks up Stock_Id in the icon factory, returning an icon set if found, otherwise null. For display to the user, you should use Lookup_Icon_Set on the Gtk_Style for the widget that will display the icon, instead of using this function directly, so that themes are taken into account.

```
function Lookup_Default
  (Stock_Id           :      String)
   return Gtk_Icon_Set;
```

Looks for an icon in the list of default icon factories. For display to the user, you should use Lookup_Icon_Set on the Gtk_Style for the widget that will display the icon, instead of using this function directly, so that themes are taken into account.

```
function Icon_Set_Get_Type     return Glib.GType;
```

Return the internal type value for Gtk_Icon_Set

```
function Copy
  (Icon_Set           :      Gtk_Icon_Set)
   return Gtk_Icon_Set;
function Get_Sizes
  (Icon_Set           :      Gtk_Icon_Set)
   return Gint_Array;
```

Obtains a list of icon sizes this icon set can render.

```
function Ref
  (Icon_Set           :      Gtk_Icon_Set)
   return Gtk_Icon_Set;
function Unref
  (Icon_Set           :      Gtk_Icon_Set)
   return Gtk_Icon_Set;
function Render_Icon
  (Icon_Set           :      Gtk_Icon_Set;
   Style              : access Gtk.Style.Gtk_Style_Record'Class;
   Direction          :      Gtk.Enums.Gtk_Text_Direction;
   State              :      Gtk.Enums.Gtk_State_Type;
   Size               :      Gtk.Enums.Gtk_Icon_Size;
   Widget             :      Gtk.Widget.Gtk_Widget := null;
   Detail             :      String := "")
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Renders an icon using Render_Icon below. In most cases, the other version of Render_Icon is better, since it automatically provides most of the arguments from the current widget settings. This function never returns null; if the icon can't be rendered (perhaps because an image file fails to load), a default "missing image" icon will be returned instead. Widget is the widget that will display the icon, or null. This is typically used to determine the screen (and thus the colormap depth). Detail is the string to pass to the theme engine to provide more information. Passing anything but the empty string will disable caching.

127.3.3 Icon sources

```
function Gtk_New               return Gtk_Icon_Source;
```

Create a new icon source.

```
function Icon_Source_Copy
  (Source             :      Gtk_Icon_Source)
   return Gtk_Icon_Source;
```

Returns a copy of Gtk_Icon_Source. This function is generally not useful

```
function Icon_Source_Get_Type  return Glib.GType;
```

Return the internal type used for Gtk_Icon_Source

```
function Render_Icon
  (Style              : access Gtk.Style.Gtk_Style_Record'Class;
   Source             :      Gtk.Icon_Factory.Gtk_Icon_Source;
   Direction          :      Gtk.Enums.Gtk_Text_Direction;
   State              :      Gtk.Enums.Gtk_State_Type;
   Size               :      Gtk.Enums.Gtk_Icon_Size;
   Widget             :      Gtk.Widget.Gtk_Widget := null;
```

```
   Detail              :      String := "")
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Renders the icon specified by Source at the given Size according to the given parameters and returns the result in pixbuf.

```
procedure Add_Source
  (Set                 :      Gtk_Icon_Set;
   Source              :      Gtk_Icon_Source);
```

Icon sets have a list of icon sources, which they use as base icons for rendering icons in different states and sizes. Icons are scaled, made to look insensitive, etc. in Gtk.Icon.Set_Render_Icon, but an icon set needs base images to work with. The base images and when to use them are described by an icon source. This function copies Source, so you can reuse the same source immediately without affecting the icon set.

```
procedure Free
  (Source              :      Gtk_Icon_Source);
```

Free memory allocated to Source.

```
procedure Set_Filename
  (Source              :      Gtk_Icon_Source;
   Filename            :      String);
function Get_Filename
  (Source              :      Gtk_Icon_Source)
   return String;
```

Set the name of an image file to use as a base image when creating icon variants for an icon set. The filename must be absolute.

```
procedure Set_Pixbuf
  (Source              :      Gtk_Icon_Source;
   Pixbuf              :      Gdk.Pixbuf.Gdk_Pixbuf);
function Get_Pixbuf
  (Source              :      Gtk_Icon_Source)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Set a pixbuf to use as a base image when creating icon variants for an icon set. If an icon source has both a filename and a pixbuf set, the pixbuf will take priority.

```
procedure Set_Size
  (Source              :      Gtk_Icon_Source;
   Size                :      Gtk.Enums.Gtk_Icon_Size);
function Get_Size
  (Source              :      Gtk_Icon_Source)
   return Gtk.Enums.Gtk_Icon_Size;
```

Set the icon size this icon source is intended to be used with

```
procedure Set_Icon_Name
  (Source              :      Gtk_Icon_Source;
   Icon_Name           :      String);
function Get_Icon_Name
  (Source              :      Gtk_Icon_Source)
   return String;
```

Retrieves the source icon name, or "" if none is set. The icon comes from an icon theme in this case.

```
procedure Set_Size_Wildcarded
  (Source              :      Gtk_Icon_Source;
   Wildcarded          :      Boolean);
```

```
function Get_Size_Wildcarded
  (Source             :      Gtk_Icon_Source)
   return Boolean;
```

Change the wilcarded state of the size for the icon source.

If the icon size is wildcarded, this source can be used as the base image for an icon of any size. if the size is not wildcarded, then the size the source applies to should be set with Set_Size, and the icon source will only be used with that specific size.

Gtk prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible.

Gtk will normally scale wildcarded source images to produce an appropriate icon at a given size, but will not change the size of source images that match exactly.

```
procedure Set_Direction_Wildcarded
  (Source             :      Gtk_Icon_Source;
   Setting            :      Boolean);

function Get_Direction_Wildcarded
  (Source             :      Gtk_Icon_Source)
   return Boolean;
```

If the text direction is wildcarded, this source can be used as the base image for an icon in any Gtk_Text_Direction. If the text direction is not wildcarded, then the text direction the icon source applies to should be set with Set_Direction(), and the icon source will only be used with that text direction. Gtk_Icon_Set prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible

```
procedure Set_Direction
  (Source             :      Gtk_Icon_Source;
   Direction          :      Gtk.Enums.Gtk_Text_Direction);

function Get_Direction
  (Source             :      Gtk_Icon_Source)
   return Gtk.Enums.Gtk_Text_Direction;
```

Sets the text direction this icon source is intended to be used with. Setting the text direction on an icon source makes no difference if the text direction is wildcarded. Therefore, you should usually call Set_Direction_Wildcarded() to un-wildcard it in addition to calling this function.

```
procedure Set_State_Wildcarded
  (Source             :      Gtk_Icon_Source;
   Setting            :      Boolean);

function Get_State_Wildcarded
  (Source             :      Gtk_Icon_Source)
   return Boolean;
```

If the widget state is wildcarded, this source can be used as the base image for an icon in any Gtk_State_Type. If the widget state is not wildcarded, then the state the source applies to should be set with Set_State and the icon source will only be used with that specific state. Gtk_Icon_Set prefers non-wildcarded sources (exact matches) over wildcarded sources, and will use an exact match when possible. Gtk_Icon_Set will normally transform wildcarded source images to produce an appropriate icon for a given state, for example lightening an image on prelight, but will not modify source images that match exactly.

```
procedure Set_State
  (Source             :      Gtk_Icon_Source;
   State              :      Gtk.Enums.Gtk_State_Type);
function Get_State
  (Source             :      Gtk_Icon_Source)
   return Gtk.Enums.Gtk_State_Type;
```

Sets the widget state this icon source is intended to be used with. Setting the widget state on an icon source makes no difference if the state is wildcarded. Therefore, you should usually call Set_State_Wildcarded to un-wildcard it in addition to calling this function.

127.3.4 Icon sizes

There are a number of predefined icon sizes (see gtk-enums.ads). These@* are used in the various gtk+ contexts. However, you can also define your own icon sizes to use in your application.

```
function Icon_Size_From_Name
  (Name               :      String)
   return Gtk.Enums.Gtk_Icon_Size;
```

Looks up the icon size associated with Name. Predefined icon sizes are associated with the following names: Icon_Size_Menu => "gtk-menu" (16x16) Icon_Size_Button => "gtk-button" (20x20) Icon_Size_Small_Toolbar => "gtk-small-toolbar" (18x18) Icon_Size_Large_Toolbar => "gtk-large-toolbar" (24x24) Icon_Size_Dnd => "gtk-dnd" (32x32) Icon_Size_Dialog => "gtk-dialog" (48x48) You can also define your own names (see Icon_Size_Register)

```
function Icon_Size_Get_Name
  (Size               :      Gtk.Enums.Gtk_Icon_Size)
   return String;
```

Gets the canonical name of the given icon size

```
procedure Icon_Size_Lookup
  (Size               :      Gtk.Enums.Gtk_Icon_Size;
   Width, Height      : out  Gint);
```

Obtains the pixel size of a semantic icon size, possibly modified by user preferences for the default Gtk_Settings. (See Icon_Size_Lookup_For_Settings). This function isn't normally needed, Render_Icon is the usual way to get an icon for rendering, then just look at the size of the rendered pixbuf. The rendered pixbuf may not even correspond to the width/height returned by Icon_Size_Lookup, because themes are free to render the pixbuf however they like, including changing the usual size. Sizes are set to -1 if Size is invalid.

```
procedure Icon_Size_Lookup_For_Settings
  (Settings           : access Gtk.Settings.Gtk_Settings_Record'Class;
   Size               :      Gtk.Enums.Gtk_Icon_Size;
   Width              : out  Gint;
   Height             : out  Gint);
```

Obtains the pixel size of a semantic icon size, possibly modified by user preferences for a particular Gtk_Settings.

```
function Icon_Size_Register
  (Name               :      String;
   Width              :      Gint;
   Height             :      Gint)
```

```
      return Gtk.Enums.Gtk_Icon_Size;
```

Registers a new icon size, along the same lines as Icon_Size_Menu, etc. Returns the integer value for the size.

```
   procedure Icon_Size_Register_Alias
     (Alias              :      String;
      Target             :      Gtk.Enums.Gtk_Icon_Size);
```

Registers Alias as another name for Target.
So calling Icon_Size_From_Name with Alias as argument will return Target

128 Package Gtk.Icon_Theme

Gtk_Icon_Theme provides a facility for looking up icons by name and size. The main reason for using a name rather than simply providing a filename is to allow different icons to be used depending on what icon theme is selecetd by the user. The operation of icon themes on Linux and Unix follows the Icon Theme Specification. There is a default icon theme, named hicolor where applications should install their icons, but more additional application themes can be installed as operating system vendors and users choose.

Named icons are similar to the Themeable Stock Images(3) facility, and the distinction between the two may be a bit confusing. A few things to keep in mind:

@itemize @bullet @item Stock images usually are used in conjunction with Stock Items(3)., such as STOCK_OK or STOCK_OPEN. Named icons are easier to set up and therefore are more useful for new icons that an application wants to add, such as application icons or window icons.

@item Stock images can only be loaded at the symbolic sizes defined by the Gtk_Icon_Size enumeration, or by custom sizes defined by Gtk.Icon_Factory.Icon_Size_Register, while named icons are more flexible and any pixel size can be specified.

@item Because stock images are closely tied to stock items, and thus to actions in the user interface, stock images may come in multiple variants for different widget states or writing directions.

@end itemize A good rule of thumb is that if there is a stock image for what you want to use, use it, otherwise use a named icon. It turns out that internally stock images are generally defined in terms of one or more named icons. (An example of the more than one case is icons that depend on writing direction; STOCK_GO_FORWARD uses the two themed icons "gtk-stock-go-forward-ltr" and "gtk-stock-go-forward-rtl".)

In many cases, named themes are used indirectly, via Gtk_Image or stock items, rather than directly, but looking up icons directly is also simple. The Gtk_Icon_Theme object acts as a database of all the icons in the current theme. You can create new Gtk_Icon_Theme objects, but its much more efficient to use the standard icon theme for the Gdk_Screen so that the icon information is shared with other people looking up icons. In the case where the default screen is being used, looking up an icon can be as simple as: Theme := Get_Default; Pixbuf := Load_Icon (Theme, "my-icon-name", 48, 0, Error); if Pixbuf = null then Put_Line ("Error " & Get_Message (Error); Error_Free (Error); end if;

128.1 Signals

- **"changed"**

  ```
  procedure Handler (Theme : access Gtk_Icon_Theme_Record'Class);
  ```

  Emitted when the current icon theme is switched or GTK+ detects that a change has occurred in the contents of the current icon theme.

128.2 Types

type Gtk_Icon_Info **is new** Glib.C_Proxy;

```
type Gtk_Icon_Lookup_Flags is mod Gint'Last;
```

128.3 Subprograms

128.3.1 Icon_Info

```
function Icon_Info_Get_Type    return GType;
function Copy
  (Icon_Info          :      Gtk_Icon_Info)
   return Gtk_Icon_Info;
```

Make a copy of a Icon_Info

```
procedure Free
  (Icon_Info          :      Gtk_Icon_Info);
```

Free a Gtk_Icon_Info and associated information

```
function Get_Attach_Points
  (Icon_Info          :      Gtk_Icon_Info)
   return Gdk.Types.Gdk_Points_Array;
```

Fetches the set of attach points for an icon. An attach point is a location in the icon that can be used as anchor points for attaching emblems or overlays to the icon.

```
function Get_Base_Size
  (Icon_Info          :      Gtk_Icon_Info)
   return Gint;
```

Gets the base size for the icon. The base size is a size for the icon that was specified by the icon theme creator. This may be different than the actual size of image; an example of this is small emblem icons that can be attached to a larger icon. These icons will be given the same base size as the larger icons to which they are attached.

Return value: the base size, or 0, if no base size is known for the icon.

```
function Get_Builtin_Pixbuf
  (Icon_Info          :      Gtk_Icon_Info)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Gets the built-in image for this icon, if any. To allow GTK+ to use built in icon images, you must pass the %GTK_ICON_LOOKUP_USE_BUILTIN to Gtk.Icon_Theme.Lookup_Icon.

Return value: the built-in image pixbuf, or null. No extra reference is added to the returned pixbuf, so if you want to keep it around, you must use Ref. The returned image must not be modified.

```
function Get_Display_Name
  (Icon_Info          :      Gtk_Icon_Info)
   return String;
```

Gets the display name for an icon. A display name is a string to be used in place of the icon name in a user visible context like a list of icons.

```
procedure Get_Embedded_Rect
  (Icon_Info          :      Gtk_Icon_Info;
   Rectangle          : in out Gdk.Rectangle.Gdk_Rectangle;
   Has_Embedded_Rectangle  : out   Boolean);
```

Gets the coordinates of a rectangle within the icon that can be used for display of information such as a preview of the contents of a text file. See Set_Raw_Coordinates for further information about the coordinate system. Rectangle is only set if Has_Embedded_Rectangle is set to True.

```
function Get_Filename
  (Icon_Info          :      Gtk_Icon_Info)
   return String;
```

Gets the filename for the icon. If the %GTK_ICON_LOOKUP_USE_BUILTIN flag was passed to Lookup_Icon, there may be no filename if a builtin icon is returned; in this case, you should use Get_Builtin_Pixbuf. Return value: the filename for the icon, or "" if Get_Builtin_Pixbuf() should be used instead.

```
function Load_Icon
  (Icon_Info          :      Gtk_Icon_Info;
   Error              :      Glib.Error.GError_Access
                      := null)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Renders an icon previously looked up in an icon theme using Lookup_Icon; the size will be based on the size passed to Lookup_Icon. Note that the resulting pixbuf may not be exactly this size; an icon theme may have icons that differ slightly from their nominal sizes, and in addition GTK+ will avoid scaling icons that it considers sufficiently close to the requested size or for which the source image would have to be scaled up too far. (This maintains sharpness.) Return value: the rendered icon; this may be a newly created icon or a new reference to an internal icon, so you must not modify the icon. Use Unref to release your reference to the icon.

```
procedure Set_Raw_Coordinates
  (Icon_Info          :      Gtk_Icon_Info;
   Raw_Coordinates    :      Boolean);
```

Sets whether the coordinates returned by Get_Embedded_Rect and Get_Attach_Points should be returned in their original form as specified in the icon theme, instead of scaled appropriately for the pixbuf returned by Load_Icon.

Raw coordinates are somewhat strange; they are specified to be with respect to the unscaled pixmap for PNG and XPM icons, but for SVG icons, they are in a 1000x1000 coordinate space that is scaled to the final size of the icon. You can determine if the icon is an SVG icon by using Get_Filename, and seeing if it is non-empty and ends in '.svg'.

This function is provided primarily to allow compatibility wrappers for older API's, and is not expected to be useful for applications.

128.3.2 Search path

```
procedure Set_Search_Path
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Path               :      GNAT.Strings.String_List);
```

Sets the search path for the icon theme object. When looking for an icon theme, GTK+ will search for a subdirectory of one or more of the directories in Path with the same name as the icon theme. (Themes from multiple of the path elements are combined to allow themes to be extended by adding icons in the user's home directory.)

In addition if an icon found isn't found either in the current icon theme or the default icon theme, and an image file with the right name is found directly in one of the elements

of Path, then that image will be used for the icon name. (This is legacy feature, and new icons should be put into the default icon theme, which is called DEFAULT_THEME_NAME, rather than directly on the icon path.)

```
procedure Append_Search_Path
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Path               :      String);
procedure Prepend_Search_Path
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Path               :      String);
```

Appends or prepends a directory to the search path.
Set_Search_Path.

```
function Get_Search_Path
  (Icon_Theme         : access Gtk_Icon_Theme_Record)
   return GNAT.Strings.String_List;
```

Gets the current search path

128.3.3 Icon themes

```
procedure Gtk_New
  (Theme              : out   Gtk_Icon_Theme);
function Get_Type              return GType;
```

Return the internal type associated with icon themes

```
procedure Add_Builtin_Icon
  (Icon_Name          :      String;
   Size               :      Gint;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf);
```

Registers a built-in icon for icon theme lookups. The idea of built-in icons is to allow an application or library that uses themed icons to function requiring files to be present in the file system. For instance, the default images for all of GTK+'s stock icons are registered as built-icons.

In general, if you use Add_Builtin_Icon you should also install the icon in the icon theme, so that the icon is generally available.

This function will generally be used with pixbufs loaded via Gdk.Pixbuf.Gdk_New_From-Inline.

```
function Get_Default           return Gtk_Icon_Theme;
```

Gets the icon theme for the default screen. See Get_For_Screen. Return value: A unique Gtk_Icon_Theme associated with the default screen. This icon theme is associated with the screen and can be used as long as the screen is open. Do not ref or unref it.

```
function Get_Example_Icon_Name
  (Icon_Theme         : access Gtk_Icon_Theme_Record)
   return String;
```

Gets the name of an icon that is representative of the current theme (for instance, to use when presenting a list of themes to the user.)

```
function Get_Icon_Sizes
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Icon_Name          :      String)
   return Glib.Gint_Array;
```

Returns an array of integers describing the sizes at which the icon is available without scaling. A size of -1 means that the icon is available in a scalable format.

```
function Has_Icon
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Icon_Name          :      String)
   return Boolean;
```

Checks whether an icon theme includes an icon for a particular name.

```
function List_Icons
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Context            :      String := "")
   return Gtk.Enums.String_List.Glist;
```

Lists the icons in the current icon theme. Only a subset of the icons can be listed by providing a context string. The set of values for the context string is system dependent, but will typically include such values as "Applications" and "MimeTypes". You must free the list with Gtk.Enums.Free_String_List.

```
function Load_Icon
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Icon_Name          :      String;
   Size               :      Glib.Gint;
   Flags              :      Gtk_Icon_Lookup_Flags)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Looks up an icon in an icon theme, scales it to the given size and renders it into a pixbuf. This is a convenience function; if more details about the icon are needed, use Lookup_Icon followed by Load_Icon.

Note that you probably want to listen for icon theme changes and update the icon. This is usually done by connecting to the GtkWidget::style-set signal. If for some reason you do not want to update the icon when the icon theme changes, you should consider using Gdk.Pixbuf.Copy to make a private copy of the pixbuf returned by this function. Otherwise GTK+ may need to keep the old icon theme loaded, which would be a waste of memory.

Return value: the rendered icon; this may be a newly created icon or a new reference to an internal icon, so you must not modify the icon. Use Unref to release your reference to the icon.

```
function Lookup_Icon
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Icon_Name          :      String;
   Size               :      Glib.Gint;
   Flags              :      Gtk_Icon_Lookup_Flags)
   return Gtk_Icon_Info;
```

Looks up a named icon and returns a structure containing information such as the filename of the icon. The icon can then be rendered into a pixbuf using Load_Icon. (Load_Icon combines these two steps if all you need is the pixbuf) Free the returned value with Free.

```
function Rescan_If_Needed
  (Icon_Theme         : access Gtk_Icon_Theme_Record)
   return Boolean;
```

Checks to see if the icon theme has changed; if it has, any currently cached information is discarded and will be reloaded next time Icon_Theme is accessed.

```
procedure Set_Custom_Theme
  (Icon_Theme         : access Gtk_Icon_Theme_Record;
   Theme_Name         :      String);
```

Sets the name of the icon theme that the Gtk_Icon_Theme object uses
overriding system configuration. This function cannot be called on the icon theme objects
returned from Get_Default and Get_For_Screen.

129 Package Gtk.Icon_View

Gtk_Icon_View provides an alternative view on a list model. It displays the model as a grid of icons with labels. Like Gtk_Tree_View, it allows to select one or multiple items (depending on the selection mode, see Set_Selection_Mode). In addition to selection with the arrow keys, Gtk_Icon_View supports rubberband selection, which is controlled by dragging the pointer.

129.1 Signals

- **"activate_cursor_item"**
- **"item_activated"**
- **"move_cursor"**
- **"select_all"**
- **"select_cursor_item"**
- **"selection_changed"**
- **"set_scroll_adjustments"**
- **"toggle_cursor_item"**
- **"unselect_all"**

129.2 Types

```
type Gtk_Icon_View_Drop_Position is
           (No_Drop,
            Drop_Into,
            Drop_Left,
            Drop_Right,
            Drop_Above,
            Drop_Below);
```

An enum for determining where a dropped item goes. If Drop_Into, then the drop item replaces the item.

129.3 Subprograms

```
procedure Gtk_New
  (Icon_View          : out   Gtk_Icon_View);
procedure Gtk_New_With_Model
  (Icon_View          : out   Gtk_Icon_View;
   Model              : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class);
function Get_Type               return GType;
```

Return the internal type used for a Gtk_Icon_View.

```
procedure Set_Column_Spacing
  (Icon_View          : access Gtk_Icon_View_Record;
   Column_Spacing     :      Glib.Gint);
function Get_Column_Spacing
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::column-spacing property which specifies the space
which is inserted between the columns of the icon view.

```
procedure Set_Columns
  (Icon_View          : access Gtk_Icon_View_Record;
   Columns            :      Glib.Gint);
function Get_Columns
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::columns property which determines in how many columns the
icons are arranged. If Columns is -1, the number of columns will be chosen automatically to fill the available area.

```
procedure Set_Cursor
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Cell               :      Gtk.Cell_Renderer.Gtk_Cell_Renderer
                      := null;
   Start_Editing      :      Boolean := False);
```

Sets the current keyboard focus to be at Path, and selects it. This is
useful when you want to focus the user's attention on a particular item. If Cell is not null, then focus is given to the cell specified by it. Additionally, if Start_Editing is True, then editing should be started in the specified cell.

This function is often followed by Grab_Focus in order to give keyboard focus to the widget. Please note that editing can only happen when the widget is realized.

```
procedure Get_Cursor
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Cell               : out   Gtk.Cell_Renderer.Gtk_Cell_Renderer;
   Cursor_Is_Set      : out   Boolean);
```

Fills in Path and Cell with the current cursor path and cell.
If the cursor isn't currently set, then Ppath will be null. If no cell currently has focus, then Cell will be null. The returned Path must be freed with Gtk.Tree_Model.Path_Free. Cursor_Is_Set is set to True if the cursor is set.

```
procedure Set_Item_Width
  (Icon_View          : access Gtk_Icon_View_Record;
   Item_Width         :      Glib.Gint);
```

```
function Get_Item_Width
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::item-width property which specifies the width to use for each item. If it is set to -1, the icon view will automatically determine a suitable item size.

```
procedure Set_Margin
  (Icon_View          : access Gtk_Icon_View_Record;
   Margin             :      Glib.Gint);
function Get_Margin
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::margin property which specifies the space which is inserted at the top, bottom, left and right of the icon view.

```
procedure Set_Orientation
  (Icon_View          : access Gtk_Icon_View_Record;
   Orientation        :      Gtk.Enums.Gtk_Orientation);
function Get_Orientation
  (Icon_View          : access Gtk_Icon_View_Record)
   return Gtk.Enums.Gtk_Orientation;
```

Sets the ::orientation property which determines whether the labels are drawn beside the icons instead of below.

```
procedure Set_Reorderable
  (Icon_View          : access Gtk_Icon_View_Record;
   Reorderable        :      Boolean);
function Get_Reorderable
  (Icon_View          : access Gtk_Icon_View_Record)
   return Boolean;
```

This function is a convenience function to allow you to reorder models that support the Gtk_Tree_Drag_Source interface and the Gtk_Tree_Drag_Dest interface. Both Gtk_Tree_Store and Gtk_List_Store support these. If Reorderable is True, then the user can reorder the model by dragging and dropping rows. The developer can listen to these changes by connecting to the model's row_inserted and row_deleted signals.

This function does not give you any degree of control over the order – any reordering is allowed. If more control is needed, you should probably handle drag and drop manually.

```
procedure Set_Row_Spacing
  (Icon_View          : access Gtk_Icon_View_Record;
   Row_Spacing        :      Glib.Gint);
function Get_Row_Spacing
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::row-spacing property which specifies the space which is inserted between the rows of the icon view.

```
procedure Set_Spacing
  (Icon_View          : access Gtk_Icon_View_Record;
   Spacing            :      Glib.Gint);
function Get_Spacing
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the ::spacing property which specifies the space which is inserted between the cells (i.e. the icon and the text) of an item.

```
procedure Item_Activated
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Activates the item determined by Path.

129.3.1 Scrolling

```
procedure Get_Visible_Range
  (Icon_View          : access Gtk_Icon_View_Record;
   Start_Path         : out    Gtk.Tree_Model.Gtk_Tree_Path;
   End_Path           : out    Gtk.Tree_Model.Gtk_Tree_Path);
```

Sets Start_Path and End_Path to be the first and last visible path.
Note that there may be invisible paths in between. Both paths should be freed with Path_Free after use.

```
procedure Scroll_To_Path
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Use_Align          :      Boolean := True;
   Row_Align          :      Glib.Gfloat := 0.5;
   Col_Align          :      Glib.Gfloat := 0.0);
```

Moves the alignments of Icon_View to the position specified by Path.
Row_Align determines where the row is placed, and Col_Align determines where column is placed. Both are expected to be between 0.0 and 1.0. 0.0 means left/top alignment, 1.0 means right/bottom alignment, 0.5 means center.

If Use_Align is False, then the alignment arguments are ignored, and the tree does the minimum amount of work to scroll the item onto the screen. This means that the item will be scrolled to the edge closest to its current position. If the item is currently visible on the screen, nothing is done.

This function only works if the model is set, and Path is a valid row on the model. If the model changes before the Icon_View is realized, the centered path will be modified to reflect this change.

129.3.2 Tree Model

```
procedure Set_Model
  (Icon_View          : access Gtk_Icon_View_Record;
   Model              :      Gtk.Tree_Model.Gtk_Tree_Model
                      := null);
function Get_Model
  (Icon_View          : access Gtk_Icon_View_Record)
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Sets the model for a Gtk_Icon_View. If the Icon_View already has a
model set, it will remove it before setting the new model. If Model is null, then it will unset the old model.

```
procedure Set_Text_Column
  (Icon_View          : access Gtk_Icon_View_Record;
   Column             :      Glib.Gint);
function Get_Text_Column
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the column with text for Icon_View to be Column. The text
column must be of type GType_String.

```
procedure Set_Pixbuf_Column
  (Icon_View          : access Gtk_Icon_View_Record;
   Column             :      Glib.Gint);
function Get_Pixbuf_Column
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the column with pixbufs for Icon_View to be Column. The pixbuf column must be of type Gdk.Pixbuf.Get_Type

```
procedure Set_Markup_Column
  (Icon_View          : access Gtk_Icon_View_Record;
   Column             :      Glib.Gint);
function Get_Markup_Column
  (Icon_View          : access Gtk_Icon_View_Record)
   return Glib.Gint;
```

Sets the column with markup information for Icon_View to be Column. The markup column must be of type GType_String. If the markup column is set to something, it overrides the text column set by Set_Text_Column.

```
function Get_Path_At_Pos
  (Icon_View          : access Gtk_Icon_View_Record;
   X                  :      Glib.Gint;
   Y                  :      Glib.Gint)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Finds the path at the point (X, Y), relative to widget coordinates. See Get_Item_At_Pos, if you are also interested in the cell at the specified position.

```
procedure Get_Item_At_Pos
  (Icon_View          : access Gtk_Icon_View_Record;
   X                  :      Gint;
   Y                  :      Gint;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Cell               : out   Gtk.Cell_Renderer.Gtk_Cell_Renderer;
   Has_Item           : out   Boolean);
```

Finds the path at the point (X, Y), relative to widget coordinates. In contrast to Get_Path_At_Pos, this function also obtains the cell at the specified position. The returned path should be freed with Path_Free. Has_Item is set to True if an item exists at the specified position.

129.3.3 Selection

```
procedure Set_Selection_Mode
  (Icon_View          : access Gtk_Icon_View_Record;
   Mode               :      Gtk.Enums.Gtk_Selection_Mode);
function Get_Selection_Mode
  (Icon_View          : access Gtk_Icon_View_Record)
   return Gtk.Enums.Gtk_Selection_Mode;
```

Sets the selection mode of the Icon_View.

```
procedure Select_All
  (Icon_View          : access Gtk_Icon_View_Record);
procedure Unselect_All
  (Icon_View          : access Gtk_Icon_View_Record);
```

Selects all the icons. Icon_View must has its selection mode set to Selection_Multiple

```
procedure Select_Path
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
procedure Unselect_Path
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Selects the row at Path.

```
function Get_Selected_Items
  (Icon_View          : access Gtk_Icon_View_Record)
   return Gtk.Tree_Model.Gtk_Tree_Path_List.Glist;
```

Creates a list of paths of all selected items. Additionally, if you are planning on modifying the model after calling this function, you may want to convert the returned list into a list of Gtk_Tree_Row_Reference. To free the returend value, use: Foreach (List, Gtk.Tree_Model.Path_Free, Null); Free (List);

```
function Path_Is_Selected
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Returns True if the icon pointed to by Path is currently selected. If Path does not point to a valid location, False is returned.

129.3.4 Drag and drop

```
function Create_Drag_Icon
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Gdk.Gdk_Pixmap;
```

Creates a Gdk_Pixmap representation of the item at Path. This image is used for a drag icon. The returned value must be Unref'd by the caller.

```
procedure Enable_Model_Drag_Dest
  (Icon_View          : access Gtk_Icon_View_Record;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Actions            :      Gdk.Dnd.Drag_Action);
procedure Unset_Model_Drag_Dest
  (Icon_View          : access Gtk_Icon_View_Record);
```

Turns Icon_view into a drop destination for automatic DND. Targets is the list of targets that the drag will support.

```
procedure Enable_Model_Drag_Source
  (Icon_View          : access Gtk_Icon_View_Record;
   Start_Button_Mask  :      Gdk.Types.Gdk_Modifier_Type;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Actions            :      Gdk.Dnd.Drag_Action);
procedure Unset_Model_Drag_Source
  (Icon_View          : access Gtk_Icon_View_Record);
```

Turns Icon_view into a drag source for automatic DND. Start_Button_Mask is the allowed buttons to start drag.

```
procedure Get_Dest_Item_At_Pos
  (Icon_View          : access Gtk_Icon_View_Record;
   Drag_X             :      Glib.Gint;
   Drag_Y             :      Glib.Gint;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                : out   Gtk_Icon_View_Drop_Position;
```

```
   Has_Item           : out   Boolean);
```

Determines the destination item for a given position.
Return value: whether there is an item at the given position.

```
procedure Set_Drag_Dest_Item
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                :      Gtk_Icon_View_Drop_Position);
procedure Get_Drag_Dest_Item
  (Icon_View          : access Gtk_Icon_View_Record;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                : out   Gtk_Icon_View_Drop_Position);
```

Sets the item that is highlighted for feedback.

129.3.5 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Cell_Layout" @end itemize

```
function "+"
  (Widget             : access Gtk_Icon_View_Record'Class)
   return Gtk.Cell_Layout.Gtk_Cell_Layout;
function "-"
  (Interf             :      Gtk.Cell_Layout.Gtk_Cell_Layout)
   return Gtk_Icon_View;
```

Converts to and from the Gtk_Cell_Layout interface

130 Package Gtk.Image

The Gtk_Image widget displays a graphical image. The image is typically created using Gdk.Image.Gdk_New.

The pixels in a Gtk_Image may be manipulated by the application after creation, as Gtk_Image store the pixel data on the client side. If you wish to store the pixel data on the server side (thus not allowing manipulation of the data after creation) you should use Gtk_Pixmap.

130.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Misc           (Package Gtk.Misc)
         \___ Gtk_Image       (Package Gtk.Image)
```

130.2 Types

```
type Gtk_Image_Type is
        (Image_Empty,
         Image_Pixmap,
         Image_Image,
         Image_Pixbuf,
         Image_Stock,
         Image_Icon_Set,
         Image_Animation);
```

```
type Property_Image_Type is new Image_Type_Properties.Property;
```

130.3 Subprograms

```
procedure Gtk_New
  (Image              : out   Gtk_Image);
procedure Gtk_New
  (Image              : out   Gtk_Image;
   Val                :       Gdk.Image.Gdk_Image;
   Mask               :       Gdk.Bitmap.Gdk_Bitmap);
procedure Gtk_New
  (Image              : out   Gtk_Image;
   Pixmap             :       Gdk.Pixmap.Gdk_Pixmap;
   Mask               :       Gdk.Bitmap.Gdk_Bitmap);
procedure Gtk_New
  (Image              : out   Gtk_Image;
   Filename           :       String);
procedure Gtk_New
  (Image              : out   Gtk_Image;
   Pixbuf             :       Gdk.Pixbuf.Gdk_Pixbuf);
```

```
procedure Gtk_New
  (Image              : out   Gtk_Image;
   Stock_Id           :       String;
   Size               :       Gtk.Enums.Gtk_Icon_Size);

procedure Gtk_New
  (Image              : out   Gtk_Image;
   Icon_Set           :       Gtk.Icon_Factory.Gtk_Icon_Set;
   Size               :       Gtk.Enums.Gtk_Icon_Size);

procedure Gtk_New
  (Image              : out   Gtk_Image;
   Animation          :       Gdk.Pixbuf.Gdk_Pixbuf_Animation);

procedure Gtk_New_From_Icon_Name
  (Image              : out   Gtk_Image;
   Icon_Name          :       String;
   Size               :       Gtk.Enums.Gtk_Icon_Size);

function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Image.

```
procedure Set
  (Image              : access Gtk_Image_Record;
   Pixmap             :       Gdk.Pixmap.Gdk_Pixmap;
   Mask               :       Gdk.Bitmap.Gdk_Bitmap);

procedure Get
  (Image              : access Gtk_Image_Record;
   Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap);
```

Set or Get the values of a Gtk_Image.
Mask indicates which parts of the image should be transparent.

```
procedure Set
  (Image              : access Gtk_Image_Record;
   Val                :       Gdk.Image.Gdk_Image;
   Mask               :       Gdk.Bitmap.Gdk_Bitmap);

procedure Get
  (Image              : access Gtk_Image_Record;
   Val                : out   Gdk.Image.Gdk_Image;
   Mask               : out   Gdk.Bitmap.Gdk_Bitmap);
```

Set or Get the value of a Gtk_Image.
Mask indicates which parts of the image should be transparent.

```
procedure Set
  (Image              : access Gtk_Image_Record;
   File               :       String);

procedure Set
  (Image              : access Gtk_Image_Record;
   Pixbuf             :       Gdk.Pixbuf.Gdk_Pixbuf);

function Get
  (Image              : access Gtk_Image_Record)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Set or get the image stored in Image

```
procedure Set
  (Image              : access Gtk_Image_Record;
   Stock_Id           :       String;
   Size               :       Gtk.Enums.Gtk_Icon_Size);
```

```
function Get
  (Image              : access Gtk_Image_Record;
   Size               : access Gtk.Enums.Gtk_Icon_Size)
   return String;
```

Set or get the image stored in Image

```
procedure Set
  (Image              : access Gtk_Image_Record;
   Icon_Set           :      Gtk.Icon_Factory.Gtk_Icon_Set;
   Size               :      Gtk.Enums.Gtk_Icon_Size);
procedure Get
  (Image              : access Gtk_Image_Record;
   Icon_Set           : out   Gtk.Icon_Factory.Gtk_Icon_Set;
   Size               : out   Gtk.Enums.Gtk_Icon_Size);
```

Set or get the image stored in Image

```
procedure Set
  (Image              : access Gtk_Image_Record;
   Animation          :      Gdk.Pixbuf.Gdk_Pixbuf_Animation);
function Get
  (Image              : access Gtk_Image_Record)
   return Gdk.Pixbuf.Gdk_Pixbuf_Animation;
```

Get the Pixbuf Animation being displayed by the given Image. The
reference counter for the returned animation is not incremented. This must be done separately if needed.

```
function Get_Storage_Type
  (Image              : access Gtk_Image_Record)
   return Gtk_Image_Type;
```

Indicates how the image was created

```
procedure Clear
  (Image              : access Gtk_Image_Record);
```

Resets the image to be empty.

```
procedure Set_From_Icon_Name
  (Image              : access Gtk_Image_Record;
   Icon_Name          :      String;
   Size               :      Gtk.Enums.Gtk_Icon_Size);
procedure Get_Icon_Name
  (Image              : access Gtk_Image_Record;
   Name               : out   GNAT.Strings.String_Access;
   Size               : out   Gtk.Enums.Gtk_Icon_Size);
```

Gets the icon name and size being displayed by the image
The storage type of the image must be Image_Empty or Image_Icon_Name. The returned string must be freed by the caller.

```
procedure Set_Pixel_Size
  (Image              : access Gtk_Image_Record;
   Pixel_Size         :      Gint);
function Get_Pixel_Size
  (Image              : access Gtk_Image_Record)
   return Gint;
```

Sets or Gets the pixel size used for named icons.
If the pixel size is set to a value different from -1, it is used instead of the icon size set by Set_From_Icon_Name.

131 Package Gtk.Image_Menu_Item

This widget works like a normal menu_item, but you can insert a arbitrary widget (most often a pixmap widget), which is displayed at the left side. The advantage is that indentation is handled the same way as GtkAda does (i.e if you create a menu with a Gtk_Check_Menu_Item, all normal menu_items are automatically indented by GtkAda - so if you use a normal menu_item to display pixmaps at the left side, the pixmaps will be indented, which is not what you want. This widget solves the problem).

131.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Bin          (Package Gtk.Bin)
            \___ Gtk_Item      (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
                  \___ Gtk_Image_Menu_Item (Package Gtk.Image_Menu_Item)
```

131.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Image_Menu_Item;
   Label              :      UTF8_String);
```

Create a new Gtk_Image_Menu_Item.
If label is non null, set the label of the menu item.

```
procedure Gtk_New_From_Stock
  (Widget             : out   Gtk_Image_Menu_Item;
   Stock_Id           :      String);
```

Create a new Gtk_Image_Menu_Item from a stock item.

```
procedure Gtk_New
  (Widget             : out   Gtk_Image_Menu_Item;
   Stock_Id           :      String;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group);
```

Create a new Gtk_Image_Menu_Item with a label.
If label contains an underscore, a mnemonic is created accordingly.

```
procedure Gtk_New_With_Mnemonic
  (Widget             : out   Gtk_Image_Menu_Item;
   Label              :      UTF8_String);
```

Create a new Gtk_Image_Menu_Item with a label.
If label contains an underscore, a mnemonic is created accordingly.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

```
procedure Set_Image
  (Menu_Item          : access Gtk_Image_Menu_Item_Record;
   Image              : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Image
  (Menu_Item          : access Gtk_Image_Menu_Item_Record)
   return Gtk.Widget.Gtk_Widget;
```

132 Package Gtk.Invisible

This widget is used internally by gtk+, and is likely not very useful to end-users. This is a widget that has no visual rendering. It is used for reliable pointer grabs and drag-and-drop

132.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Invisible      (Package Gtk.Invisible)
```

132.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Invisible);
function Get_Type             return Gtk.Gtk_Type;
```

133 Package Gtk.Item

This package declares an abstract type, parent of several widgets in GtkAda.

133.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Item     (Package Gtk.Item)
```

133.2 Signals

- **"deselect"**

  ```
  procedure Handler (Item : access Gtk_Item_Record'Class);
  ```

  Emitted when the item is deselected

- **"select"**

  ```
  procedure Handler (Item : access Gtk_Item_Record'Class);
  ```

  Emitted when the item is selected

- **"toggle"**

  ```
  procedure Handler (Item : access Gtk_Item_Record'Class);
  ```

  Emitted when the item is toggled

133.3 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Item.

```
procedure Item_Select
  (Item               : access Gtk_Item_Record);
```

Emits the "select" signal on Item

```
procedure Item_Deselect
  (Item               : access Gtk_Item_Record);
```

Emits the "deselect" signal on item

```
procedure Toggle
  (Item               : access Gtk_Item_Record);
```

Emits the "toggle" signal on item

134 Package Gtk.Item_Factory

In recent versions of gtk+, this package has been deprecated in favor of Gtk.UIManager. However, the subprograms have not been marked as such in the C files themselves, so are still available in GtkAda as well.

134.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Item_Factory      (Package Gtk.Item_Factory)
```

135 Package Gtk.Label

A Gtk_Label is a light widget associated with some text you want to display on the screen. You can change the text dynamically if needed.

The text can be on multiple lines if you separate each line with the ASCII.LF character. However, this is not the recommended way to display long texts (see the Gtk_Text widget instead).

Mnemonics ========== Labels may contain mnemonics. Mnemonics are underlined characters in the label, used for keyboard navigation. Mnemonics are created by providing string with an underscore before the mnemonic character, such as "_File", to the functions gtk_new_with_mnemonic or set_text_with_mnemonic().

Mnemonics automatically activate any activatable widget the label is inside, such as a Gtk_Button; if the label is not inside the mnemonic's target widget, you have to tell the label about the target using set_mnemonic_widget(). For instance: declare Button : Gtk_Button; Label : Gtk_Label; begin Gtk_New (Button); Gtk_New_With_Mnemonic (Label, "_File"); Add (Button, Label); end; However, there exists a convenience function in Gtk.Button to create such a button already.

Markup ====== To make it easy to format text in a label (changing colors, fonts, etc.), label text can be provided in a simple markup format. Here's how to create a label with a small font: Gtk_New (Label, "<small>hello</small>");

The markup must be valid, and <>& characters must be escaped with < > and &

Markup strings are just a convenient way to set the Pango_Attr_List on label; Set_Attributes() may be a simpler way to set attributes in some cases. Be careful though; Pango_Attr_List tends to cause internationalization problems, unless you're applying attributes to the entire string (i.e. unless you set the range of each attribute to [0, G_MAXINT)). The reason is that specifying the start_index and end_index for a Pango_Attribute requires knowledge of the exact string being displayed, so translations will cause problems.

Selectable labels ================ Labels can be made selectable with Set_Selectable. Selectable labels allow the user to copy the label contents to the clipboard. Only

should be made selectable.

135.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Misc           (Package Gtk.Misc)
         \___ Gtk_Label       (Package Gtk.Label)
```

135.2 Signals

- **"copy_clipboard"**

  ```
  procedure Handler (Label : access Gtk_Label_Record'Class);
  ```

  Request a copy the label's text into the clipboard. This should be bound to a key.

- **"move_cursor"**

```
procedure Handler
(Label  : access Gtk_Label_Record'Class;
Step   : Gtk_Movement_Step;
Amount : Gint;
Extend_Selection : Boolean);
```

  You should emit this signal to request that the cursor be moved inside the label. This is mostly useful from a keybinding. The cursor is also used as the insertion point when modifying the label

- **"populate_popup"**

```
procedure Handler
(Label : access Gtk_Label_Record'Class;
Menu  : access Gtk_Menu_Record'Class);
```

  ???

135.3 Subprograms

```
procedure Gtk_New
  (Label              : out   Gtk_Label;
   Str                :       UTF8_String := "");
```

```
procedure Gtk_New_With_Mnemonic
  (Label              : out   Gtk_Label;
   Str                :       UTF8_String);
```

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Label.

```
procedure Set_Justify
  (Label              : access Gtk_Label_Record;
   Jtype              :       Enums.Gtk_Justification);
```

```
function Get_Justify
  (Label              : access Gtk_Label_Record)
   return Enums.Gtk_Justification;
```

Set the justification for the label.
The default value is Justify_Center, which means that the text will be centered in the label. Note that this setting has an impact only when the Gtk_Label is larger than the text (its default width is the same as the text) and contains multiple lines. To justify a single line label, you should instead call Set_Alignment and make sure that the label or any surrounding container fills its horizontal allocated space.

```
procedure Set_Line_Wrap
  (Label              : access Gtk_Label_Record;
   Wrap               :       Boolean);
```

```
function Get_Line_Wrap
  (Label              : access Gtk_Label_Record)
   return Boolean;
```

Toggle line wrapping within Label.
if Wrap is True, then Label will break lines if the text is larger then the widget's size. If Wrap is False, then the text is simply cut off.

```
procedure Set_Selectable
  (Label              : access Gtk_Label_Record;
   Selectable         :       Boolean);
```

```
function Get_Selectable
  (Label              : access Gtk_Label_Record)
   return Boolean;
```

Selectable labels allow the user to select text from the label,
for copy-and-paste.

```
procedure Set_Use_Markup
  (Label              : access Gtk_Label_Record;
   Markup             :      Boolean);
function Get_Use_Markup
  (Label              : access Gtk_Label_Record)
   return Boolean;
```

Sets whether the text of the label contains markup in Pango's
text markup language. If Markup is True, then Label will be parsed for markup.

```
procedure Set_Use_Underline
  (Label              : access Gtk_Label_Record;
   Underline          :      Boolean);
function Get_Use_Underline
  (Label              : access Gtk_Label_Record)
   return Boolean;
```

Indicates wether an embedded underline in the label indicates the
mnemonic accelerator key.

```
procedure Set_Angle
  (Label              : access Gtk_Label_Record;
   Angle              :      Gdouble);
function Get_Angle
  (Label              : access Gtk_Label_Record)
   return Gdouble;
```

Sets the angle of rotation for the label. An angle of 90 reads from
from bottom to top, an angle of 270, from top to bottom. The angle setting for the label is
ignored if the label is selectable, wrapped, or ellipsized.

```
procedure Set_Ellipsize
  (Label              : access Gtk_Label_Record;
   Mode               :      Pango.Layout.Pango_Ellipsize_Mode);
function Get_Ellipsize
  (Label              : access Gtk_Label_Record)
   return Pango.Layout.Pango_Ellipsize_Mode;
```

Sets the mode used to ellipsize (add an ellipsis: "...") to the text if
there is not enough space to render the entire string.

```
procedure Set_Text
  (Label              : access Gtk_Label_Record;
   Str                :      UTF8_String);
function Get_Text
  (Label              : access Gtk_Label_Record)
   return UTF8_String;
```

Change the text of the label.
The new text is visible on the screen at once. Note that the underline pattern is not
modified.

```
procedure Set_Label
  (Label              : access Gtk_Label_Record;
   Str                :      String);
```

```
function Get_Label
  (Label              : access Gtk_Label_Record)
   return String;
```

Sets the text of the label. The label is interpreted as
including embedded underlines and/or Pango markup depending on the values of label->use_underline and label->use_markup.

```
function Get_Layout
  (Label              : access Gtk_Label_Record)
   return Pango.Layout.Pango_Layout;
```

Gets the layout used to display the label.
The layout is useful to e.g. convert text positions to pixel positions, in combination with Get_Layout_Offsets(). The returned layout is owned by the label so need not be freed by the caller.

```
procedure Get_Layout_Offsets
  (Label              : access Gtk_Label_Record;
   X, Y               : out   Gint);
```

Obtains the coordinates where the label will draw the layout
representing the text in the label; useful to convert mouse events into coordinates inside the layout, e.g. to take some action if some part of the label is clicked. Of course you will need to create a Gtk_Event_Box to receive the events, and pack the label inside it, since labels are a #GTK_NO_WINDOW widget. Remember when using the layout functions you need to convert to and from pixels using PANGO_PIXELS() or #PANGO_SCALE.

```
procedure Set_Max_Width_Chars
  (Label              : access Gtk_Label_Record;
   N_Chars            :      Gint);
function Get_Max_Width_Chars
  (Label              : access Gtk_Label_Record)
   return Gint;
```

Sets the desired maximum width in characters of Label

```
procedure Set_Width_Chars
  (Label              : access Gtk_Label_Record;
   N_Chars            :      Gint);
function Get_Width_Chars
  (Label              : access Gtk_Label_Record)
   return Gint;
```

Sets the desired width in characters of Label.

```
procedure Set_Single_Line_Mode
  (Label              : access Gtk_Label_Record;
   Single_Line_Mode   :      Boolean);
function Get_Single_Line_Mode
  (Label              : access Gtk_Label_Record)
   return Boolean;
```

Sets whether the label is in single line mode.

```
function Get_Mnemonic_Keyval
  (Label              : access Gtk_Label_Record)
   return Gdk.Types.Gdk_Key_Type;
```

Return the key value of the mnemonic accelerator key indicated by an
embedded underline in the label. If there is no mnemonic set up it returns Gdk.Types.Keysyms.GDK_VoidSymbol.

```
procedure Set_Attributes
  (Label              : access Gtk_Label_Record;
   Attrs              :      Pango.Attributes.Pango_Attr_List);
function Get_Attributes
  (Label              : access Gtk_Label_Record)
   return Pango.Attributes.Pango_Attr_List;
```

Sets a list of attributes to be applied to the label text. These
attributes will be ignored if the use_underline or use_markup properties are set. Get_Attributes does not reflect attributes that come from the label's markup (see Set_Markup). If you want to get the effective attributes for the label, use Pango.Layout.Get_Attribute (Get_Layout (Label)).

```
procedure Set_Text_With_Mnemonic
  (Label              : access Gtk_Label_Record;
   Str                :      UTF8_String);
```

Change the text and mnemonic key of the label.
The new text and mnemonic are visible on the screen at once. The mnemonic key can be used to activate another widget, chosen automatically or explicitely using Set_Mnemonic_Widget.

```
procedure Set_Markup
  (Label              : access Gtk_Label_Record;
   Str                :      UTF8_String);
```

Parses Str which is marked up with the Pango text markup language,
setting the label's text and attribute list based on the parse results.

```
procedure Set_Markup_With_Mnemonic
  (Label              : access Gtk_Label_Record;
   Str                :      UTF8_String);
```

Parse Str which is marked up with the Pango text markup language,
setting the label's text and attribute list based on the parse results. If characters in Str are preceded by an underscore, they are underlined indicating that they represent a mnemonic. The mnemonic key can be used to activate another widget, chosen automatically or explicitely using Set_Mnemonic_Widget.

```
procedure Set_Mnemonic_Widget
  (Label              : access Gtk_Label_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Mnemonic_Widget
  (Label              : access Gtk_Label_Record)
   return Gtk.Widget.Gtk_Widget;
```

If the label has been set so that it has an mnemonic key, the label can
be associated with a widget that is the target of the mnemonic. When the label is inside a widget (like a Gtk_Button or a Gtk_Notebook tab), it is automatically associated with the correct widget, but sometimes (i.e. when the target is a Gtk_Entry next to the label), you need to set it explicitly using this procedure. The target widget will be accelerated by emitting "mnemonic_activate" on it. The default handler for this signal will activate the widget if there are no mnemonic collisions and toggle focus between the colliding widgets otherwise.

```
procedure Select_Region
  (Label              : access Gtk_Label_Record;
   Start_Offset       :      Integer := -1;
   End_Offset         :      Integer := -1);
```

Selects a range of characters in the label, if the label is
selectable. If Start or End are -1, then the end of the label will be substituted.

```
procedure Get_Selection_Bounds
  (Label              : access Gtk_Label_Record;
   First, Last        : out   Gint;
   Has_Selection      : out   Boolean);
```

Gets the selected range of characters in the label, returning True
if there's a selection.

```
procedure Set_Pattern
  (Label              : access Gtk_Label_Record;
   Pattern            :      String);
```

Change the underlines pattern.
Pattern is a simple string made of underscore and space characters, matching the ones in the string. GtkAda will underline every letter that matches an underscore. An empty string disables the underlines. example: If the text is FooBarBaz and the Pattern is "___ ___" then both "Foo" and "Baz" will be underlined, but not "Bar".

136 Package Gtk.Layout

A Gtk_Layout is a widget that can have an almost infinite size, without occupying a lot of memory. Its children can be located anywhere within it, but will only appear on the screen if the visible area of the layout contains them. Just like a Gtk_Viewport, its visible area is indicated by two Gtk_Adjustment widgets, and thus a Gtk_Layout can be put as is in a Gtk_Scrolled_Window. As for Gtk_Fixed containers, the children can be located anywhere in the layout (no automatic organization is done). But, as opposed to Gtk_Fixed widgets, a Gtk_Layout does not try to resize itself to show all its children.

Starting from GtkAda 2.0, you have to call Set_Size and specify the maximum size of the layout, otherwise children added with Put outside the size defined for the layout will never be visible. One way to do this is to systematically call Set_Size before calling Put, and make sure you specify a size big enough for the layout.

136.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Layout      (Package Gtk.Layout)
```

136.2 Signals

- **"set_scroll_adjustments"**

```
procedure Handler (Layout : access Gtk_Layout_Record'Class;
Hadj   : in Gtk.Adjustment.Gtk_Adjustment;
Vadj   : in Gtk.Adjustment.Gtk_Adjustment);
```

Emitted whenever at least one of the adjustment of the layout is changed.

136.3 Subprograms

```
procedure Gtk_New
  (Layout             : out   Gtk_Layout;
   Hadjustment        :       Adjustment.Gtk_Adjustment
                       := null;
   Vadjustment        :       Adjustment.Gtk_Adjustment
                       := null);
```

Create new layout.
You can either give an explicit couple of adjustments, that will indicate the current visible area. If you don't specify any, they will be created automatically by GtkAda, which is the usual way to do. The Layout does not occupy any area on the screen, and you have to explicitly specify one with Set_Size below.

```
function Get_Type                return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Layout.

```
procedure Put
  (Layout             : access Gtk_Layout_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   X                  :       Gint;
   Y                  :       Gint);
```

Insert a new child in the layout.
The child will be displayed on the screen only if at least part of it intersects the visible area of the layout. The layout does not resize itself to automatically show the widget. You also need to call Set_Size, if the size you initially defined is smaller than (X, Y), or the child will never be visible even if the layout is scrolled.

```
procedure Move
  (Layout             : access Gtk_Layout_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   X                  :      Gint;
   Y                  :      Gint);
```

Move a child of the layout.
Nothing is done if Widget is not already a child of Layout.

```
function Get_Bin_Window
  (Widget             : access Gtk_Layout_Record)
   return Gdk.Window.Gdk_Window;
```

Return the window associated with the layout.
You should use this one rather than Gtk.Widget.Get_Window.

```
procedure Set_Size
  (Layout             : access Gtk_Layout_Record;
   Width              :      Guint;
   Height             :      Guint);
procedure Get_Size
  (Layout             : access Gtk_Layout_Record;
   Width              : out    Guint;
   Height             : out    Guint);
```

Specify an absolute size for the layout.
This is not the size on the screen, but the internal size of the widget. The screen's size can be set with Gtk.Widget.Set_Usize.

```
procedure Set_Hadjustment
  (Layout             : access Gtk_Layout_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
function Get_Hadjustment
  (Layout             : access Gtk_Layout_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment that indicate the horizontal visual area
of the layout. You generally do not have to modify the value of this adjustment yourself, since this is done automatically when the layout has been put in a Gtk_Scrolled_Window.

```
procedure Set_Vadjustment
  (Layout             : access Gtk_Layout_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
function Get_Vadjustment
  (Layout             : access Gtk_Layout_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment that indicate the vertical visual area
of the layout. You generally do not have to modify the value of this adjustment yourself, since this is done automatically when the layout has been put in a Gtk_Scrolled_Window.

137 Package Gtk.List_Store

This package implements a specific model to store your data in. Each item in the list will correspond to one line in the tree view. Multiple columns can be displayed for each line.

137.1 Subprograms

```
procedure Gtk_New
  (List_Store         : out   Gtk_List_Store;
   Types              :       GType_Array);
```

Creates a new list store using Types to fill the columns.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

```
procedure Set_Column_Types
  (List_Store         : access Gtk_List_Store_Record;
   Types              :       GType_Array);
procedure Set_Value
  (List_Store         : access Gtk_List_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Glib.Values.GValue);
```

Set the data in the cell specified by Iter and Column.
The type of Value must be convertible to the type of the column.

```
procedure Set
  (Tree_Store         : access Gtk_List_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       UTF8_String);
```

Same as above, for an UTF8 string.

```
procedure Set
  (Tree_Store         : access Gtk_List_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Gint);
```

Same as above, for a Gint.

```
procedure Set
  (Tree_Store         : access Gtk_List_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Gdk.Pixbuf.Gdk_Pixbuf);
```

Same as above for a pixbuf

```
procedure Set
  (Tree_Store         : access Gtk_List_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Boolean);
```

Same as above for a boolean

```
procedure Remove
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter);
```

Remove the given row from the list store.
After being removed, Iter is set to be the next valid row, or invalidated if it pointed to the last row in List_Store.

```
procedure Insert
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Position           :      Gint);
```

Create a new row at Position.
Iter will be changed to point to this new row. If Position is larger than the number of rows on the list, then the new row will be appended to the list. The row will be empty before this function is called. To fill in values, you need to call Set_Value.

```
procedure Insert_Before
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Sibling            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Insert a new row before Sibling.
If Sibling is Null_Iter, then the row will be appended to the end of the list. Iter will be changed to point to this new row. The row will be empty before this function is called. To fill in values, you need to call Set_Value.

```
procedure Insert_After
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Sibling            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Insert a new row after Sibling.
If Sibling is Null_Iter, then the row will be prepended to the beginning of the list. Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value.

```
procedure Insert_With_Valuesv
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Position           :      Glib.Gint;
   Columns            :      Glib.Gint_Array;
   Values             :      Glib.Values.GValue_Array);
```

Creates a new row at Position. Iter will be changed to point to this new
row. If Position is larger than the number of rows on the list, then the new row will be appended to the list. The row will be filled with the values given to this function. Using this function is more efficient that calling Insert and then Set for each column, since that will not emit the rows_reordered signal when the model is sorted.

```
procedure Prepend
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter);
```

Prepend a new row to List_Store.
Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value.

```
procedure Append
  (List_Store         : access Gtk_List_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter);
```

Append a new row to List_Store.
Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value.

```
procedure Clear
  (List_Store          : access Gtk_List_Store_Record);
```

Remove all the rows in List_Store.

```
function Iter_Is_Valid
  (List_Store          : access Gtk_List_Store_Record;
   Iter                :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Boolean;
```

WARNING: This function is slow. Only use it for debugging and/or testing purposes. Checks if the given iter is a valid iter for List_Store.

```
procedure Move_After
  (Store               : access Gtk_List_Store_Record;
   Iter                :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Position            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Moves the row pointed to by Iter to the position after Position. Note that this function only works with unsorted stores. If Position is Null_Iter, Iter will be moved to the start of the list.

```
procedure Move_Before
  (Store               : access Gtk_List_Store_Record;
   Iter                :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Position            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Moves the row pointed to by Iter to the position before Position. Note that this function only works with unsorted stores. If Position is Null_Iter, Iter will be moved to the end of the list.

```
procedure Reorder
  (Store               : access Gtk_List_Store_Record;
   New_Order           :      Glib.Gint_Array);
```

Reorders Store to follow the order indicated by New_order. Note that this function only works with unsorted stores. New_Order is an array of integers mapping the new position of each child to its old position before the re-ordering, i.e. New_Order[newpos] = oldpos

```
procedure Swap
  (Store               : access Gtk_List_Store_Record;
   A                   :      Gtk.Tree_Model.Gtk_Tree_Iter;
   B                   :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Swaps the rwos pointed to by A and B. Note that this function only works with unsorted stores.

137.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Tree_Sortable" This interface allows you to specify your own sort function

@item "Gtk_Tree_Drag_Source" This interface allows this widget to act as a dnd source

@item "Gtk_Tree_Drag_Dest" This interface allows this widget to act as a dnd destination @end itemize

```
function "+"
  (Model               : access Gtk_List_Store_Record'Class)
   return Gtk.Tree_Sortable.Gtk_Tree_Sortable;
```

```
function "-"
  (Sortable           :      Gtk.Tree_Sortable.Gtk_Tree_Sortable)
   return Gtk_List_Store;
```

Converts to and from the Gtk_Tree_Sortable interface

```
function "+"
  (Model              : access Gtk_List_Store_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Source;
function "-"
  (Drag_Source        :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Source)
   return Gtk_List_Store;
```

Converts to and from the Gtk_Tree_Drag_Source interface

```
function "+"
  (Model              : access Gtk_List_Store_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Dest;
function "-"
  (Drag_Dest          :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Dest)
   return Gtk_List_Store;
```

Converts to and from the Gtk_Tree_Drag_Source interface

138 Package Gtk.Main

This package contains top-level subprograms that are used to initialize GtkAda and interact with the main event loop.

It also provides a set of packages to set up idle functions, timeout functions, and functions to be called before and after entering the main loop.

138.1 Types

```
type Init_Function is access procedure
          (Data : System.Address);
```

```
type Key_Snooper_Func is access function
          (Widget : System.Address;
           Event  : Gdk.Event.Gdk_Event_Key;
           Data   : System.Address) return Gboolean;
```

```
type Key_Snooper_Id is new Guint;
```

```
type Quit_Function is access function return Boolean;
```

Type of function that can be called when the main loop exits. It should return False if it should not be called again when another main loop exits.

```
type Quit_Handler_Id is new Guint;
```

registration ID for functions that will be called before the main loop exits.

138.2 Subprograms

138.2.1 Initialization and exit routines

```
procedure Init;
```

Initialize GtkAda's internal structures.
This subprogram should be called before any other one in GtkAda. If GtkAda could not be initialized (no access to the display, etc.), the application exits with an error

```
function Init_Check             return Boolean;
```

Initialize GtkAda's internal structures.
Return False if there was an error (no access to the display, etc.)

```
function Set_Locale             return String;
```

Read and parse the local settings, such as time format, ...
Return the name of the local settings, which can also be set with the environment variable LOCALE

```
procedure Set_Locale;
```

Read and parse the local settings, such as time format, ...

```
procedure Disable_Setlocale;
```

Prevents Init, Init_Check and Parse_Args from automatic calling
Set_Locale (LC_ALL, ""). You would want to use this function if you wanted to set the locale for your program to something other than the user's locale, or if you wanted to set different values for different locale categories.

Most programs should not need to call this function.

```
function Check_Version
  (Required_Major     :      Guint := Gtk.Major_Version;
   Required_Minor     :      Guint := Gtk.Minor_Version;
   Required_Micro     :      Guint := Gtk.Micro_Version)
   return String;
```

Checks that the GTK+ library in use is compatible with the given
version. Generally you would pass in the constants Gtk.Major_Version, Gtk.Minor_Version, Gtk.Micro_Version as the three arguments to this function; that produces a check that the library in use is compatible with the version of GTK+ the application or module was compiled against.

Compatibility is defined by two things: first the version of the running library is newer than the version required_major.required_minor.required_micro. Second the running library must be binary compatible with the version required_major.required_minor.required_micro (same major version.)

This function is primarily for GTK+ modules; the module can call this function to check that it wasn't loaded into an incompatible version of GTK+. However, such a check isn't completely reliable, since the module may be linked against an old version of GTK+ and calling the old version of gtk_check_version(), but still get loaded into an application using a newer version of GTK+.

Return value: %NULL if the GTK+ library is compatible with the given version, or a string describing the version mismatch.

```
function Get_Default_Language  return Pango.Font.Pango_Language;
```

Returns the Pango_Language for the default language currently in
effect. (Note that this can change over the life of an application.) The default language is derived from the current locale. It determines, for example, whether GTK+ uses the right-to-left or left-to-right text direction.

138.2.2 Init and Quit functions

```
procedure Init_Add
  (Func               :      Init_Function;
   Data               :      System.Address);
```

Register a function to be called just before starting a main loop.
This function is called only once, even if a new main loop is started recursively.

```
function Quit_Add
  (Main_Level         :      Guint;
   Func               :      Quit_Function)
   return Quit_Handler_Id;
```

Register a new function to be called when the current main loop exits.
The function will be called once when the current main loop exists. If it returns False, it

will then be deleted from the list of quit functions, and won't be called again next time a main loop is exited. The function will only be called when exiting a main loop at level Main_Level. If Main_Level is 0, the function will be called for the current main_loop.

```
function Quit_Add_Destroy
  (Main_Level         :      Guint;
   Object             : access Gtk.Object.Gtk_Object_Record'Class)
   return Quit_Handler_Id;
```

Ensure that Object is destroyed when exiting the main loop at Main_Level
(or the current main loop level is 0).

```
procedure Quit_Remove
  (Id                 :      Quit_Handler_Id);
```

Remove a Quit Handler, that has been previously set by Quit_Add.

138.2.3 The main loop

```
function Events_Pending        return Boolean;
```

Return True if there are some events waiting in the event queue.

```
procedure Main;
```

Start the main loop, and returns only when the main loop is exited.
This subprogram can be called recursively, to start new internal loops. Each of these loops is exited through a call to Main_Quit. This is the recommended method to use when you want to popup a dialog and wait for the user answer before going any further. Note that this procedure can only be called within a single task.

```
function Main_Level            return Guint;
```

Return the level of the current main loop.
Since there can be nested loops, this returns the depth of the current one, starting from 1 (0 if there is none).

```
procedure Main_Quit;
```

Quit the current main loop.
If this was the last active main loop, no more events will be processed by GtkAda.

```
function Main_Iteration
  (Blocking           :      Boolean := True)
   return Boolean;
```

Do one iteration of the main loop.
Blocking indicates whether GtkAda should wait for an event to be available, or simply exit if there is none. Returns True if no main loop is running (ie Main_Quite was called for the innermost main loop). When doing some heavy calculations in an application, it is recommended that you check from time to time if there are any events pending and process them, so that your application still reacts to events. To do that, you would add a loop like:

while Gtk.Main.Events_Pending loop Dead := Gtk.Main.Main_Iteration; end loop;

```
procedure Do_Event
  (Event              :      Gdk.Event.Gdk_Event);
```

Process Event as if it was in the event queue.
This function should almost never be used in your own application, this is the core function for event processing in GtkAda. The user should not free Event, this is already done by GtkAda.

While you should not call this function directly, you might want to know how exactly events are handled. So here is what this function does with the event: * Compress enter/leave notify events. If the event passed build an enter/leave pair together with the next event (peeked from GDK) both events are thrown away. This is to avoid a backlog of (de-)highlighting widgets crossed by the pointer.

* Find the widget which got the event. If the widget can't be determined the event is thrown away unless it belongs to a INCR transaction. In that case it is passed to gtk_selection_incr_event().

* Then the event is passed on a stack so you can query the currently handled event with gtk_get_current_event().

* The event is sent to a widget. If a grab is active all events for widgets that are not in the container in the grab widget are sent to the latter with a few exceptions:

- Deletion and destruction events are still sent to the event widget for obvious reasons.
- Events which directly relate to the visual representation of the event widget.
- Leave events are delivered to the event widget if there was an enter event delivered to it before without the paired leave event
- Drag events are not redirected because it is unclear what the semantics of that would be.
- Another point of interest might be that all key events are first passed through the key snooper functions if there are any. Read the description of Key_Snooper_Install if you need this feature.

* After finishing the delivery the event is popped from the event stack.

```
procedure Propagate_Event
  (Widget              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Event               :      Gdk.Event.Gdk_Event);
```

Sends an event to a widget, propagating the event to parent widgets
if the event remains unhandled. Events received by GTK+ from GDK normally begin in Do_Event. Depending on the type of event, existence of modal dialogs, grabs, etc., the event may be propagated; if so, this function is used. Propagate_Event calls Gtk.Widget.Event on each widget it decides to send the event to. So Gtk.Widget.Event is the lowest-level function; it simply emits the "event" and possibly an event-specific signal on a widget. Propagate_Event is a bit higher-level, and Do_Event is the highest level.

All that said, you most likely don't want to use any of these functions; synthesizing events is rarely needed. Consider asking on the mailing list for better ways to achieve your goals. For example, use gdk_window_invalidate_rect() or gtk_widget_queue_draw() instead of making up expose events.

```
function Get_Event_Widget
  (Event               :      Gdk.Event.Gdk_Event)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget to which Event applies.

```
function Get_Current_Event     return Gdk.Event.Gdk_Event;
```

Return a copy of the event being processed by gtk+. The returned
value must be freed by the caller. If there is no current event, null is returned.

```
procedure Get_Current_Event_State
  (State              : out   Gdk.Types.Gdk_Modifier_Type;
   Had_Current_Event  : out   Boolean);
```

If there is a current event and it has a state field, place
that state field in State and set Had_Current_Event to True, otherwise to False.

```
function Get_Current_Event_Timereturn Guint32;
```

If there is a current event and it has a timestamp, return that
timestamp, otherwise return Gdk.Types.Current_Time

138.2.4 Keys

```
function Key_Snooper_Install
  (Snooper            :      Key_Snooper_Func;
   Func_Data          :      System.Address)
   return Key_Snooper_Id;
```

Install a new key snooper function, which will get called before events
are delivered normally.

```
procedure Key_Snooper_Remove
  (Snooper_Handler_Id  :      Key_Snooper_Id);
```

Remove the snooper with the given Id

138.2.5 Grab functions

```
procedure Grab_Add
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add a new widget to the grab list.
The widget at the front of this list gets all the events even if it does not have the focus. This feature should be used with care. If you want a whole window to get the events, it is better to use Gtk.Window.Set_Modal instead which does the grabbing and ungrabbing for you. The grab is only done for the application. Events outside the application are still sent to their respective windows.

See also Gtk.Window.Gtk_Window_Group

```
procedure Grab_Remove
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Remove a widget from the grab list.

```
function Grab_Get_Current       return Gtk.Widget.Gtk_Widget;
```

Return the widget that currently has the focus.

139 Package Gtk.Marshallers

This package provides a set of generic packages to easily create some Marshallers. Although this package has been designed to be easily reusable, its primary aim is to simplify the use of callbacks.

Note that most users don't need to understand or even look at this package, since the main functions are also renamed in the Gtk.Handlers package (They are called To_Marshaller). This package is rather complex (generic packages inside generic packages), and thus you should understand correctly how Gtk.Handlers work before looking at this one.

To understand the paradigm used in this package, some definitions are necessary:

A Handler, or Callback, is a subprogram provided by the user. This handler, when attached to a particular object, will be called when certain events happen during the life of this object. All handlers take as a first argument an access to the object they were attached to. Depending on the signal, this handler can also have some extra parameters; most of the time, only one extra parameter will be used. For more information about Handlers, refer to the package Gtk.Handlers, where this notion is explained in more details.

A General_Handler is an access to any Handler. Note that this is a type used internally, most users should *not* be using it. It is publicly declared so that users can create new marshallers that would not be already provided here.

A Handler_Proxy is a subprogram that calls its associated handler with the appropriate arguments (from an array of arguments stored in Glib.Values.GValues)

A Marshaller is the association of a General_Handler and a Handler_Proxy.

This package is divided in four generic packages. Each package has been designed to cover a certain kind of callback by providing the associated marshallers. There are two primary factors that describe a callback, and that decide which marshaller to use: Does the callback have access to some user data? Does the callback return some value?

Depending on that, the appropriate generic package should be chosen. For example, if the callback returns a value, but does not expect user data, then the "Return_Marshallers" package should be used. More details about the usage of each package is provided individually below.

Each of these packages is in turn divided into three generic sub-packages. The organization of these subpackages is always the same : o The type "Handler" is defined. It describes the profile of the Handler covered in this generic package. o a "To_Marshaller" function is provided to build a Marshaller from any Handler. o A "Emit_By_Name" procedure is also provided to allow the user to "emit" a signal. This service is explained in more details in Gtk.Handlers. o A private function "Call" is also defined. This is the actual Handler_Proxy that will be used when creating Marshallers with the "To_Marshaller" service.

Once again, selecting the right generic sub-package depends on the callback. For instance, the first sub-package, always called "Generic_Marshaller", is to be used when the handler has one extra argument which is a simple non-tagged type. More details about the usage of each sub-package is also provided individually.

Although most of the cases are covered by the packages below, some unusual cases may appear. This is the case for example when the callback accepts several extra parameters.

In such cases, two options are available: The first option is to use the "standard" callback mechanism with one parameter, this parameter being an array of arguments that you will parse yourself. The second option is to create a new Marshaller package. This is more interesting if more than one callback will follow the same pattern. The body of this package can be used as a good model to build such new marshallers. See also the example in the GtkAda distribution for how to create your own marshallers.

140 Package Gtk.Menu

This widget implements a drop-down menu. This is basically a simple box that contains a series of Gtk_Menu_Item widgets, on which the user can click to perform actions.

Such a menu is usually part of a Gtk_Menu_Bar (at the top of the window), or activated by clicking on an item in another Gtk_Menu. See Gtk.Option_Menu for another way of displaying menus.

All the menus in GtkAda can be "Tear off" menus, i.e you can detach them from their parent (either a menu bar or another menu) to keep them visible on the screen at all times).

It is worth noting that by default, the user of your application will be able to dynamically modify the shortcuts associated with each menu item. For instance, selecting a menu item and pressing a key will assign this new shortcut to the item, possibly removing the shortcut from any other item it was associated with.

Note that pressing <backspace> will simply remove the shortcut.

This default behavior, somewhat unexpected, can be canceled. There are two ways to control this behavior: you can lock a specific menu item by calling Gtk.Widget.Lock_Accelerators on it. But you can also lock all the menu items at once by calling Gtk.Accel_Group.Lock for all the accelerator groups that were used (the GUI builder gate generally creates a single one), as well as on the group returned by Gtk.Accel_Group.Get_Default, which is the one used for items that don't initially have a shortcut.

140.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Menu_Shell  (Package Gtk.Menu_Shell)
            \___ Gtk_Menu     (Package Gtk.Menu)
```

140.2 Signals

- **"move_scroll"**

  ```
  procedure Handler
  (Menu : access Gtk_Menu_Record'Class;
  Typ  : Gtk_Scroll_Type);
  ```

  Requests that another part of the menu be made visible. Menus that display lots of items might not fit on the screen. When this is the case, gtk+ will insert some scrolling arrows on both ends of the menus and emitting this signal will behave as if the user had clicked on one of these arrows. This signal is mostly useful as a keybinding

140.3 Types

type Gtk_Menu_Detach_Func **is access procedure**

type Gtk_Menu_Position_Func **is access procedure**

140.4 Subprograms

140.4.1 Creating a menu

```
procedure Gtk_New
  (Widget              : out   Gtk_Menu);
```

Create a new empty menu.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Menu.

```
procedure Set_Active
  (Menu                : access Gtk_Menu_Record;
   Index               :      Guint);
function Get_Active
  (Menu                : access Gtk_Menu_Record)
   return Gtk.Menu_Item.Gtk_Menu_Item;
```

Select a specified item in the menu.
You will almost never need this function, it is used internally by Gtk_Option_Menu, for which it is the item that is currently selected. Note that the item is not considered as being pressed by the user when you call Set_Active, and thus no callback is called as a result.

```
procedure Set_Tearoff_State
  (Menu                : access Gtk_Menu_Record;
   Torn_Off            :      Boolean);
function Get_Tearoff_State
  (Menu                : access Gtk_Menu_Record)
   return Boolean;
```

Modify the tearoff status of the menu.
If Torn_Off is False, the menu is displayed as a drop down menu which disappears when the menu is not active. If Torn_Off is True, the menu persists until it is closed or reattached. Note that you can give the user access to this functionality by inserting a Gtk_Tearoff_Menu_Item in the menu.

```
procedure Set_Title
  (Menu                : access Gtk_Menu_Record;
   Title               :      UTF8_String);
function Get_Title
  (Menu                : access Gtk_Menu_Record)
   return UTF8_String;
```

Set the title of the menu.
Title is displayed when the menu is displayed as a tearoff menu in an independent window.

```
procedure Reorder_Child
  (Menu                : access Gtk_Menu_Record;
   Child               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position            :      Gint);
```

Move an existing menu_item within the menu.
Its new position is given by Position, 0 being the first item in the menu. If Child does not exist in the menu, nothing is done.

```
procedure Attach
  (Menu              : access Gtk_Menu_Record;
   Child             : access Gtk.Menu_Item.Gtk_Menu_Item_Record'Class;
   Left_Attach       :      Guint;
   Right_Attach      :      Guint;
   Top_Attach        :      Guint;
   Bottom_Attach     :      Guint);
```

Adds a new #GtkMenuItem to a (table) menu. The number of 'cells' that an item will occupy is specified by left_attach, right_attach, top_attach and bottom_attach. These each represent the leftmost, rightmost, uppermost and lower column and row numbers of the table. (Columns and rows are indexed from zero).

Note that this function is not related to Detach().

Adding items to a standard menu is simply done by calling Add().

140.4.2 Displaying a menu

```
procedure Popup
  (Menu              : access Gtk_Menu_Record;
   Parent_Menu_Shell : in    Gtk.Menu_Shell.Gtk_Menu_Shell
                      := null;
   Parent_Menu_Item  : in    Gtk.Menu_Item.Gtk_Menu_Item
                      := null;
   Func              : in    Gtk_Menu_Position_Func := null;
   Button            : in    Guint := 1;
   Activate_Time     : in    Guint32 := 0);
```

Display a menu on the screen.
This is the function to use to create contextual menus. Most of the time, the parameters can have a null value. Parent_Menu_Shell is the Gtk_Menu_Shell that contains Parent_Menu_Item, i.e. the widget that triggered the display of the menu. Func is a function that returns the coordinates for the menu. If it is null, then a default function that positions the menu at the pointer location is used. Button is the mouse button that was pressed to initiate the event. Activate_Time is the time at which the event occurred (you can get it directly from the Gdk_Event structure).

Note that a variant of this function is given in the generic package User_Menu_Popup.

```
procedure Popup
  (Menu              : access Gtk_Menu_Record'Class;
   Data              : access Data_Type;
   Parent_Menu_Shell : in    Gtk.Menu_Shell.Gtk_Menu_Shell
                      := null;
   Parent_Menu_Item  : in    Gtk.Menu_Item.Gtk_Menu_Item
                      := null;
   Func              : in    Gtk_Menu_Position_Func := null;
   Button            : in    Guint := 1;
   Activate_Time     : in    Guint32 := 0);
```

Same as the Popup function above.
Note that Data is not duplicated, thus you should take care of the memory allocation/deallocation yourself.

```
procedure Popdown
  (Menu              : access Gtk_Menu_Record);
```

Remove the menu from the screen

```
procedure Reposition
```

```
   (Menu               : access Gtk_Menu_Record);
```

Reposition a menu according to its position function.
This function is set when Popup is called.

```
procedure Set_Monitor
   (Menu               : access Gtk_Menu_Record;
    Monitor_Num        :      Gint);
```

Informs GTK+ on which monitor a menu should be popped up.
See gdk_screen_get_monitor_geometry().

This function should be called from a Gtk_Menu_Position_Func if the menu should not appear on the same monitor as the pointer. This information can't be reliably inferred from the coordinates returned by a Gtk_Menu_Position_Func, since, for very long menus, these coordinates may extend beyond the monitor boundaries or even the screen boundaries.

140.4.3 Modifying the accelerators

```
procedure Set_Accel_Group
   (Menu               : access Gtk_Menu_Record;
    Accel              :      Accel_Group.Gtk_Accel_Group);
function Get_Accel_Group
   (Menu               : access Gtk_Menu_Record)
    return Accel_Group.Gtk_Accel_Group;
```

Set the Accel_Group that holds the global accelerators and key bindings
for the menu.

```
procedure Set_Accel_Path
   (Menu               : access Gtk_Menu_Record;
    Accel_Path         :      UTF8_String);
```

Set an accelerator path for this menu from which accelerator paths
for its immediate children, its menu items, can be constructed. The main purpose of this function is to spare the programmer the inconvenience of having to call Gtk.Menu_Item.Set_Accel_Path on each menu item that should support runtime user changable accelerators. Instead, by just calling Gtk.Menu.Set_Accel_Path on their parent, each menu item of this menu, that contains a label describing its purpose, automatically gets an accel path assigned. For example, a menu containing menu items "New" and "Exit", will, after Set_Accel_Path (menu, "<Gnumeric-Sheet>/File"); has been called, assign its items the accel paths: "<Gnumeric-Sheet>/File/New" and "<Gnumeric-Sheet>/File/Exit". Assigning accel paths to menu items then enables the user to change their accelerators at runtime.

140.4.4 Attaching a menu to a widget

```
procedure Attach_To_Widget
   (Menu               : access Gtk_Menu_Record;
    Attach_Widget      : access Gtk.Widget.Gtk_Widget_Record'Class;
    Detacher           :      Gtk_Menu_Detach_Func);
```

Attach a menu to the widget.
When the menu is detached from the widget (for instance when it is destroyed), the procedure Detacher will be called. You will almost never need to use this function, unless you specifically want a call back when a widget becomes unavailable. If Attach_Widget is a menu_item with a single label in it, the name of the window created when Menu is teared-off will be the label in the menu_item.

```
procedure Detach
  (Menu               : access Gtk_Menu_Record);
```

Detach the menu from its widget, and call the Detacher set in Attach_To_Widget.

```
function Get_Attach_Widget
  (Menu               : access Gtk_Menu_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget to which the menu was attached.
If the menu was not attached, this function returns null.

```
function Get_For_Attach_Widget
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gtk.Widget.Widget_List.Glist;
```

Returns a list of the menus which are attached to this widget.
This list is owned by GTK+ and must not be modified.

140.5 Example

```
--  This example shows how you create contextual menus with the third mouse
--  button.

with Gtk.Handlers; use Gtk.Handlers;
with Gtk.Menu; use Gtk.Menu;
with Gdk.Event; use Gdk.Event;
with Glib; use Glib;
with Gtk.Window; use Gtk.Window;
with Gtk.Menu_Item; use Gtk.Menu_Item;
with Gtk.Enums; use Gtk.Enums;
with Gtk.Main; use Gtk.Main;

procedure Contextual is

   package Menu_Cb is new Gtk.Handlers.Return_Callback
     (Widget_Type => Gtk_Menu_Record,  Return_Type => Boolean);

   function Popup_Menu_Handler
     (Menu  : access Gtk_Menu_Record'Class;
      Event : Gdk.Event.Gdk_Event) return Boolean is
   begin
      if Gdk.Event.Get_Event_Type (Event) = Button_Press
        and then Gdk.Event.Get_Button (Event) = 3
      then
         Popup (Menu,
                Button        => Gdk.Event.Get_Button (Event),
                Activate_Time => Gdk.Event.Get_Time (Event));
      end if;

      return False;
```

```
   end Popup_Menu_Handler;


   Menu  : Gtk_Menu;
   Win   : Gtk_Window;
   Item  : Gtk_Menu_Item;
begin
   Gtk.Main.Init;

   --  create the menu as usual
   Gtk_New (Menu);
   Gtk_New (Item, "Item1");
   Append (Menu, Item);
   Show (Item);

   --  create the widget on which you want a contextual menu
   --  Prepares it to receive button_press events
   Gtk_New (Win, Window_Toplevel);
   Set_Events (Win, Button_Press_Mask);

   --  Finally, connect both:
   Menu_Cb.Object_Connect
      (Win, "button_press_event",
       Menu_Cb.To_Marshaller (Popup_Menu_Handler'Access),
       Slot_Object => Menu);

   Show_All (Win);
   Gtk.Main.Main;
end Contextual;
```

141 Package Gtk.Menu_Bar

Gtk_Menu_Bar is a subclass of Gtk_Menu_Shell which contains one to many Gtk_Menu_Item. The result is a standard menu bar which can hold many menu items. Gtk_Menu_Bar allows for a shadow type to be set for aesthetic purposes.

141.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Menu_Shell  (Package Gtk.Menu_Shell)
            \___ Gtk_Menu_Bar (Package Gtk.Menu_Bar)
```

141.2 Subprograms

```
procedure Gtk_New
  (Menu_Bar           : out   Gtk_Menu_Bar);
```

Create a menu bar.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Menu_Bar.

```
procedure Set_Child_Pack_Direction
  (Menubar            : access Gtk_Menu_Bar_Record;
   Child_Pack_Dir     :      Gtk.Enums.Gtk_Pack_Direction);
function Get_Child_Pack_Direction
  (Menubar            : access Gtk_Menu_Bar_Record)
   return Gtk.Enums.Gtk_Pack_Direction;
```

Sets how widgets should be packed inside the children of a menubar.

```
procedure Set_Pack_Direction
  (Menubar            : access Gtk_Menu_Bar_Record;
   Pack_Dir           :      Gtk.Enums.Gtk_Pack_Direction);
function Get_Pack_Direction
  (Menubar            : access Gtk_Menu_Bar_Record)
   return Gtk.Enums.Gtk_Pack_Direction;
```

Sets how items should be packed inside a menubar.

142 Package Gtk.Menu_Item

This widget represents one of the lines in a menu, on which the user can click to execute an action. The menu items can be bound to a submenu, so that clicking on them will in fact display the submenu on the screen.

They can also be associated with key shortcuts (called accelerators). See the subprogram Set_Accel_Path, and the subprograms in the package Gtk.Accel_Map.

Activating the proper options in the theme files will allow the user to interactively modify the shortcuts.

142.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Item     (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
```

142.2 Signals

- **"activate"**

  ```
  procedure Handler
  (Menu_Item : access Gtk_Menu_Item_Record'Class);
  ```

  Emitted when the menu item has been activated, ie the user has clicked on it (or use a key shortcut for this)

- **"activate_item"**

  ```
  procedure Handler
  (Menu_Item : access Gtk_Menu_Item_Record'Class);
  ```

  ???

- **"toggle_size_allocate"**

  ```
  procedure Handler
  (Menu_Item  : access Gtk_Menu_Item_Record'Class;
  Allocation : Gtk_Allocation_Request);
  ```

  You should emit this signal to allocate a specific size for the item. In practice, you will not need to do this yourself, since gtk+ takes care of it correctly most of the time.

- **"toggle_size_request"**

  ```
  procedure Handler
  (Menu_Item : access Gtk_Menu_Item_Record'Class;
  Request   : Gtk_Requisition_Access);
  ```

  Query the menu item to ask for its preferred size (this might not be the one actually allocated for it, depending on screen space)

142.3 Subprograms

```
procedure Gtk_New
  (Menu_Item          : out   Gtk_Menu_Item;
   Label              :      UTF8_String := "");
```

```
procedure Gtk_New_With_Mnemonic
  (Menu_Item          : out   Gtk_Menu_Item;
   Label              :       UTF8_String);

function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Menu_Item.

```
procedure Set_Submenu
  (Menu_Item          : access Gtk_Menu_Item_Record;
   Submenu            : access Widget.Gtk_Widget_Record'Class);

function Get_Submenu
  (Menu_Item          : access Gtk_Menu_Item_Record)
   return Gtk.Widget.Gtk_Widget;
```

Set or Get the submenu underneath Menu_Item.

```
procedure Remove_Submenu
  (Menu_Item          : access Gtk_Menu_Item_Record);
```

Remove the menu_item's submenu

```
procedure Set_Right_Justified
  (Menu_Item          : access Gtk_Menu_Item_Record;
   Justify            :       Boolean := True);

function Get_Right_Justified
  (Menu_Item          : access Gtk_Menu_Item_Record)
   return Boolean;
```

Sets whether the menu item appears justified at the right side of a menu bar. This was traditionally done for "Help" menu items, but is now considered a bad idea. (If the widget layout is reversed for a right-to-left language like Hebrew or Arabic, right-justified-menu-items appear at the left.)

```
procedure Set_Accel_Path
  (Menu_Item          : access Gtk_Menu_Item_Record;
   Accel_Path         :       UTF8_String);
```

Set the path that will be used to reference the widget in calls to the subprograms in Gtk.Accel_Map. This means, for instance, that the widget is fully setup for interactive modification of the shortcuts by the user, should he choose to activate this possibility in his themes (see gtk-accel_map.ads for more information).

142.3.1 Signals

```
procedure Gtk_Select
  (Menu_Item          : access Gtk_Menu_Item_Record);
```

Emits the "select" signal on Menu_Item

```
procedure Deselect
  (Menu_Item          : access Gtk_Menu_Item_Record);
```

Emits the "deselect" signal on Menu_Item

```
procedure Activate
  (Menu_Item          : access Gtk_Menu_Item_Record);
```

Emits the "activate" signal on Menu_Item

```
procedure Toggle_Size_Allocate
  (Menu_Item          : access Gtk_Menu_Item_Record;
   Allocation         :       Gtk.Widget.Gtk_Allocation);
```

Emits the "toggle_size_allocate" signal on Menu_Item

```
procedure Toggle_Size_Request
  (Menu_Item          : access Gtk_Menu_Item_Record;
   Requisition        : out   Gtk.Widget.Gtk_Requisition);
```

Emits the "toggle_size_request" signal on Menu_Item

143 Package Gtk.Menu_Shell

This widget is a base class for all menu widgets. It contains a list of items that can be navigated, selected and activated by the user. It can not be instantiated directly.

A menu is considered "active" when it is displayed on the screen, or, in the case of a menu_bar when one of its menus is active.

An item is "selected" if it is displayed in a prelight state and its submenu (if any) displayed.

143.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Menu_Shell  (Package Gtk.Menu_Shell)
```

143.2 Signals

- **"activate_current"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class;
  Force_Hide : Gboolean);
  ```

  Activates the current menu item within the Menu_Shell. if Force_Hide is True, hide the menu afterwards.

- **"cancel"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class);
  ```

  Cancels the selection within the menu_shell. Causes a "selection-done" signal to be emitted.

- **"cycle_focus"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class;
  Direction  : Gtk_Menu_Direction_Type);
  ```

  You should emit this signal to request that another child of Menu_Shell gets the focus. The child is not activated.

- **"deactivate"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class);
  ```

  Emitted when the menu is deactivated, ie is erased from the screen.

- **"move_current"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class;
  Direction  : Gtk_Menu_Direction_Type);
  ```

  You should emit this signal to request that another menu item be selected. It is mostly useful when bound to a keybinding. In a menu, this is bound by default to the arrow keys to move the the selection.

- **"selection-done"**

  ```
  procedure Handler (Menu_Shell : access Gtk_Menu_Shell_Record'Class);
  ```

  Emitted when an item has been selected. The menu shell might not be activated when the signal is emitted.

143.3 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Menu_Shell.

```
procedure Append
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Child              : access Gtk_Menu_Item_Record'Class);
```

Add a new item at the end of the menu.

```
procedure Prepend
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Child              : access Gtk_Menu_Item_Record'Class);
```

Add a new item at the beginning of the menu

```
procedure Insert
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Child              : access Gtk_Menu_Item_Record'Class;
   Position           :      Gint);
```

Add a new item at a specific position in the menu.
The first item is at position 0. To insert as the last item in the menu, set Position to -1.

```
procedure Set_Take_Focus
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Take_Focus         :      Boolean := True);

function Get_Take_Focus
  (Menu_Shell         : access Gtk_Menu_Shell_Record)
   return Boolean;
```

If Take_Focus is TRUE the menu shell will take the keyboard focus so
that it will receive all keyboard events which is needed to enable keyboard navigation in menus.

Setting Take_Focus to FALSE is useful only for special applications like virtual keyboard implementations which should not take keyboard focus.

The Take_Focus state of a menu or menu bar is automatically propagated to submenus whenever a submenu is popped up, so you don't have to worry about recursively setting it for your entire menu hierarchy. Only when programmatically picking a submenu and popping it up manually, the Take_Focus property of the submenu needs to be set explicitely.

Note that setting it to %ALSE has side-effects:

If the focus is in some other app, it keeps the focus and keynav in the menu doesn't work. Consequently, keynav on the menu will only work if the focus is on some toplevel owned by the onscreen keyboard.

To avoid confusing the user, menus with Take_Focus set to FALSE should not display mnemonics or accelerators, since it cannot be guaranteed that they will work.

```
procedure Select_First
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Search_Sensitive   :      Boolean);
```

Select the first visible or selectable child of the menu shell;
don't select tearoff items unless the only item is a tearoff item. If Search_Sensitive is True, search for the first selectable menu item, otherwise select nothing if the first item isn't sensitive. This should be False if the menu is being popped up initially.

143.3.1 Signals emission

```
procedure Deactivate
  (Menu_Shell         : access Gtk_Menu_Shell_Record);
```

Emit the "deactivate" signal.
This deselects the selected item, ungrabs the mouse and keyboard, and erase the Menu_Shell from the screen.

```
procedure Select_Item
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Item               : access Gtk_Menu_Item_Record'Class);
```

Select a new item in the menu, after deselecting the current item.

```
procedure Deselect
  (Menu_Shell         : access Gtk_Menu_Shell_Record);
```

Deselect the currently selected item.

```
procedure Activate_Item
  (Menu_Shell         : access Gtk_Menu_Shell_Record;
   Item               : access Gtk_Menu_Item_Record'Class;
   Force_Deactivate   :      Boolean);
```

Activate the item.
If Force_Deactivate is True or the menu_shell sets this property, Menu_Shell and all its parent menus are deactivated and erased from the screen.

```
procedure Cancel
  (Menu_Shell         : access Gtk_Menu_Shell_Record);
```

Cancels the selection within the menu shell.

144 Package Gtk.Menu_Tool_Button

This package defines a special kind of menu, that can be inserted in a toolbar. This is not something used very often, as in general a toolbar provides a quick access to features that are already accessible in the menu bar itself. In practice, it is used internally by gtk+ itself to implement the overflow menu in the toolbar.

144.1 Signals

- **"show-menu"**

```
procedure Handler (Menu : access Gtk_Menu_Tool_Button_Record'Class);
```

Emitted when the menu is being displayed

144.2 Subprograms

```
procedure Gtk_New
  (Menu               : out   Gtk_Menu_Tool_Button;
   Icon_Widget        :       Gtk.Widget.Gtk_Widget := null;
   Label              :       String := "");
procedure Gtk_New_From_Stock
  (Menu               : out   Gtk_Menu_Tool_Button;
   Stock_Id           :       String);
function Get_Type                return GType;
```

Return the internal type used for this class of widgets

```
procedure Set_Menu
  (Button             : access Gtk_Menu_Tool_Button_Record;
   Menu               : access Gtk.Menu.Gtk_Menu_Record'Class);
function Get_Menu
  (Button             : access Gtk_Menu_Tool_Button_Record)
   return Gtk.Menu.Gtk_Menu;
```

Set or Get the menu that it displayed when the button is clicked on

```
procedure Set_Arrow_Tooltip
  (Button             : access Gtk_Menu_Tool_Button_Record;
   Tooltips           : access Gtk.Tooltips.Gtk_Tooltips_Record'Class;
   Tip_Text           :       String;
   Tip_Private        :       String := "");
```

Set the tooltip set on the arrow button that will display the menu when clicked on.

145 Package Gtk.Message_Dialog

Gtk_Message_Dialog presents a dialog with an image representing the type of message (Error, Question, etc.) alongside some message text. It's simply a convenience widget; you could construct the equivalent of Gtk_Message_Dialog from Gtk_Dialog without too much effort, but Gtk_Message_Dialog saves typing.

The easiest way to do a modal message dialog is to use Gtk.Dialog.Run, though you can also pass in the MODAL flag, Gtk.Dialog.Run automatically makes the dialog modal and waits for the user to respond to it. Gtk.Dialog.Run returns when any dialog button is clicked.

145.1 Types

```
type Gtk_Buttons_Type is
          (Buttons_None,
           Buttons_Ok,
           Buttons_Close,
           Buttons_Cancel,
           Buttons_Yes_No,
           Buttons_Ok_Cancel);
```

```
type Gtk_Message_Type is
          (Message_Info,
           Message_Warning,
           Message_Question,
           Message_Error);
```

145.2 Subprograms

```
procedure Gtk_New
  (Dialog             : out   Gtk_Message_Dialog;
   Parent             :       Gtk.Window.Gtk_Window := null;
   Flags              :       Gtk.Dialog.Gtk_Dialog_Flags
                       := 0;
   Typ                :       Gtk_Message_Type
                       := Message_Info;
   Buttons            :       Gtk_Buttons_Type
                       := Buttons_Close;
   Message            :       String);
procedure Gtk_New_With_Markup
  (Dialog             : out   Gtk_Message_Dialog;
   Parent             :       Gtk.Window.Gtk_Window := null;
   Flags              :       Gtk.Dialog.Gtk_Dialog_Flags
                       := 0;
   Typ                :       Gtk_Message_Type
                       := Message_Info;
   Buttons            :       Gtk_Buttons_Type
                       := Buttons_Close;
```

```
   Message            :      String);
function Get_Type               return GType;
```

Return the internal type used for a Gtk_Message_Dialog

```
procedure Set_Markup
  (Message_Dialog     : access Gtk_Message_Dialog_Record;
   Str                :      String);
```

Sets the text of the message dialog to be Str, which is marked up with the >Pango text markup language. This means that you can for instance <b> to get bold text.

```
procedure Format_Secondary_Markup
  (Message_Dialog     : access Gtk_Message_Dialog_Record;
   Message            :      String);
procedure Format_Secondary_Text
  (Message_Dialog     : access Gtk_Message_Dialog_Record;
   Message            :      String);
```

Sets the secondary text of the message dialog to be Message. When using markup, special marks are interpreted (<b> for bold, <i> for italic,...) Note that setting a secondary text makes the primary text become bold, unless you have provided explicit markup.

146 Package Gtk.Misc

This widget is a base class for all the widgets that require an alignment and padding. This widget can not be instantiated directly.

146.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Misc           (Package Gtk.Misc)
```

146.2 Subprograms

```
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Misc.

```
procedure Set_Alignment
  (Misc               : access Gtk_Misc_Record;
   Xalign             :      Gfloat;
   Yalign             :      Gfloat);
procedure Get_Alignment
  (Misc               : access Gtk_Misc_Record;
   Xalign             : out    Gfloat;
   Yalign             : out    Gfloat);
```

Modify the alignment for the widget.
Xalign and Yalign are both values between 0.0 and 1.0 that specify the alignment: if Xalign is 0.0, the widget will be left aligned; if it is 0.5, the widget will be centered; if it is 1.0 the widget will be right aligned. Yalign is from top (0.0) to bottom (1.0). Both Xalign and Yalign will be constrained to the range 0.0 .. 1.0 Note that if the widget fills its allocated area, these two parameters won't have any effect.

```
procedure Set_Padding
  (Misc               : access Gtk_Misc_Record;
   Xpad               :      Gint;
   Ypad               :      Gint);
procedure Get_Padding
  (Misc               : access Gtk_Misc_Record;
   Xpad               : out    Gint;
   Ypad               : out    Gint);
```

Set the padding (i.e. the extra spaces on the side of the widget).
If Xpad or Ypad is negative, they will be changed to 0.

147 Package Gtk.Notebook

A Gtk_Notebook is a container that displays all of its children at the same location on the screen. They are organized into pages, that can be selected through tabs (either by clicking on them or by a contextual menu). This is the best way to organize complicated interfaces that have a lot of widgets, by putting the children into groups of coherent widgets.

You can hide some of the pages of the notebook by simply calling Hide on the widget that is contained in the page (or returned from Get_Nth_Page).

147.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Notebook    (Package Gtk.Notebook)
```

147.2 Signals

- **"change_current_page"**

  ```
  procedure Handler
  (Notebook : access Gtk_Notebook_Record'Class;
  Offset   : Gint);
  ```

  You should emit this signal to request that the notebook selects another page as the current page. The offset is relative to the currently selected page.

- **"focus_tab"**

  ```
  function Handler
  (Notebook : access Gtk_Notebook_Record'Class;
  Tab      : Gtk_Notebook_Tab) return Boolean;
  ```

  Gives the focus to one of the tabs in the notebook. This signal is mostly used as a keybinding for Home, End,... so that the proper behavior can be implemented

- **"move_focus_out"**

  ```
  procedure Handler
  (Notebook  : access Gtk_Notebook_Record'Class;
  Direction : Gtk_Direction_Type);
  ```

  You should emit this signal to request that the focus be transfered from the current page to the parent widget. Seldom used.

- **"select_page"**

  ```
  function Handler
  (Notebook   : access Gtk_Notebook_Record'Class;
  Move_Focus : Boolean) return Boolean;
  ```

  You should emit this signal to request that the notebook selects the page corresponding to the focus tab. If Move_Focus is true, the page acquires the keyboard focus. Seldom used.

- **"switch_page"**

  ```
  procedure Handler
  (Notebook : access Gtk_Notebook_Record'Class;
  Page     : Gtk_Notebook_Page;
  Page_Num : Guint);
  ```

Notify when the current page is modified in the notebook. This is called every time the user selected a new page, or the program selected a new page with Next_Page, Prev_Page, ...

147.3 Types

```
subtype Gtk_Notebook_Page is Gtk.Gtk_Notebook_Page;
```

```
type Gtk_Notebook_Tab is
          (Notebook_Tab_First,
           Notebook_Tab_Last);
```

```
type Gtk_Notebook_Window_Creation_Func is access function
          (Source : System.Address; --  Gtk_Notebook
           Page   : System.Address; --  Gtk_Widget
           X      : System.Address; --  Gint
           Y      : System.Address; --  Gint
           Data   : System.Address) return Gtk_Notebook;
```

147.4 Subprograms

147.4.1 Creating a notebook and inserting pages

```
procedure Gtk_New
  (Widget             : out   Gtk_Notebook);
```

Create a new empty notebook.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Notebook.

```
procedure Append_Page
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label          : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Insert a new page in Notebook.
The page is put at the end of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. No entry is associated with the page in the contextual menu.

```
procedure Append_Page
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Same as above, but no label is specified.

```
procedure Append_Page_Menu
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label          : access Gtk.Widget.Gtk_Widget_Record'Class;
```

```
   Menu_Label          : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Insert a new page in Notebook.
The page is put at the end of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. A new entry is inserted into the contextual menu. This new entry is made with Menu_Label.

```
procedure Prepend_Page
  (Notebook            : access Gtk_Notebook_Record;
   Child               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label           : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Insert a new page in Notebook.
The page is put at the beginning of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. No entry is associated with the page in the contextual menu.

```
procedure Prepend_Page_Menu
  (Notebook            : access Gtk_Notebook_Record;
   Child               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label           : access Gtk.Widget.Gtk_Widget_Record'Class;
   Menu_Label          : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Insert a new page in Notebook.
The page is put at the beginning of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. A new entry is inserted into the contextual menu. This new entry is made with Menu_Label.

```
procedure Insert_Page
  (Notebook            : access Gtk_Notebook_Record;
   Child               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label           : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position            :      Gint);
```

Insert a new page at a specific position in Notebook.
The page is put at the beginning of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. No entry is associated with the page in the contextual menu. The first position in the list of pages is 0.

```
procedure Insert_Page_Menu
  (Notebook            : access Gtk_Notebook_Record;
   Child               : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label           : access Gtk.Widget.Gtk_Widget_Record'Class;
   Menu_Label          : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position            :      Gint);
```

Insert a new page at a specific position in Notebook.
The page is put at the beginning of the list of pages. The user will select it through a button that contains the Tab_Label widget, which is generally a Gtk_Label, but could be a box with a pixmap in it for instance. A new entry is inserted into the contextual menu. This new entry is made with Menu_Label. The first position in the list of pages is 0.

```
procedure Remove_Page
  (Notebook            : access Gtk_Notebook_Record;
   Page_Num            :      Gint);
```

Remove a page from the notebook.
The first position in the list of pages is 0.

147.4.2 Tabs drag and drop

```
procedure Set_Window_Creation_Hook
  (Func               :      Gtk_Notebook_Window_Creation_Func;
   Data               :      System.Address;
   Destroy            :      Glib.G_Destroy_Notify_Address);
```

Install a global function used to create a window when a detached tab
is dropped in an empty area.

```
procedure Set_Group_Id
  (Notebook           : access Gtk_Notebook_Record;
   Group_Id           :      Gint);
```

Set a group identificator for Notebook. Notebooks sharing
the same group identificator will be able to exchange tabs via drag and drop. A notebook with group identificator -1 will not be able to exchange tabs with any other notebook.

```
function Get_Group_Id
  (Notebook           : access Gtk_Notebook_Record)
   return Gint;
```

Gets the current group identificator for Notebook or -1 if not set.

147.4.3 Modifying and getting the current page

```
function Get_Current_Page
  (Notebook           : access Gtk_Notebook_Record)
   return Gint;
```

Get the number of the current page.
The first page has the number 0.

```
function Get_Nth_Page
  (Widget             : access Gtk_Notebook_Record'Class;
   Page_Num           :      Gint)
   return Gtk.Widget.Gtk_Widget;
```

Convert from a page number to the real page.

```
function Get_N_Pages
  (Notebook           : access Gtk_Notebook_Record)
   return Gint;
```

Return the number of pages in the notebook

```
function Page_Num
  (Widget             : access Gtk_Notebook_Record'Class;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gint;
```

Convert from a child to a page number.
Note that Child is not the notebook page, but the widget you inserted with Insert_Page, Append_Page,...

```
procedure Set_Current_Page
  (Notebook           : access Gtk_Notebook_Record;
   Page_Num           :      Gint := -1);
```

Modify the current page.
The current page is the page that is currently visible on the screen. Nothing happens if

there is no such page. Note also that the page has to be visible on the screen (ie you must have called Gtk.Widget.Show on it first). Use -1 to set the current page to the last one.

Note: This call won't succeeded unless you have called Show on the widget displayed in the page.

```
procedure Set_Page
  (Notebook           : access Gtk_Notebook_Record;
   Page_Num           :      Gint := -1);
procedure Next_Page
  (Notebook           : access Gtk_Notebook_Record);
```

Display the next page on the screen.

```
procedure Prev_Page
  (Notebook           : access Gtk_Notebook_Record);
```

Display the previous page on the screen.

147.4.4 Style and visual aspect

```
procedure Set_Show_Border
  (Notebook           : access Gtk_Notebook_Record;
   Show_Border        :      Boolean := True);
```

Indicate whether the notebook should display borders.
This border gives a 3D aspect to the notebook.

```
function Get_Show_Border
  (Notebook           : access Gtk_Notebook_Record)
   return Boolean;
```

Return whether the notebook displays borders.

```
procedure Set_Show_Tabs
  (Notebook           : access Gtk_Notebook_Record;
   Show_Tabs          :      Boolean := True);
```

Indicate whether the tabs should be displayed.
If the tabs are not displayed, the only way for the user to select a new page is through the contextual menu, and thus you should make sure that the pages were created with the Insert_Page_Menu, ... subprograms.

```
function Get_Show_Tabs
  (Notebook           : access Gtk_Notebook_Record)
   return Boolean;
```

Return whether the tabs are displayed.

```
procedure Set_Tab_Pos
  (Notebook           : access Gtk_Notebook_Record;
   Pos                :      Gtk.Enums.Gtk_Position_Type);
```

Change the position of the tabs.
The tabs can be displayed on any of the four sides of the notebook.

```
function Get_Tab_Pos
  (Widget             : access Gtk_Notebook_Record)
   return Gtk.Enums.Gtk_Position_Type;
```

Return the current position of the tabs.

```
procedure Set_Scrollable
  (Notebook           : access Gtk_Notebook_Record;
   Scrollable         :      Boolean := True);
```

Indicate whether Notebook display scrolling arrows when there are
too many tabs. The default is not to display such scrolling arrows. Note also that a

notebook with too many pages, even if the scrolling is activated, is sometimes hard to use for the user.

```
function Get_Scrollable
  (Notebook           : access Gtk_Notebook_Record)
   return Boolean;
```

Return whether Notebook is scrollable.
See Set_Scrollable for more details.

147.4.5 Popup Menu

The pages of a notebook can be selected both via tabs and a contextual@* menu (right mouse button). Note however that the menu is available only if the pages were inserted with Insert_Page_Menu, Append_Page_Menu or Prepend_Page_Menu.

```
procedure Popup_Enable
  (Notebook           : access Gtk_Notebook_Record);
```

Enable the popup menu.
When the user pressed the right mouse button, a menu is selected that allows him to select a new page.

```
procedure Popup_Disable
  (Notebook           : access Gtk_Notebook_Record);
```

Disable the popup menu.
This menu won't be display any more when the user pressed the right mouse button.

147.4.6 Page properties

```
function Get_Tab_Label
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget displayed in the tab used to select Page.
This widget is in fact the one given in argument to Insert_Page,etc. when the page was created.

```
procedure Set_Tab_Label
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Label          : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Modify the widget displayed in the tab for the page that contains Child.
Tab_Label is generally a Gtk_Label, although it can also be a Gtk_Box that contains a Gtk_Pixmap and a Gtk_Label if you want to show pixmaps.

Note that you will need to call Show_All on Tab_Label: since it is not a Child of the notebook in the sense of Gtk_Container, the Show_All passed to the notebook will not be transmitted to the Tab_Label.

```
procedure Set_Tab_Label_Text
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tab_Text           :      UTF8_String);
```

Modify the text displayed in the tab for the page that contains Child.
This is a less general form of Set_Tab_Label above.

```
function Get_Tab_Label_Text
  (Notebook           : access Gtk_Notebook_Record;
```

```
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return UTF8_String;
```

Return the text displayed in the tab for the page that contains Child.

```
procedure Set_Tab
  (Notebook           : access Gtk_Notebook_Record;
   Page_Num           :      Gint;
   Tab_Label          : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Set Notebook tab widget for a given page number.
This function is mainly intended for use by Gate.

```
function Get_Menu_Label
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget displayed in the contextual menu for the Child.
This is the widget given in argument to Insert_Page_Menu, Append_Page_Menu and Prepend_Page_Menu.

```
procedure Set_Menu_Label
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Menu_Label         : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Modify the widget displayed in the contextual menu for the page
that contains Child.

```
procedure Set_Menu_Label_Text
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Menu_Text          :      UTF8_String);
```

Modify the text displayed in the contextual menu for the page that
contains Child. This is a less general form of Set_Menu_Label above.

```
function Get_Menu_Label_Text
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class)
   return UTF8_String;
```

Return the text displayed in the contextual menu for the page that
contains Child.

```
procedure Query_Tab_Label_Packing
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             : out   Boolean;
   Fill               : out   Boolean;
   Pack_Type          : out   Gtk.Enums.Gtk_Pack_Type);
```

Return the packing used for the tab associated with the page
that contains Child. See the Gtk.Box package for more information on the parameters.

```
procedure Set_Tab_Label_Packing
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Expand             :      Boolean;
   Fill               :      Boolean;
   Pack_Type          :      Gtk.Enums.Gtk_Pack_Type);
```

Modify the packing used for the tab associated with the page that
contains Child.

```
procedure Reorder_Child
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position           :      Gint);
```

Change the position of the page that contains Child.

```
function Get_Tab_Reorderable
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position           :      Gint)
   return Boolean;
```

Get whether the tab can be reordered via drag and drop or not.

```
procedure Set_Tab_Reorderable
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Reorderable        :      Boolean := True);
```

Set whether the notebook tab can be reordered.

```
function Get_Tab_Detachable
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Position           :      Gint)
   return Boolean;
```

Return whether the tab contents can be detached from Notebook.

```
procedure Set_Tab_Detachable
  (Notebook           : access Gtk_Notebook_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Detachable         :      Boolean := True);
```

Set whether the tab can be detached from Notebook to another notebook or widget.

Note that 2 notebooks must share a common group identificator (see Set_Group_Id) to allow automatic tabs interchange between them.

If you want a widget to interact with a notebook through DnD (i.e.: accept dragged tabs from it) it must be set as a drop destination and accept the target "GTK_NOTEBOOK_TAB". The notebook will fill the selection with a Gtk_Widget pointing to the child widget that corresponds to the dropped tab.

If you want a notebook to accept drags from other widgets, you will have to set your own DnD code to do it.

147.4.7 GValue support

```
function Get_Notebook_Page
  (Value              :      Glib.Values.GValue)
   return Gtk_Notebook_Page;
```

Convert a Value into a notebook page.

148 Package Gtk.Object

This is the base class of the widget hierarchy. Everything in GtkAda inherits from this class Gtk_Object, except for a few structures in the Gdk.* packages (low-level drawing routines).

This class provides a set of handful features that you can choose to reuse in your applications:

@itemize @bullet @item Reference counting: an object is not deleted while there exists at least one reference to it. Although GtkAda mostly takes care of that aspect transparently, you might need in some obscure cases to increment or decrement the reference counting for a widget manually, so that it is not removed from memory while you still need it.

@item User data: any number of data can be attached to a Gtk_Object or one of its children. Theses data are referenced by a String, in a hash-table. GtkAda itself uses this feature to provide an easy conversion between C and Ada widgets. Although you might prefer to have a completely object-oriented application (and thus associate data through class inheritance), it might be convenient to directly attach some data to your objects.

@item It also contains the basic structures and subprograms required for signal emission. This is of course used to implement the signal mechanism in GtkAda itself, but can also be used to implement a Model/View/Controller framework.

@end itemize Note that a lot of functions provided in the C interface are not provided here. They are used to emulate an object-oriented language in C, which can of course be done much more conveniently in Ada. Therefore most of these functions are not needed.

Here is a brief explanation on how the reference counting and destruction process work. You should not have to understand all this to use GtkAda, but it might help anyway.

When an object (descendant of Gtk.Object) is created, it has initially a ref_count of 1. A flag is set to say the object is "floating". See the Flags functions in this package for how to retrieve the status of this flag.

When the object gets a parent (ie Gtk.Widget.Set_Parent is called, possibly from other subprograms like Gtk.Container.Add, Gtk.Box.Pack_Start, ...), the ref_count of the object is incremented to 2. If the object was still "floating", it is also "sinked", ie its ref_count is decremented to 1, and the "floating" flag is cleared.

The same behavior as above happens when the object is registered as a top-level widget (i.e. we know it won't have any parent).

Thus the normal life cycle of an object is to have a ref_count to 1, and not be a "floating" object.

When the object is destroyed, the following happens: A temporary reference to the object is created (call to Ref), and ref_count to 2. The object is shutdown: It is removed from its parent (if any), and its ref_count is decremented to 1. The "destroy" signal is emitted, the user's handlers are called, and then all the handlers connected to the object are destroyed. The object is unref-ed. If its ref_count goes down to 0 (normal case), the memory used by the object and its user_data is freed.

148.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
```

148.2 Signals

- **"destroy"**

```
procedure Handler (Object : access Gtk_Object_Record'Class);
```

Raised when the object is about to be destroyed. The "destroyed" flag has been set on the object first. Handlers should not keep a reference on the object. Note that when your destroy handlers are called, the user_data is still available. The default implementation destroys all the handlers.

148.3 Subprograms

```
procedure Sink
  (Object             : access Gtk_Object_Record);
```

Sink the object.
If the object is floating (does not have a parent yet), it is unref-ed once and the floating flag is cleared.

```
procedure Destroy
  (Object             : access Gtk_Object_Record);
```

Destroy the object.
This emits a "destroy" signal, calls all your handlers, and then unconnects them all. The object is then unref-ed, and if its reference count goes down to 0, the memory associated with the object and its user data is freed. Note that when you destroy handlers are called, the user_data is still available.

When a widget is destroyed, it will break any references it holds to other objects. If the widget is inside a container, the widget will be removed from the container. If the widget is a toplevel (derived from Gtk_Window), it will be removed from the list of toplevels, and the reference GTK+ holds to it will be removed. Removing widget from its container or the list of toplevels results in the widget being finalized, unless you've added additional references to the widget with Ref.

In most cases, only toplevel widgets (windows) require explicit destruction, because when you destroy a toplevel its children will be destroyed as well.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Object internally.

```
function Get_Type
  (Object             : access Gtk_Object_Record)
   return Gtk_Type;
```

This function is now obsolete, and is temporarily kept for backward compatibility only. Use Glib.Object.Get_Type instead. ???

148.3.1 Lists

```
function Convert
  (W                  :      Gtk_Object)
   return System.Address;

function Convert
  (W                  :      System.Address)
   return Gtk_Object;
```

148.3.2 Flags

Each object is associated with a set of flags, that reports the state@* of the object. The following flags are known by all objects:

@itemize @bullet @item "Destroyed": Set if the object is marked as destroyed (if its reference count is not yet 0, the memory has not been freed, but you should not use it anyway).

@item "Floating": The object has no parent yet, since it was just created. Its reference count is still 1 (as it was initially). This flag is cleared as soon as Set_Parent is called on the widget or the widget is qualified as a toplevel widget (see Gtk.Container.Register_Toplevel). @end itemize

```
function Flags
  (Object             : access Gtk_Object_Record)
   return Guint32;
```

Return the flags that are set for the object, as a binary mask.

```
procedure Set_Flags
  (Object             : access Gtk_Object_Record;
   Flags              :      Guint32);
```

Set some specific flags for the object.
Flags is a mask that will be added to the current flags of the object.

```
procedure Unset_Flags
  (Object             : access Gtk_Object_Record;
   Flags              :      Guint32);
```

Unset some specific flags for the object.
Flags is a mask that will be deleted from the current flags of the object.

```
function Flag_Is_Set
  (Object             : access Gtk_Object_Record;
   Flag               :      Guint32)
   return Boolean;
```

Return True if the specific flag Flag is set for the object.

```
function In_Destruction_Is_Set
  (Object             : access Gtk_Object_Record'Class)
   return Boolean;
```

Test if the Destroyed flag is set for the object.

```
function Floating_Is_Set
  (Object             : access Gtk_Object_Record'Class)
   return Boolean;
```

Test if the Floating flag is set for the object.

149 Package Gtk.Old_Editable

This widget is an abstract widget designed to support the common functionalities of all widgets for editing text. It provides general services to manipulate an editable widget, a large number of action signals used for key bindings, and several signals that an application can connect to to modify the behavior of a widget.

149.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Old_Editable   (Package Gtk.Old_Editable)
```

149.2 Signals

- **"activate"**

  ```
  procedure Handler (Widget : access Gtk_Old_Editable_Record'Class);
  ```

  Emitted when the user has activated the widget in some fashion.

- **"changed"**

  ```
  procedure Handler (Widget : access Gtk_Old_Editable_Record'Class);
  ```

  emitted when the user has changed the text of the widget.

- **"copy_clipboard"**

  ```
  procedure Handler (Widget : access Gtk_Old_Editable_Record'Class);
  ```

  Emitting this signal will copy the current selection to the clipboard.

- **"cut_clipboard"**

  ```
  procedure Handler (Widget : access Gtk_Old_Editable_Record'Class);
  ```

  Emitting this signal will cut the current selection to the clipboard.

- **"delete_text"**

  ```
  procedure Handler (Widget    : access Gtk_Old_Editable_Record'Class;
  Start_Pos : in Gint;
  End_Pos   : in Gint);
  ```

  Emitted when some text is deleted by the user. As for the "insert-text" handler, it is possible to override the default behavior by connecting a handler to this signal, and then stopping the signal.

- **"insert_text"**

  ```
  procedure Handler (Widget   : access Gtk_Old_Editable_Record'Class;
  Text     : in UTF8_String;
  Length   : in Gint;
  Position : in Gint_Access);
  ```

  Emitted when some text is inserted inside the widget by the user. The default handler inserts the text into the widget. By connecting a handler to this signal, and then by stopping the signal with Gtk.Handlers.Emit_Stop_By_Name, it is possible to modify the inserted text, or even prevent it from being inserted. Position.all should be modified by the callback, and indicates the new position of the cursor after the text has been inserted.

- **"kill_char"**

```
procedure Handler (Widget    : access Gtk_Old_Editable_Record'Class;
Direction : in Gint);
```

Emitting this signal deletes a single character. If Direction is positive, delete forward, else delete backward.

- **"kill_line"**

```
procedure Handler (Widget    : access Gtk_Old_Editable_Record'Class;
Direction : in Gint);
```

Emitting this signal deletes a single line. If Direction is positive, delete forward, otherwise delete backward.

- **"kill_word"**

```
procedure Handler (Widget    : access Gtk_Old_Editable_Record'Class;
Direction : in Gint);
```

Emitting this signal deletes a single word. If Direction is positive, delete forward, otherwise delete backward.

- **"move_cursor"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class;
X, Y   : in Gint);
```

Emitting this signal will move the cursor position for X characters horizontally, and Y characters vertically.

- **"move_page"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class;
X, Y   : in Gint);
```

Emitting this signal will move the cursor for X pages horizontally, and Y pages vertically.

- **"move_to_column"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class;
Column : in Gint);
```

Emitting this signal will move the cursor to the given column.

- **"move_to_row"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class;
Row    : in Gint);
```

Emitting this signal will move the cursor to the given row.

- **"move_word"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class;
N      : in Gint);
```

Emitting this signal will move the cursor by N words (N can be negative).

- **"paste_clipboard"**

```
procedure Handler (Widget : access Gtk_Old_Editable_Record'Class);
```

Emitting this signal will paste the clipboard into the text of the widget at the current cursor position.

- **"set-editable"**

```
procedure Handler (Widget     : access Gtk_Old_Editable_Record'Class;
Is_Editable: in Boolean);
```

Emitting this signal is equivalent to calling Set_Old_Editable.

149.3 Subprograms

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Old_Editable.

```
procedure Changed
  (Editable           : access Gtk_Old_Editable_Record);
```

Cause the "changed" signal to be emitted.

```
procedure Claim_Selection
  (Editable           : access Gtk_Old_Editable_Record;
   Claim              : in    Boolean := True;
   Time               : in    Guint32);
```

If Claim is set to True, claim the ownership of the primary X selection.
Otherwise, release it. "Time" should be set to the time of the last-change time for the specified selection. It is discarded if it is earlier than the current last-change time, or later than the current X server time.

```
procedure Copy_Clipboard
  (Editable           : access Gtk_Old_Editable_Record;
   Time               : in    Guint32);
```

Copy the characters in the current selection to the clipboard.

```
procedure Cut_Clipboard
  (Editable           : access Gtk_Old_Editable_Record;
   Time               :      Guint32);
```

Copy the characters in the current selection to the clipboard.
The selection is then deleted.

```
procedure Delete_Selection
  (Editable           : access Gtk_Old_Editable_Record);
```

Disclaim and delete the current selection.

```
procedure Delete_Text
  (Editable           : access Gtk_Old_Editable_Record;
   Start_Pos          :      Gint := 0;
   End_Pos            :      Gint := -1);
```

Delete the characters from Start_Pos to End_Pos.
If End_Pos is negative, the characters are deleted from Start_Pos to the end of the text.

```
function Get_Chars
  (Editable           : access Gtk_Old_Editable_Record;
   Start_Pos          :      Gint := 0;
   End_Pos            :      Gint := -1)
   return UTF8_String;
```

Get the text from Start_Pos to End_Pos.
If End_Pos is negative, the text from Start_Pos to the end is returned.

```
function Get_Clipboard_Text
  (Widget             : access Gtk_Old_Editable_Record)
   return UTF8_String;
```

Return the last text copied from the clipboard.

```
function Get_Editable
  (Widget             : access Gtk_Old_Editable_Record)
   return Boolean;
```

Return True if the widget is editable by the user.

```
procedure Set_Editable
  (Widget             : access Gtk_Old_Editable_Record;
   Editable           :      Boolean := True);
```

Set the editable status of the entry.
If Editable is False, the user can not modify the contents of the entry. This does not affect the user of the insertion functions above.

```
function Get_Has_Selection
  (Widget             : access Gtk_Old_Editable_Record)
   return Boolean;
```

Return True if the selection is owned by the widget.

```
function Get_Selection_End_Pos
  (Widget             : access Gtk_Old_Editable_Record)
   return Guint;
```

Return the position of the end of the current selection.

```
function Get_Selection_Start_Pos
  (Widget             : access Gtk_Old_Editable_Record)
   return Guint;
```

Return the position of the beginning of the current selection.

```
procedure Insert_Text
  (Editable           : access Gtk_Old_Editable_Record;
   New_Text           :      UTF8_String;
   Position           : in out Gint);
```

Insert the given string at the given position.
Position is set to the new cursor position.

```
procedure Paste_Clipboard
  (Editable           : access Gtk_Old_Editable_Record;
   Time               :      Guint32);
```

The contents of the clipboard is pasted into the given widget at
the current cursor position.

```
procedure Select_Region
  (Editable           : access Gtk_Old_Editable_Record;
   Start              :      Gint;
   The_End            :      Gint := -1);
```

Select the region of text from Start to The_End.
The characters that are selected are those characters at positions from Start up to, but not including The_End. If The_End_Pos is negative, then the characters selected will be those characters from Start to the end of the text.

```
procedure Set_Position
  (Editable           : access Gtk_Old_Editable_Record;
   Position           :      Gint);
```

Change the position of the cursor in the entry.
The cursor is displayed before the character with the given index in the widget (the first character has index 0). The value must be less than or equal to the number of characters in the widget. A value of -1 indicates that the position should be set after the last character in the entry. Note that this position is in characters, not in bytes.

```
function Get_Position
  (Editable           : access Gtk_Old_Editable_Record)
   return Gint;
```

Return the position of the cursor.

150 Package Gtk.Option_Menu

A Gtk_Option_Menu is a widget that allows the user to choose from a list of valid choices. The Gtk_Option_Menu displays the selected choice. When activated, the Gtk_Option_Menu displays a popup Gtk_Menu which allows the user to make a new choice.

150.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Button   (Package Gtk.Button)
               \___ Gtk_Option_Menu (Package Gtk.Option_Menu)
```

150.2 Signals

- **"changed"**

  ```
  procedure Handler (Option : access Gtk_Option_Menu_Record'Class);
  ```

  Emitted when the selected value has changed

150.3 Subprograms

```
procedure Gtk_New
  (Option_Menu        : out   Gtk_Option_Menu);
```

Create a new Gtk_Option_Menu.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Option_Menu.

```
procedure Set_Menu
  (Option_Menu        : access Gtk_Option_Menu_Record;
   Menu               : access Widget.Gtk_Widget_Record'Class);
function Get_Menu
  (Option_Menu        : access Gtk_Option_Menu_Record)
   return Gtk.Menu.Gtk_Menu;
```

Provide the Gtk_Menu that is popped up to allow the user to choose a new value. You should provide a simple menu avoiding the use of tearoff menu items, submenus, and accelerators.

```
procedure Remove_Menu
  (Option_Menu        : access Gtk_Option_Menu_Record;
   Menu               : access Widget.Gtk_Widget_Record'Class);
```

Remove the menu from the option menu.

```
procedure Set_History
  (Option_Menu        : access Gtk_Option_Menu_Record;
   Index              :      Gint);
function Get_History
  (Option_Menu        : access Gtk_Option_Menu_Record)
   return Gint;
```

Select the menu item specified by index making it the newly selected value for the option menu.

151 Package Gtk.Paned

A Gtk_Paned is a container that organizes its two children either horizontally or vertically. The initial size allocated to the children depends on the size they request. However, the user has the possibility to interactively move a separation bar between the two to enlarge one of the children, while at the same time shrinking the second one. The bar can be moved by clicking with the mouse on a small cursor displayed in the bar, and then dragging the mouse.

No additional decoration is provided around the children.

Each child has two parameters, Resize and Shrink.

If Shrink is True, then the widget can be made smaller than its requisition size by the user. Set this to False if you want to set a minimum size.

if Resize is True, this means that the child accepts to be resized, and will not require any size. Thus, the size allocated to it will be the total size allocated to the container minus the size requested by the other child. If Resize is False, the child should ask for a specific size, which it will get. The other child will be resized accordingly. If both Child have the same value for Resize (either True or False), then the size allocated to each is a ratio between the size requested by both.

When you use Set_Position with a parameter other than -1, or the user moves the handle to resize the widgets, the behavior of Resize is canceled.

151.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Paned       (Package Gtk.Paned)
```

151.2 Signals

- **"accept_position"**

  ```
  procedure Handler (Paned : access Gtk_Paned_Record'Class);
  ```

  You should emit this signal to request that the current handle position be considered as valid, and the focus moved out of the handle. By default, this is bound to the key <enter>, so that after resizing through the keyboard (see cycle_handle_focus), the user can validate the new position

- **"cancel_position"**

  ```
  procedure Handler (Paned : access Gtk_Paned_Record'Class);
  ```

  Similar to accept_position, but cancel the last resizing operation. This is bound to <esc> by default.

- **"cycle_child_focus"**

  ```
  procedure Handler
  (Paned   : access Gtk_Paned_Record'Class;
  To_Next : Boolean);
  ```

  You should emit this signal to request that the child be moved to the next child of paned. After the last child, the focus is moved to the parent of Paned (if it is a

Gtk_Paned_Record itself),... This signal is mostly intended to be used from a keybinding (by default, gtk+ attaches it to F6 and shift-F6).

- **"cycle_handle_focus"**

```
procedure Handler
(Paned   : access Gtk_Paned_Record'Class;
To_Next : Boolean);
```

  You should emit this signal to request that the focus be given to the handle, so that the user can then resize the children through the keyboard. It it mostly intended to be used from a keybinding. By default, gtk+ attaches it to F8. The children can then be resized with the arrow keys, PageUp, PageDown, Home and End.

- **"move_handle"**

```
procedure Handler
(Paned : access Gtk_Paned_Record'Class;
Typ   : Gtk_Scroll_Type);
```

  You should emit this signal to request a resizing of the children. This is mostly useful when bound to keys, so that when the handle has the focus (cycle_handle_focus), the children can be resized with the arrow keys, PageUp, PageDown, Home and End

- **"toggle_handle_focus"**

```
procedure Handler (Paned : access Gtk_Paned_Record'Class);
```

  Mostly intended to be bound to a keybinding. When called, this removes the keyboard focus from the handle (if it was given through cycle_handle_focus above).

151.3 Types

subtype Gtk_Hpaned **is** Gtk_Paned;

subtype Gtk_Vpaned **is** Gtk_Paned;

151.4 Subprograms

```
procedure Gtk_New_Vpaned
  (Widget             : out   Gtk_Paned);
```

Create a new vertical container.
The children will be displayed one on top of the other.

```
procedure Gtk_New_Hpaned
  (Widget             : out   Gtk_Paned);
```

Create a new horizontal container.
The children will be displayed one besides the other.

```
function Get_Type              return Glib.GType;
function Get_Type_Vpaned       return Glib.GType;
function Get_Type_Hpaned       return Glib.GType;
```

Return the internal value associated with a Gtk_Paned.

```
procedure Add1
  (Paned              : access Gtk_Paned_Record;
   Child              : access Gtk_Widget_Record'Class);
```

Add the first child of the container.
The child will be displayed either in the top or in the left pane, depending on the orientation of the container. This is equivalent to using the Pack1 procedure with its default parameters.

```
procedure Pack1
  (Paned              : access Gtk_Paned_Record;
   Child              : access Gtk_Widget_Record'Class;
   Resize             :      Boolean := False;
   Shrink             :      Boolean := True);
```

Add a child to the top or left pane.
You can not change dynamically the attributes Resize and Shrink. Instead, you have to remove the child from the container, and put it back with the new value of the attributes. You should also first call Gtk.Object.Ref on the child so as to be sure it is not destroyed when you remove it, and Gtk.Object.Unref it at the end. See the example in testgtk/ in the GtkAda distribution.

```
procedure Add2
  (Paned              : access Gtk_Paned_Record;
   Child              : access Gtk_Widget_Record'Class);
```

Add the second child of the container.
It will be displayed in the bottom or right pane, depending on the container's orientation. This is equivalent to using Pack2 with its default parameters.

```
procedure Pack2
  (Paned              : access Gtk_Paned_Record;
   Child              : access Gtk_Widget_Record'Class;
   Resize             :      Boolean := False;
   Shrink             :      Boolean := False);
```

Add a child to the bottom or right pane.

```
procedure Set_Position
  (Paned              : access Gtk_Paned_Record;
   Position           :      Gint);

function Get_Position
  (Paned              : access Gtk_Paned_Record)
   return Gint;
```

Change or get the position of the separator.
If position is negative, the remembered position is forgotten, and the division is recomputed from the requisitions of the children. Position is in fact the size (either vertically or horizontally, depending on the container) set for the first child.

```
function Get_Child1
  (Paned              : access Gtk_Paned_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the child displayed in the top or left pane.

```
function Get_Child2
  (Paned              : access Gtk_Paned_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the child displayed in the bottom or right pane.

```
function Get_Child1_Resize
  (Paned              : access Gtk_Paned_Record)
   return Boolean;
```

Get the value of the resize attribute for the first child.

```
function Get_Child2_Resize
  (Paned              : access Gtk_Paned_Record)
   return Boolean;
```

Get the value of the resize attribute for the second child.

```
function Get_Child1_Shrink
  (Paned              : access Gtk_Paned_Record)
   return Boolean;
```

Get the value of the shrink attribute for the first child.

```
function Get_Child2_Shrink
  (Paned              : access Gtk_Paned_Record)
   return Boolean;
```

Get the value of the shrink attribute for the second child.

152 Package Gtk.Plug

Note that this package is currently not supported under Win32 systems.

Together with Gtk_Socket, Gtk_Plug provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a Gtk_Socket widget and, passes the XID of that widgets window to the other process, which then creates a Gtk_Plug window with that XID. Any widgets contained in the Gtk_Plug then will appear inside the first applications window.

152.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Window   (Package Gtk.Window)
               \___ Gtk_Plug  (Package Gtk.Plug)
```

152.2 Signals

- **"embedded"**

  ```
  procedure Handler (Plug : access Gtk_Plug_Record'Class);
  ```

  Emitted when the plug has been successfully added to a socket.

152.3 Subprograms

```
procedure Gtk_New
  (Plug               : out   Gtk_Plug;
   Socket_Id          :       Guint32);
```

Create a new plug widget inside the Gtk_Socket identified by socket_id.
Socket_Id is the XID of the socket's window.

```
function Get_Id
  (Plug               : access Gtk_Plug_Record)
   return Guint32;
```

Return the low level window id associated with Plug.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Plug.

153 Package Gtk.Progress

This package is deprecated. It now only acts as an abstract base class for other widgets.

153.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Progress       (Package Gtk.Progress)
```

153.2 Subprograms

```
function Get_Type              return Glib.GType;
```

Return the internal value associated with a Gtk_Progress.

```
function Get_Current_Percentage
  (Progress           : access Gtk_Progress_Record)
   return Gdouble;

function Get_Current_Text
  (Progress           : access Gtk_Progress_Record)
   return UTF8_String;

function Get_Percentage_From_Value
  (Progress           : access Gtk_Progress_Record;
   Value              :      Gdouble)
   return Gdouble;

function Get_Text_From_Value
  (Progress           : access Gtk_Progress_Record;
   Value              :      Gdouble)
   return UTF8_String;

function Get_Value
  (Progress           : access Gtk_Progress_Record)
   return Gdouble;

function Get_Adjustment
  (Widget             : access Gtk_Progress_Record)
   return Gtk.Adjustment.Gtk_Adjustment;

procedure Configure
  (Progress           : access Gtk_Progress_Record;
   Value              :      Gdouble;
   Min                :      Gdouble;
   Max                :      Gdouble);

procedure Set_Activity_Mode
  (Progress           : access Gtk_Progress_Record;
   Activity_Mode      :      Boolean);

function Get_Activity_Mode
  (Progress           : access Gtk_Progress_Record)
   return Boolean;

procedure Set_Adjustment
  (Progress           : access Gtk_Progress_Record;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);

procedure Set_Format_String
  (Progress           : access Gtk_Progress_Record;
   Format             :      UTF8_String);
```

```
procedure Set_Percentage
  (Progress           : access Gtk_Progress_Record;
   Percentage         :      Gdouble);
procedure Set_Show_Text
  (Progress           : access Gtk_Progress_Record;
   Show_Text          :      Boolean);
procedure Set_Text_Alignment
  (Progress           : access Gtk_Progress_Record;
   X_Align            :      Gfloat;
   Y_Align            :      Gfloat);
procedure Set_Value
  (Progress           : access Gtk_Progress_Record;
   Value              :      Gdouble);
```

154 Package Gtk.Progress_Bar

The progress bar provides a convenient way of displaying a state of completion for typically lengthy tasks.

154.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Progress        (Package Gtk.Progress)
         \___ Gtk_Progress_Bar (Package Gtk.Progress_Bar)
```

154.2 Types

```
type Gtk_Progress_Bar_Orientation is
        (Progress_Left_To_Right,
         Progress_Right_To_Left,
         Progress_Bottom_To_Top,
         Progress_Top_To_Bottom);
```

154.3 Subprograms

```
procedure Gtk_New
  (Progress_Bar       : out   Gtk_Progress_Bar);
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Progress_Bar.

```
procedure Pulse
  (Progress_Bar       : access Gtk_Progress_Bar_Record);
```

Indicate that some progress is made, but you don't know how much.
Causes the progress bar to enter "activity mode," where a block bounces back and forth. Each call to Pulse causes the block to move by a little bit (the amount of movement per pulse is determined by Set_Pulse_Step).

```
procedure Set_Text
  (Progress_Bar       : access Gtk_Progress_Bar_Record;
   Text               :      UTF8_String);
function Get_Text
  (Progress_Bar       : access Gtk_Progress_Bar_Record)
   return UTF8_String;
```

Causes the given Text to appear superimposed on the progress bar.
Text: a UTF-8 string.

```
procedure Set_Fraction
  (Progress_Bar       : access Gtk_Progress_Bar_Record;
   Fraction           :      Gdouble);
function Get_Fraction
  (Progress_Bar       : access Gtk_Progress_Bar_Record)
   return Gdouble;
```

Cause the progress bar to "fill in" the given fraction of the bar.
The fraction should be between 0.0 and 1.0, inclusive.

```
procedure Set_Pulse_Step
  (Progress_Bar       : access Gtk_Progress_Bar_Record;
   Step               :      Gdouble);
function Get_Pulse_Step
  (Progress_Bar       : access Gtk_Progress_Bar_Record)
   return Gdouble;
```

Set the fraction of total progress bar length to move the bouncing block for each call to Pulse.

```
procedure Set_Orientation
  (Progress_Bar       : access Gtk_Progress_Bar_Record;
   Orientation        :      Gtk_Progress_Bar_Orientation);
function Get_Orientation
  (Progress_Bar       : access Gtk_Progress_Bar_Record)
   return Gtk_Progress_Bar_Orientation;
```

Cause the progress bar to switch to a different orientation (left-to-right, right-to-left, top-to-bottom, or bottom-to-top).

```
procedure Set_Ellipsize
  (Pbar               : access Gtk_Progress_Bar_Record;
   Mode               :      Pango.Layout.Pango_Ellipsize_Mode);
function Get_Ellipsize
  (Pbar               : access Gtk_Progress_Bar_Record)
   return Pango.Layout.Pango_Ellipsize_Mode;
```

Sets the mode used to ellipsize (add an ellipsis: "...") the text if there is not enough space to render the entire string.

155 Package Gtk.Radio_Action

A Gtk_Radio_Action is similar to Gtk_Radio_Menu_Item. A number of radio actions can be linked together so that only one may be active at any one time.

155.1 Signals

- **"changed"**

```
procedure Handler
(Action  : access Gtk_Radio_Action_Record'Class;
Current : access Gtk_Radio_Action_Record'Class);
```

The changed signal is emitted on every member of a radio group when the active member is changed. The signal gets emitted after the activate signals for the previous and current active members. Current is the action that is currently active

155.2 Subprograms

```
procedure Gtk_New
  (Action             : out   Gtk_Radio_Action;
   Name               :       String;
   Label              :       String := "";
   Tooltip            :       String := "";
   Stock_Id           :       String := "";
   Value              :       Gint);
function Get_Type               return GType;
```

Return the internal type used for a Gtk_Radio_Action

```
function Get_Current_Value
  (Action             : access Gtk_Radio_Action_Record)
   return Glib.Gint;
```

Obtains the value property of the currently active member of the group to which Action belongs.

```
procedure Set_Group
  (Action             : access Gtk_Radio_Action_Record;
   Group              :      Gtk.Widget.Widget_SList.GSlist);
function Get_Group
  (Action             : access Gtk_Radio_Action_Record)
   return Gtk.Widget.Widget_SList.GSlist;
```

Returns the list representing the radio group for this object.
Note that the returned list is only valid until the next change to the group.

A common way to set up a group of radio group is the following: Group : GSlist := null; Action : Gtk_Radio_Action; while ... loop Gtk_New (Action, ...); Set_Group (Action, Group); Group := Get_Group (Action); end loop;

156 Package Gtk.Radio_Button

A Gtk_Radio_Button is a simple button that has two states, like a Gtk_Toggle_Button. However, Gtk_Radio_Buttons can be grouped together to get a special behavior: only one button in the group can be active at any given time. Thus, when the user selects one of the buttons from the group, the button that was previously selected is disabled.

The radio buttons always belongs to a group, even if there is only one in this group

156.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Button     (Package Gtk.Button)
               \___ Gtk_Toggle_Button (Package Gtk.Toggle_Button)
                  \___ Gtk_Check_Button (Package Gtk.Check_Button)
                     \___ Gtk_Radio_Button (Package Gtk.Radio_Button)
```

156.2 Signals

- **"group-changed"**

  ```
  procedure Handler (Radio : access Gtk_Radio_Button_Record'Class);
  ```

  This signal is emitted when the group of the button is changed

156.3 Subprograms

```
procedure Gtk_New
  (Radio_Button       : out   Gtk_Radio_Button;
   Group              :       Widget_SList.GSlist
                       := Widget_SList.Null_List;
   Label              :       UTF8_String := "");
```

```
procedure Gtk_New
  (Radio_Button       : out   Gtk_Radio_Button;
   Group              :       Gtk_Radio_Button;
   Label              :       UTF8_String := "");
```

```
procedure Gtk_New_With_Mnemonic
  (Radio_Button       : out   Gtk_Radio_Button;
   Group              :       Widget_SList.GSlist
                       := Widget_SList.Null_List;
   Label              :       UTF8_String);
```

```
procedure Gtk_New_With_Mnemonic
  (Radio_Button       : out   Gtk_Radio_Button;
   Group              :       Gtk_Radio_Button;
   Label              :       UTF8_String);
```

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Radio_Button.

```
procedure Set_Group
  (Radio_Button       : access Gtk_Radio_Button_Record;
   Group              :       Widget_SList.GSlist);
```

```
function Get_Group
  (Radio_Button       : access Gtk_Radio_Button_Record)
   return Widget_SList.GSlist;
```

Modify the group to which the button belongs.
This will not change anything visually. This can be used as an argument to the first version of Gtk_New above, or the list can also be traversed to get all the buttons.

156.4 Example

```
 --  This creates a group of two buttons. Note how the group is initial-
ized.

 declare
    Radio_Button : Gtk_Radio_Button;
 begin
    Gtk_New (Radio_Button,
             Group       => Radio_Button,
             Label       => "First button");
    Gtk_New (Radio_Button,
             Group       => Radio_Button,
             Label       => "Second button");
 end;
```

157 Package Gtk.Radio_Menu_Item

This widget provides a special kind of menu item that represents a radio button. Such a button can be checked or unchecked by the user, but only one button in a group can be selected at any given time, as opposed to a toggle menu item.

157.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Item       (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
                  \___ Gtk_Check_Menu_Item (Package Gtk.Check_Menu_Item)
                     \___ Gtk_Radio_Menu_Item
                             (Package Gtk.Radio_Menu_Item)
```

157.2 Signals

- **"group-changed"**

  ```
  procedure Handler (Item : access Gtk_Radio_Menu_Item_Record'Class);
  ```

  Emitted when the group of Item has been changed.

157.3 Subprograms

```
procedure Gtk_New
  (Radio_Menu_Item    : out   Gtk_Radio_Menu_Item;
   Group              :       Widget_SList.GSlist;
   Label              :       UTF8_String := "");

procedure Gtk_New_With_Mnemonic
  (Radio_Menu_Item    : out   Gtk_Radio_Menu_Item;
   Group              :       Widget_SList.GSlist;
   Label              :       UTF8_String);

procedure Gtk_New_From_Widget
  (Radio_Menu_Item    : out   Gtk_Radio_Menu_Item;
   Group              : access Gtk_Radio_Menu_Item_Record'Class);

procedure Gtk_New_With_Label_From_Widget
  (Radio_Menu_Item    : out   Gtk_Radio_Menu_Item;
   Group              : access Gtk_Radio_Menu_Item_Record'Class;
   Label              :       String);

procedure Gtk_New_With_Mnemonic_From_Widget
  (Radio_Menu_Item    : out   Gtk_Radio_Menu_Item;
   Group              : access Gtk_Radio_Menu_Item_Record'Class;
   Label              :       String);

function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Radio_Menu_Item.

```
procedure Set_Group
  (Radio_Menu_Item    : access Gtk_Radio_Menu_Item_Record;
   Group              :       Widget_SList.GSlist);

function Get_Group
  (Radio_Menu_Item    : access Gtk_Radio_Menu_Item_Record)
```

```
      return Widget_SList.GSlist;
```

Set or Get to which the menu item belongs. Only one menu item in this group can be selected at any given time

```
   function Selected_Button
     (In_Group           :      Widget_SList.GSlist)
      return Natural;
```

Return the button number of the selected button in the group.
Note: This function is not part of Gtk+ itself, but is provided as a convenient function

158 Package Gtk.Radio_Tool_Button

This package defines a special kind of toggle button that can be inserted in a toolbar. Such buttons are usually created in groups, and only one of them can be active at any time

158.1 Subprograms

```
procedure Gtk_New
  (Radio              : out   Gtk_Radio_Tool_Button;
   Group              :       Widget_SList.GSlist
                       := Widget_SList.Null_List);
procedure Gtk_New_From_Stock
  (Radio              : out   Gtk_Radio_Tool_Button;
   Group              :       Widget_SList.GSlist
                       := Widget_SList.Null_List;
   Stock_Id           :       String);
procedure Gtk_New_From_Widget
  (Radio              : out   Gtk_Radio_Tool_Button;
   Group              : access Gtk_Radio_Tool_Button_Record'Class);
procedure Gtk_New_With_Stock_From_Widget
  (Radio              : out   Gtk_Radio_Tool_Button;
   Group              : access Gtk_Radio_Tool_Button_Record'Class;
   Stock_Id           :       String);
procedure Set_Group
  (Button             : access Gtk_Radio_Tool_Button_Record;
   Group              :       Widget_SList.GSlist);
```

Change the group to which the button belongs

```
function Get_Group
  (Button             : access Gtk_Radio_Tool_Button_Record)
   return Widget_SList.GSlist;
```

Return the group to which the button belongs

```
function Get_Type               return GType;
```

Return the internal type used for this class of widgets

159 Package Gtk.Rc

This package provides an interface to Gtk's configuration files. GTK+ provides resource file mechanism for configuring various aspects of the operation of a GTK+ program at runtime.

Default files ==========

An application can cause GTK+ to parse a specific RC file by calling Gtk.RC.Parse. In addition to this, certain files will be read at the end of Gtk.Main.Init. Unless modified, the files looked for will be <SYSCONFDIR>/gtk-2.0/gtkrc and .gtkrc-2.0 in the users home directory. (<SYSCONFDIR> defaults to /usr/local/etc. It can be changed with the –prefix or –sysconfdir options when configuring GTK+.) Note that although the filenames contain the version number 2.0, all 2.x versions of GTK+ look for these files.

The set of these default files can be retrieved with Gtk.RC.Get_Default_Files and modified with Gtk.RC.Add_Default_File and Gtk.RC.Set_Default_Files. Additionally, the GTK2_RC_FILES environment variable can be set to a G_SEARCHPATH_SEPARATOR_S-separated list of files in order to overwrite the set of default files at runtime.

For each RC file, in addition to the file itself, GTK+ will look for a locale-specific file that will be parsed after the main file. For instance, if LANG is set to ja_JP.ujis, when loading the default file ~/.gtkrc then GTK+ looks for ~/.gtkrc.ja_JP and ~/.gtkrc.ja, and parses the first of those that exists.

Pathnames and patterns ==========

A resource file defines a number of styles and key bindings and attaches them to particular widgets. The attachment is done by the widget, widget_class, and class declarations. As an example of such a statement: widget "mywindow.*.GtkEntry" style "my-entry-class" attaches the style "my-entry-class" to all widgets whose widget class matches the pattern "mywindow.*.GtkEntry".

The patterns here are given in the standard shell glob syntax. The "?" wildcard matches any character, while "*" matches zero or more of any character. The three types of matching are against the widget path, the class path and the class hierarchy. Both the widget and the class paths consists of a "." separated list of all the parents of the widget and the widget itself from outermost to innermost. The difference is that in the widget path, the name assigned by Gtk.Widget.Set_Name is used if present, otherwise the class name of the widget, while for the class path, the class name is always used.

So, if you have a GtkEntry named "myentry", inside of a of a window named "mywindow", then the widget path is: "mwindow.GtkHBox.myentry" while the class path is: "GtkWindow.GtkHBox.GtkEntry".

Matching against class is a little different. The pattern match is done against all class names in the widgets class hierarchy (not the layout hierarchy) in sequence, so the pattern: class "GtkButton" style "my-style" will match not just Gtk_Button widgets, but also Gtk_Toggle_Button and Gtk_Check_Button widgets, since those classes derive from Gtk_Button.

Additionally, a priority can be specified for each pattern, and styles override other styles first by priority, then by pattern type and then by order of specification (later overrides earlier). The priorities that can be specified are (highest to lowest): highest rc theme application gtk lowest

rc is the default for styles read from an RC file, theme is the default for styles read from theme RC files, application should be used for styles an application sets up, and gtk is used for styles that GTK+ creates internally.

Toplevel declarations ====================

An RC file is a text file which is composed of a sequence of declarations. '#' characters delimit comments and the portion of a line after a '#' is ignored when parsing an RC file.

The possible toplevel declarations are: binding name { ... } Declares a binding set. class pattern [style | binding][: priority] name Specifies a style or binding set for a particular branch of the inheritance hierarchy. include filename Parses another file at this point. If filename is not an absolute filename, it is searched in the directories of the currently open RC files. GTK+ also tries to load a locale-specific variant of the included file. module_path path Sets a path (a list of directories separated by colons) that will be searched for theme engines referenced in RC files. pixmap_path path Sets a path (a list of directories separated by colons) that will be searched for pixmaps referenced in RC files. im_module_file pathname Sets the pathname for the IM modules file. Setting this from RC files is deprecated; you should use the environment variable GTK_IM_MODULE_FILE instead. style name [= parent] { ... } Declares a style. widget pattern [style | binding][: priority] name Specifies a style or binding set for a particular group of widgets by matching on the widget pathname. widget_class pattern [style | binding][: priority] name Specifies a style or binding set for a particular group of widgets by matching on the class pathname. setting = value Specifies a value for a setting. Note that settings in RC files are overwritten by system-wide settings which are managed by an XSettings manager. See Gtk.Settings.

Styles ======

A RC style is specified by a style declaration in a RC file, and then bound to widgets with a widget, widget_class, or class declaration. All styles applying to a particular widget are composited together with widget declarations overriding widget_class declarations which, in turn, override class declarations. Within each type of declaration, later declarations override earlier ones.

Within a style declaration, the possible elements are: bg[state] = color Sets the color used for the background of most widgets. fg[state] = color Sets the color used for the foreground of most widgets. base[state] = color Sets the color used for the background of widgets displaying editable text. This color is used for the background of, among others, Gtk_Text, Gtk_Entry, Gtk_List, and Gtk_CList. text[state] = color Sets the color used for foreground of widgets using base for the background color. xthickness = number Sets the xthickness, which is used for various horizontal padding values in GTK+. ythickness = number Sets the ythickness, which is used for various vertical padding values in GTK+. bg_pixmap[state] = pixmap Sets a background pixmap to be used in place of the bg color (or for GtkText, in place of the base color. The special value "<parent>" may be used to indicate that the widget should use the same background pixmap as its parent. The special value "<none>" may be used to indicate no background pixmap. font = font fontset = font Starting with GTK+ 2.0, the "font" and "fontset" declarations are ignored; use "font_name" declarations instead. font_name = font Sets the font for a widget. font must be a Pango font name, e.g. "Sans Italic 10". For details about Pango font names, see Pango.Font.Font_Description_From_String. stock["stock-id"] = { icon source specifications } Defines the icon for a stock item. engine "engine" { engine-specific settings } Defines

the engine to be used when drawing with this style. class::property = value Sets a style property for a widget class.

The colors and background pixmaps are specified as a function of the state of the widget. The states are: NORMAL A color used for a widget in its normal state. ACTIVE A variant of the NORMAL color used when the widget is in the GTK_STATE_ACTIVE state, and also for the trough of a ScrollBar, tabs of a NoteBook other than the current tab and similar areas. Frequently, this should be a darker variant of the NORMAL color. PRELIGHT A color used for widgets in the GTK_STATE_PRELIGHT state. This state is the used for Buttons and MenuItems that have the mouse cursor over them, and for their children. SELECTED A color used to highlight data selected by the user. for instance, the selected items in a list widget, and the selection in an editable widget. INSENSITIVE A color used for the background of widgets that have been set insensitive with Gtk.Widget.Set_Sensitive().

Colors can be specified as a string containing a color name (GTK+ knows all names from the X color database /usr/lib/X11/rgb.txt), in one of the hexadecimal forms #rrrrggggbbbb, #rrrgggbbb, #rrggbb, or #rgb, where r, g and b are hex digits, or they can be specified as a triplet { r, g, b}, where r, g and b are either integers in the range 0-65535 or floats in the range 0.0-1.0.

In a stock definition, icon sources are specified as a 4-tuple of image filename or icon name, text direction, widget state, and size, in that order. Each icon source specifies an image filename or icon name to use with a given direction, state, and size. Filenames are specified as a string such as "itemltr.png", while icon names (looked up in the current icon theme), are specified with a leading @, such as @"item-ltr". The * character can be used as a wildcard, and if direction/state/size are omitted they default to *. So for example, the following specifies different icons to use for left-to-right and right-to-left languages:

stock["my-stock-item"] = { { "itemltr.png", LTR, *, * }, { "itemrtl.png", RTL, *, * }}

This could be abbreviated as follows:

stock["my-stock-item"] = { { "itemltr.png", LTR }, { "itemrtl.png", RTL }}

You can specify custom icons for specific sizes, as follows:

stock["my-stock-item"] = { { "itemmenusize.png", *, *, "gtk-menu" }, { "itemtoolbarsize.png", *, *, "gtk-large-toolbar" } { "itemgeneric.png" }} /* implicit *, *, * as a fallback */

The sizes that come with GTK+ itself are "gtk-menu", "gtk-small-toolbar", "gtk-large-toolbar", "gtk-button", "gtk-dialog". Applications can define other sizes (see also Gtk.Icon_Factory to learn more about this)

It's also possible to use custom icons for a given state, for example:

stock["my-stock-item"] = { { "itemprelight.png", *, PRELIGHT }, { "iteminsensitive.png", *, INSENSITIVE }, { "itemgeneric.png" }} /* implicit *, *, * as a fallback */

When selecting an icon source to use, GTK+ will consider text direction most important, state second, and size third. It will select the best match based on those criteria. If an attribute matches exactly (e.g. you specified PRELIGHT or specified the size), GTK+ won't modify the image; if the attribute matches with a wildcard, GTK+ will scale or modify the image to match the state and size the user requested.

Key bindings ==========

Key bindings allow the user to specify actions to be taken on particular key presses. The form of a binding set declaration is:

binding name { bind key { signalname (param, ...) ... } ... }

key is a string consisting of a series of modifiers followed by the name of a key. The modifiers can be: <alt>, <control>, <mod1>, <mod2>, <mod3>, <mod4>, <mod5> <release>, <shft>, <shift> <shft> is an alias for <shift> and <alt> is an alias for <mod1>.

The action that is bound to the key is a sequence of signal names (strings) followed by parameters for each signal. The signals must be action signals. Each parameter can be a float, integer, string, or unquoted string representing an enumeration value. The types of the parameters specified must match the types of the parameters of the signal.

Binding sets are connected to widgets in the same manner as styles, with one difference: Binding sets override other binding sets first by pattern type, then by priority and then by order of specification. The priorities that can be specified and their default values are the same as for styles.

159.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Rc_Style          (Package Gtk.Rc_Style)
```

159.2 Subprograms

```
procedure Gtk_New
  (Rc_Style           : out   Gtk_Rc_Style);
function Get_Type             return Glib.GType;
```

Return the internal value associated with Gtk_Rc_Style.

```
function Copy
  (Orig               : access Gtk_Rc_Style_Record)
   return Gtk_Rc_Style;
```

Make a copy of the specified Gtk_Rc.Style.
This function will correctly copy an rc style that is a member of a class derived from Gtk_Rc_Style.

```
procedure Add_Default_File
  (Filename           :      String);
```

Add a file to the list of files to be parsed at the end of Gtk.Main.Init

```
procedure Set_Default_Files
  (Filenames          :      Chars_Ptr_Array);
function Get_Default_Files     return Chars_Ptr_Array;
```

Set the list of files that GtkAda will read at the end of Gtk.Main.Init

```
function Get_Style
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gtk_Style;
```

Find all matching RC styles for a given widget, composites them
together, and then create a Gtk_Style representing the composite appearance. (GtkAda actually keeps a cache of previously created styles, so a new style may not be created) Return the resulting style. No refcount is added to the returned style, so if you want to save this style around, you should add a reference yourself.

```
procedure Parse
  (Filename           :      String);
procedure Parse_String
  (Rc_String          :      String);
```

Parse either a file or a string containing a gtk+ configuration (see the description at the top of this package).

```
function Reparse_All           return Boolean;
```

If the modification time on any previously read file for the default Gtk_Settings has changed, discard all style information and then reread all previously read RC files. Return True if the files were reread.

```
function Find_Module_In_Path
  (Module_File        :      String)
   return String;
function Get_Theme_Dir         return String;
```

Returns the standard directory in which themes should be installed. (GTK+ does not actually use this directory itself.)

```
function Get_Module_Dir        return String;
```

Returns a directory in which GTK+ looks for theme engines. This is a dynamic library loaded by gtk+ that will be responsible for drawing parts of the application (ie implement all the functions in Gtk.Style)

```
function Get_Im_Module_Path    return String;
```

Obtains the path in which to look for IM modules. See the documentation of the GTK_PATH environment variable for more details about looking up modules. This function is useful solely for utilities supplied with GTK+ and should not be used by applications under normal circumstances.

```
function Get_Im_Module_File    return String;
```

Obtains the path to the IM modules file. See the documentation of the GTK_IM_MODULE_FILE environment variable for more details.

```
function Reparse_All_For_Settings
  (Settings           : access Gtk.Settings.Gtk_Settings_Record'Class;
   Force_Load         :      Boolean)
   return Boolean;
```

If the modification time on any previously read file for the given Gtk_Settings has changed, discard all style information and then reread all previously read RC files. If Force_Load is true, the files are reloaded even if unmodified. Return True if some files have been reparsed

```
procedure Reset_Styles
  (Settings           : access Gtk.Settings.Gtk_Settings_Record'Class);
```

This function recomputes the styles for all widgets that use a particular Gtk_Settings object. (There is one Gtk_Settings object per Gdk_Screen, see Gtk.Settings.Get_For_Screen); It is useful when some global parameter has changed that affects the appearance of all widgets, because when a widget gets a new style, it will both redraw and recompute any cached information about its appearance. As an example, it is used when the default font size set by the operating system changes. Note that this function doesn't affect widgets that have a style set explicitely on them with Gtk.Widget.Set_Style.

```
function Get_Style_By_Paths
  (Settings           : access Gtk.Settings.Gtk_Settings_Record'Class;
   Widget_Path        :      String := "";
   Class_Path         :      String := "";
   Typ                :      Glib.GType := Glib.GType_None)
   return Gtk.Style.Gtk_Style;
```

Creates up a Gtk_Style from styles defined in a RC file by providing
the raw components used in matching. This function may be useful when creating pseudo-widgets that should be themed like widgets but don't actually have corresponding GTK+ widgets. An example of this would be items inside a GNOME canvas widget. Returns null if nothing matching was found and the default style should be used. You must call Ref if you intend to keep a reference on the style.

159.2.1 Widget related functions

```
procedure Modify_Style
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Style              : access Gtk_Rc_Style_Record'Class);
```

Modifies style values on the widget. Modifications made using this
technique take precedence over style values set via an RC file, however, they will be overriden if a style is explicitely set on the widget using gtk_widget_set_style(). The #GtkRcStyle structure is designed so each field can either be set or unset, so it is possible, using this function, to modify some style values and leave the others unchanged.

Note that modifications made with this function are not cumulative with previous calls to gtk_widget_modify_style() or with such functions as gtk_widget_modify_fg(). If you wish to retain previous values, you must first call gtk_widget_get_modifier_style(), make your modifications to the returned style, then call gtk_widget_modify_style() with that style. On the other hand, if you first call gtk_widget_modify_style(), subsequent calls to such functions gtk_widget_modify_fg() will have a cumulative effect with the initial modifications.

```
function Get_Modifier_Style
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Gtk_Rc_Style;
```

Return the current modifier style for the widget.
(As set by Modify_Style.) If no style has previously set, a new Gtk_Rc_Style will be created with all values unset, and set as the modifier style for the widget. If you make changes to this rc style, you must call Modify_Style, passing in the returned rc style, to make sure that your changes take effect.

Return value: the modifier style for the widget. This rc style is owned by the widget. If you want to keep a pointer to value this around, you must add a refcount using Ref.

160 Package Gtk.Rc_Style

161 Package Gtk.Ruler

This widget is generally put on the sides of a drawing area to help the user measure distances. It indicates the current position of the mouse cursor within the drawing area, and can be graduated in multiple units.

161.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Ruler          (Package Gtk.Ruler)
```

161.2 Types

subtype Gtk_Hruler **is** Gtk_Ruler;

subtype Gtk_Vruler **is** Gtk_Ruler;

161.3 Subprograms

```
procedure Gtk_New_Hruler
  (Ruler              : out   Gtk_Ruler);
procedure Gtk_New_Vruler
  (Ruler              : out   Gtk_Ruler);
function Get_Type               return Gtk.Gtk_Type;
function Hruler_Get_Type        return Gtk.Gtk_Type;
function Vruler_Get_Type        return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Ruler.

```
procedure Set_Metric
  (Ruler              : access Gtk_Ruler_Record;
   Metric             :      Gtk_Metric_Type);
function Get_Metric
  (Ruler              : access Gtk_Ruler_Record)
   return Gtk_Metric_Type;
```

Set or get the units used for a Gtk_Ruler. See Set_Metric.

```
procedure Set_Range
  (Ruler              : access Gtk_Ruler_Record;
   Lower              :      Gdouble;
   Upper              :      Gdouble;
   Position           :      Gdouble;
   Max_Size           :      Gdouble);
procedure Get_Range
  (Ruler              : access Gtk_Ruler_Record;
   Lower              : out   Gdouble;
   Upper              : out   Gdouble;
   Position           : out   Gdouble;
   Max_Size           : out   Gdouble);
```

Retrieve values indicating the range and current position of a Ruler.
See Set_Range. Lower: Lower limit of the ruler. Upper: Upper limit of the ruler. Position: Current position of the mark on the ruler. Max_Size: Maximum size of the ruler used when calculating the space to leave for the text.

```
procedure Draw_Ticks
  (Ruler              : access Gtk_Ruler_Record);
procedure Draw_Pos
  (Ruler              : access Gtk_Ruler_Record);
```

???

162 Package Gtk.Scale

A scale is a horizontal or vertical widget that a user can slide to choose a value in a given range. This is a kind of cursor, similar to what one finds on audio systems to select the volume for instance.

162.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Range          (Package Gtk.Grange)
         \___ Gtk_Scale       (Package Gtk.Scale)
```

162.2 Types

subtype Gtk_Hscale **is** Gtk_Scale;

subtype Gtk_Vscale **is** Gtk_Scale;

162.3 Subprograms

```
procedure Gtk_New_Hscale
  (Scale              : out   Gtk_Scale;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment);
procedure Gtk_New_Hscale
  (Scale              : out   Gtk_Scale;
   Min                :       Gdouble;
   Max                :       Gdouble;
   Step               :       Gdouble);
```

Create a new horizontal scale widget that lets the user input a number between Min and Max with an increment of Step. Step must be non-zero; it is the distance the slider moves when using the arrow keys to adjust the scale value. An adjustment can be used to specify the range instead.

```
procedure Gtk_New_Vscale
  (Scale              : out   Gtk_Scale;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment);
procedure Gtk_New_Vscale
  (Scale              : out   Gtk_Scale;
   Min                :       Gdouble;
   Max                :       Gdouble;
   Step               :       Gdouble);
```

Create a new vertical scale widget that lets the user input a number between Min and Max with an increment of Step. Step must be non-zero; it is the distance the slider moves when using the arrow keys to adjust the scale value. An adjustment can be used to specify the range instead.

```
function Get_Type               return Gtk.Gtk_Type;
```

```
function Hscale_Get_Type       return GType;
function Vscale_Get_Type       return GType;
```

Return the internal value associated with a Gtk_Scale, a Gtk_Hscale or a Gtk_Vscale.

```
procedure Set_Digits
  (Scale              : access Gtk_Scale_Record;
   The_Digits         :      Gint);
function Get_Digits
  (Scale              : access Gtk_Scale_Record)
   return Gint;
```

Sets the number of decimal places that are displayed in the value. Also causes the value of the adjustment to be rounded off to this number of digits, so the retrieved value matches the value the user saw.

```
procedure Set_Draw_Value
  (Scale              : access Gtk_Scale_Record;
   Draw_Value         :      Boolean);
function Get_Draw_Value
  (Scale              : access Gtk_Scale_Record)
   return Boolean;
```

Specifies whether the current value is displayed as a string next to the slider.

```
procedure Set_Value_Pos
  (Scale              : access Gtk_Scale_Record;
   Pos                :      Gtk_Position_Type);
function Get_Value_Pos
  (Scale              : access Gtk_Scale_Record)
   return Gtk_Position_Type;
```

Sets the position in which the current value is displayed.

```
function Get_Layout
  (Scale              : access Gtk_Scale_Record)
   return Pango.Layout.Pango_Layout;
```

Gets the Pango_Layout used to display the scale. The returned object is owned by the scale so does not need to be freed by the caller.

```
procedure Get_Layout_Offsets
  (Scale              : access Gtk_Scale_Record;
   X, Y               : out   Gint);
```

Obtains the coordinates where the scale will draw the Pango_Layout representing the text in the scale. Remember when using the Pango_Layout functions you need to convert to and from pixels using Pango.Enums.To_Pixels If the draw_value property is False, the return values are undefined.

163 Package Gtk.Scrollbar

This widget represents a widget that can be dragged by the user to change the visible area of another widget. It is typically only used through a Gtk.Scrolled_Window, although you might need, from time to time, to use it directly if the widget you want to scroll isn't entirely suitable for a scrolled window. For instance, if you are creating your own drawing area, unlimited in size, you do not want to create a Gtk_Drawing_Area 100_000 pixels large, since that would use too much memory. Instead, you create one with just the size of the visible area on the screen, then connect it with a scrollbar so that when the user moves the scrollbar, you change what should be displayed in the drawing area.

163.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Range          (Package Gtk.Grange)
         \___ Gtk_Scrollbar   (Package Gtk.Scrollbar)
```

163.2 Types

subtype Gtk_Hscrollbar **is** Gtk_Scrollbar;

subtype Gtk_Vscrollbar **is** Gtk_Scrollbar;

163.3 Subprograms

```
procedure Gtk_New_Hscrollbar
  (Widget             : out   Gtk_Scrollbar;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
procedure Gtk_New_Vscrollbar
  (Widget             : out   Gtk_Scrollbar;
   Adjustment         :      Gtk.Adjustment.Gtk_Adjustment);
function Get_Type               return Gtk.Gtk_Type;
function Hscrollbar_Get_Type    return Gtk.Gtk_Type;
function Vscrollbar_Get_Type    return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Scrollbar.

164 Package Gtk.Scrolled_Window

Gtk_Scrolled_Window is a Gtk_Bin child: it's a container the accepts a single child widget. Gtk_Scrolled_Window adds scrollbars to the child widget.

The scrolled window can work in two ways. Some widgets have native scrolling support; these widgets have "slots" for Gtk_Adjustment objects. The scrolled window installs Gtk_Adjustment objects in the child window's slots using the "set_scroll_adjustments" signal (Conceptually, these widgets implement a "Scrollable" interface).

The second way to use the scrolled window is useful with widgets that lack the "set_scroll_adjustments" signal. The Gtk_Viewport widget acts as a proxy, implementing scrollability for child widgets that lack their own scrolling capabilities.

If a widget has native scrolling abilities, it can be added to the Gtk_Scrolled_Window with Gtk.Container.Add. If a widget does not, you must first add the widget to a Gtk_Viewport, then add the Gtk_Viewport to the scrolled window. The convenience function Add_With_Viewport does exactly this, so you can ignore the presence of the viewport.

If you want to create your own new widget type that can be inserted directly into a scrolled_window, you need to specify a signal for Set_Scroll_Adjustments in the call to Gtk.Object.Initialize_Class_Record.

164.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Bin          (Package Gtk.Bin)
            \___ Gtk_Scrolled_Window (Package Gtk.Scrolled_Window)
```

164.2 Signals

- **"move_focus_out"**

  ```
  procedure Handler
  (Window     : access Gtk_Scrolled_Window_Record'Class;
  Direction  : Gtk_Direction_Type);
  ```

  Request that the keyboard focus be moved. You almost never have to emit this signal yourself, unless you are binding it to a key for user interaction. You do not need to connect to this signal

- **"scroll_child"**

  ```
  procedure Handler
  (Window     : access Gtk_Scrolled_Window_Record'Class;
  Typ        : Gtk_Scroll_Type;
  Horizontal : Gboolean);
  ```

  You should emit this signal to request a scrolling of the child. This signal is almost never needed directly, unless you connect it to a key binding. The boolean is used to further qualify Scroll_Start and Scroll_End, which do not have horizontal and vertical variants.

164.3 Subprograms

```
procedure Gtk_New
  (Scrolled_Window    : out   Gtk_Scrolled_Window;
   Hadjustment        :       Gtk_Adjustment := null;
   Vadjustment        :       Gtk_Adjustment := null);
```

Create a new scrolled window.
The two arguments are the scrolled window's horizontal and vertical adjustments; these will be shared with the scrollbars and the child widget to keep the bars in sync with the child. Usually you want to use the default value Null_Adjustment for the adjustments, which will cause the scrolled window to create them for you.

```
function Get_Type                return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Scrolled_Window.

```
procedure Set_Hadjustment
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   Hadjustment        :       Gtk_Adjustment);

function Get_Hadjustment
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk_Adjustment;
```

Set the Gtk_Adjustment for the horizontal scrollbar.
This adjustment is used to connect the horizontal scrollbar to the child widget's horizontal scroll functionality.

```
procedure Set_Vadjustment
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   Vadjustment        :       Gtk_Adjustment);

function Get_Vadjustment
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk_Adjustment;
```

Set the Gtk_Adjustment for the vertical scrollbar.
This adjustment is used to connect the vertical scrollbar to the child widget's vertical scroll functionality.

```
function Get_Hscrollbar
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk.Scrollbar.Gtk_Scrollbar;
```

Returns the horizontal scrollbar, or null if it doesn't have one.

```
function Get_Vscrollbar
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk.Scrollbar.Gtk_Scrollbar;
```

Returns the vertical scrollbar, or null if it doesn't have one.

```
procedure Set_Policy
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   H_Scrollbar_Policy  :       Enums.Gtk_Policy_Type;
   V_Scrollbar_Policy  :       Enums.Gtk_Policy_Type);

procedure Get_Policy
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   H_Scrollbar_Policy  : out   Enums.Gtk_Policy_Type;
   V_Scrollbar_Policy  : out   Enums.Gtk_Policy_Type);
```

Set the scrollbar policy for the horizontal and vertical scrollbars.
It determines when the scrollbar should appear; it is a value from the Gtk_Policy_Type enumeration. If Policy_Always, the scrollbar is always present; if Policy_Never, the scrollbar

is never present; if Policy_Automatic, the scrollbar is present only if needed (that is, if the slider part of the bar would be smaller than the trough - the display is larger than the page size).

```
procedure Set_Placement
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   Window_Placement   :      Gtk.Enums.Gtk_Corner_Type);
function Get_Placement
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk.Enums.Gtk_Corner_Type;
```

Determine or return the location of the widget with respect to the scrollbars. The default is Corner_Top_Left.

```
procedure Set_Shadow_Type
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   Shadow_Type        :      Gtk.Enums.Gtk_Shadow_Type);
function Get_Shadow_Type
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
   return Gtk.Enums.Gtk_Shadow_Type;
```

Change the type of shadow drawn around the contents of Scrolled_Window.

```
procedure Add_With_Viewport
  (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Used to add children without native scrolling capabilities.
This is simply a convenience function; it is equivalent to adding the unscrollable child to a viewport, then adding the viewport to the scrolled window. If a child has native scrolling, use Gtk.Container.Add instead of this function.

The viewport scrolls the child by moving its Gdk_Window, and takes the size of the child to be the size of its toplevel Gdk_Window. This will be very wrong for most widgets that support native scrolling; for example, if you add a Gtk_Clist with a viewport, the whole widget will scroll, including the column headings. Thus Gtk_Clist supports scrolling already, and should not be used with the GtkViewport proxy.

A widget supports scrolling natively if it contains a valid "set_scroll_adjustments" signal.

165 Package Gtk.Selection

This package implements support for the selection mechanism (ie a way to get a currently active selection anywhere on your Xserver or on your Windows machine).

This also acts as the low-level support for drag-and-drop, as described in Gtk.Dnd.

A lot of subprograms in this package work on Gdk_Atom types, instead of strings. Converting from one to the other can easily be done through calls to the subprograms in Gdk.Property (Atom_Intern and Atom_Name). The reason we use Gdk_Atom is for efficiency, since comparing two integers is of course faster than comparing two strings.

The selection mechanism is the primary mechanism by which applications can transfer data to each other on a given system. Even though both applications must be visible on the same screen, this does not mean that they can access the same files or ressources, since they might in fact be running on different machines. You should always keep this in mind when setting the data to be transfered.

A selection is a essentially a named clipboard, identified by a string interned as a Gdk_Atom. By claiming ownership of a selection, an application indicates that it will be responsible for supplying its contents.

The contents of a selection can be represented in a number of formats, called targets. Each target is identified by an atom. A list of all possible targets supported by the selection owner can be retrieved by requesting the special target TARGETS. When a selection is retrieved, the data is accompanied by a type (an atom), and a format (an integer, representing the number of bits per item).

See also http://standards.freedesktop.org/clipboards-spec/ for more information on the way selection works on X-Window systems.

165.1 Signals

- **"selection_get"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class;
  Data   : Selection_Data;
  Info   : Guint;
  Time   : Guint);
  ```

  This signal is sent to the owner of a selection whenever some other widget wants to get data from that selection. The type of the data is indicated in Info, and is the third field that was set in the Target_Entrys for that specific widget and selection.

  The handler should modify the Data in the selection.

- **"selection_received"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class;
  Data   : Selection_Data;
  Time   : Guint);
  ```

  This signal is sent to the receiver end of a selection, when the data has been sent by the owner. The receiver should call Convert, which will emit the signal selection_get to ask for the contents of the selection, and then selection_received will be emitted to warn the receiver.

  Note: you can not connect this signal to a widget that does not have an associated Gdk_Window (i.e the flag Gtk.Widget.No_Window must not be set for this widget),

since it needs to be able to receive Property_Notify events from the server. It will not work with a Gtk_Label for instance.

165.2 Types

subtype Gdk_Selection **is** Gdk.Types.Gdk_Atom;

These are predefined atom values for several common selections. You are of course free to create new ones, but most of the time you should simply use Selection_Primary unless you foresee the need for multiple simultaneous selections. To access the clipboard on windows machines, you might need to create a new selection with Gdk.Property.Atom_Intern ("CLIPBOARD");

subtype Gdk_Selection_Type **is** Gdk.Types.Gdk_Atom;

Predefined atom values for selection types. Although the preferred way in GtkAda to indicate the type of a selection is to use mime types, these values are provided for compatibility with older X11 applications.

subtype Gdk_Target **is** Gdk.Types.Gdk_Atom;

Predefined atom values which are used to describe possible targets for a selection. Other atoms can be used, and the recommended practice for GtkAda is to to use mime types for this purpose. However, supporting these types may be useful for compatibility with older programs.

type Selection_Data **is new** Gdk.C_Proxy;

Contents of a selection or a drag-and-drop operation. This structure can only be created internally by GtkAda. However, you need to be able to access it to get the selection. - Selection and Target identify the request. - Type specifies the type of the return. - if Length is negative, the Data should be ignored. Otherwise, it contains the data itself. - Time gives the timestamp at which the data was sent.

```
type Target_Entry is record
          Target : Gtkada.Types.Chars_Ptr;
          Flags  : Target_Flags;
          Info   : Guint;
          end record;
```

A single type of data that can be supplied or received during a drag-and-drop or a selection. Target is a string that represents the drag type. This can be any string if you want to implement drag-and-drop inside your application. However, if you want to communicate with other external application, you should use MIME types, ie "text/plain", "text/uri-list", ... See the RFCs 2045, 2046, 2047, 2048, 2049 for more information on MIME types. For more information, see ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/ Another set of supported names are the ones associated with the X Inter-Client Communications Conventions Manual (ICCCM). Here some of the default names and their meaning. See the ICCCM manual online for a complete list (for instance at http://www.tronche.com/gui/x/icccm/). - "TIMESTAMP" (type Integer) Timestamp used to acquire the selection - "TEXT" (type Text) Text in owner's encoding - "STRING" (type String) Iso Latin1 text - "PIXMAP"

(type Drawable) Pixmap Id - "BITMAP" (type Bitmap) Bitmap Id - "FOREGROUND" (type Pixel) Pixel Value - "BACKGROUND" (type Pixel) Pixel Value Info is an application-assigned integer (i.e. that you choose), that will be passed as a signal parameter for all the dnd-related signals, like "selection_get". This saves a lot of expensive string compares, and in fact replaced Target everywhere in your application expect in Source_Set and Dest_Set.

type Target_Entry_Array **is array** (Natural **range** <>) **of** Target_Entry;

type Target_Flags **is new** Integer;

Used to specify constraints on an entry

type Target_List **is new** Gdk.C_Proxy;

A list of targets. You can only manipulate this list through the functions below.

165.3 Subprograms

165.3.1 Target_List

```
function Target_List_New
  (Targets            :      Target_Entry_Array)
   return Target_List;
```

Create a new list of target, starting from an array.

```
procedure Target_List_Ref
  (List               :      Target_List);
```

Increment the reference count for the list.
You should almost never have to use this function, this is done transparently by GtkAda.

```
procedure Target_List_Unref
  (List               :      Target_List);
```

Decrement the reference count for the list.
You should almost never have to use this function, since everything is done transparently by GtkAda. As usual, the list is freed when the reference count reaches 0.

```
procedure Target_List_Add
  (List               :      Target_List;
   Target             :      Gdk.Types.Gdk_Atom;
   Flags              :      Guint;
   Info               :      Guint);
```

Add a new target to the list.
You can for instance use the result of Get_Targets (Drag_Context) for the value of Target.

```
procedure Target_List_Add_Table
  (List               :      Target_List;
   Targets            :      Target_Entry_Array);
```

Add a new set of targets to the list.

```
procedure Target_List_Add_Text_Targets
  (List               :      Target_List;
   Info               :      Guint);
```

Appends the text targets supported internally by gtk+ to List.
All targets are added with the same info. Info will be passed back to the application.

```
procedure Target_List_Add_URI_Targets
  (List               :      Target_List;
   Info               :      Guint);
```

Appends the URI targets supported internally by gtk+ to List.
All targets are added with the same info.

```
procedure Target_List_Add_Image_Targets
  (List               :      Target_List;
   Info               :      Guint;
   Writable           :      Boolean);
```

Appends the image targets supported internally by gtk+ to List.
All targets are added with the same info. If Writable is True, then only those targets for which gtk+ knows how to convert a Gdk_Pixbuf into the format are added.

```
procedure Target_List_Remove
  (List               :      Target_List;
   Target             :      Gdk.Types.Gdk_Atom);
```

Remove a specific target from the list.

```
procedure Target_List_Find
  (List               :      Target_List;
   Target             :      Gdk.Types.Gdk_Atom;
   Info               : out  Guint;
   Found              : out  Boolean);
```

Search for a specific target in the list.
If the target was found, Found is set to True and Info contains the integer that was associated with the target when it was created.

165.3.2 Selection_Data

```
function Selection_Get_Type    return Glib.GType;
```

Return the internal type used for a selection

```
function Get_Selection
  (Selection          :      Selection_Data)
   return Gdk_Selection;
```

Return the selection used (primary, clipboard, ...)

```
function Get_Target
  (Selection          :      Selection_Data)
   return Gdk.Types.Gdk_Atom;
```

Return the target of the selection (ie a MIME string that identifies
the selection).

```
function Get_Type
  (Selection          :      Selection_Data)
   return Gdk.Types.Gdk_Atom;
```

Return the type of the selection, as defined in Gdk_Selection_Type,
ie for compatibility with older X11 applications.

```
function Get_Format
  (Selection          :      Selection_Data)
   return Gint;
```

Return the format of the data.
The semantics depends on the type of data. For instance, for strings this is the number of bits per character.

```
function Get_Data
  (Selection          :      Selection_Data)
   return System.Address;
```

Return the data of the selection.
This should be ignored if Get_Length returns a value < 0.

```
function Get_Data_As_String
  (Selection          :      Selection_Data)
   return String;
```

Return the data as a string.
This is only a convenience function, since it simply creates a string from the return of Get_Data.

```
function Get_Length
  (Selection          :      Selection_Data)
   return Gint;
```

Return the length of the data.

165.3.3 Setting and getting contents

```
function Set_Pixbuf
  (Selection          :      Selection_Data;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf)
   return Boolean;
```

Sets the contents of the selection from a pixbuf
The pixbuf is converted to the form determined by Get_Target (Selection_Data). Returns True if the selection was successfully set.

```
function Get_Pixbuf
  (Selection          :      Selection_Data)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Gets the contents of the selection data as a pixbuf.
Return value: if the selection data contained a recognized image type and it could be converted to a pixbuf, a newly allocated pixbuf is returned, otherwise null. If the result is non-null it must be freed with Unref.

```
function Targets_Include_Image
  (Selection          :      Selection_Data;
   Writable           :      Boolean := True)
   return Boolean;
```

Given a Selection object holding a list of targets, determines if any of
the targets in these targets can be used to provide a Gdk.Pixbuf. Writable: whether to accept only targets for which gtk+ knows how to convert a pixbuf into the format. Returns True if Selection holds a list of targets and a suitable target for images is included

```
function Set_Text
  (Selection          :      Selection_Data;
   Str                :      UTF8_String)
   return Boolean;
```

Sets the contents of the selection from a UTF-8 encoded string.
The string is converted to the form determined by Get_Target (Selection_Data).

```
function Get_Text
  (Selection          :      Selection_Data)
   return UTF8_String;
```

Gets the contents of the selection data as a UTF-8 string.
Return value: if the selection data contained a recognized text type and it could be converted to UTF-8, the string is returned.

```
function Targets_Include_Text
  (Selection          :      Selection_Data)
   return Boolean;
```

Given a Selection object holding a list of targets, determines if any of
the targets can be used to provide text.

```
function Set_Uris
  (Selection          :      Selection_Data;
   URIs               :      GNAT.Strings.String_List)
   return Boolean;
```

Sets the contents of the selection from a list of URIs.
The string is converted to the form determined by Get_Target (Selection). Return True if the selection was successfully set.

```
function Get_Uris
  (Selection          :      Selection_Data)
   return GNAT.Strings.String_List;
```

Gets the contents of the selection data as array of URIs.
The returned value must be freed by the caller.

```
function Get_Targets
  (Selection          :      Selection_Data)
   return Gdk.Types.Gdk_Atom_Array;
```

Gets the contents of Selection_Data as an array of targets.
This can be used to interpret the results of getting the standard TARGETS target that is always supplied for any selection. This is different from Get_Target, which indicate the current format that the selection contains. Get_Targets only applies when Get_Target is "TARGETS".

```
procedure Selection_Data_Set
  (Selection          :      Selection_Data;
   The_Type           :      Gdk.Types.Gdk_Atom;
   Format             :      Gint;
   Data               :      System.Address;
   Length             :      Gint);
```

General form of Selection_Data_Set.
Any data can be transmitted. Length is the number of bytes in Data.

```
procedure Selection_Data_Set
  (Selection          :      Selection_Data;
   The_Type           :      Gdk.Types.Gdk_Atom;
   Format             :      Gint;
   Data               :      String);
```

Set the data for a selection (special case for strings)
This function is generally called when a drag-and-drop operation ask the source widget for the data to be transmitted. In that case, a Selection_Data was already transmitted and is given as a handler parameter for the signal "drag_data_get". The_Type can simply be extracted from the Selection_Data.

```
function Selection_Data_Copy
  (Selection          :      Selection_Data)
   return Selection_Data;
```

Make a copy of a selection data.

```
procedure Selection_Data_Free
  (Selection          :      Selection_Data);
```

Free a Selection_Data structure returned from Selection_Data_Copy.

165.3.4 Manipulating the selection

```
function Owner_Set
  (Widget             :      Gtk.Widget.Gtk_Widget;
   Selection          :      Gdk_Selection
                      := Selection_Primary;
   Time               :      Guint32 := 0)
   return Boolean;
```

Claim ownership of a given selection for a particular widget,
or, if widget is null, release ownership of the selection.

Once a Widget has claimed selection, it is responsible for delivering the data whenever it is needed.

Time is the timestamp for claiming the selection (default is the current time). This function returns True if the operation succeeded.

```
procedure Add_Target
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Selection          :      Gdk_Selection;
   Target             :      Gdk.Types.Gdk_Atom;
   Info               :      Guint);
```

Add specified target to the list of supported targets for a given
widget and selection. Info is an integer which will be passed back to the application instead of a string when the target is used.

```
procedure Add_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Selection          :      Gdk_Selection;
   Targets            :      Target_Entry_Array);
```

Add a set of targets to the list of supported targets for a given widget
and selection.

```
procedure Clear_Targets
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Selection          :      Gdk_Selection);
```

Clear the list of supported targets for a given widget and selection.

```
function Convert
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Selection          :      Gdk_Selection
                      := Selection_Primary;
   Target             :      Gdk.Types.Gdk_Atom;
   Time               :      Guint32 := 0)
   return Boolean;
```

Request the contents of a selection.
When received, a "selection_received" signal will be generated, and the widget needs to have a handler for it.

Target is the form of information desired, for instance an intern Gdk_Atom whose name is "text/plain", or one of the Gdk_Target values.

This function returns True if the request succeeded, False if the request could not be processed, for instance if there was already a request in process for this widget or this target is not known by the owner of the selection.

Widget is the widget which acts as a requestor.

```
procedure Remove_All
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Remove all handlers and unsets ownership of all selections for a widget.
Called when widget is being destroyed. This function will not generally be called by applications.

165.3.5 Signals

```
function Make_Atom
  (Num                :      Gulong)
   return Gdk.Types.Gdk_Atom;
```

166 Package Gtk.Separator

A separator is a vertical or horizontal line that can be displayed between widgets, to provide visual grouping of the widgets into meaningful groups. It is for instance used in dialogs to isolate the actual contents of the dialogs and the various buttons to acknowledge the dialog (OK, Cancel,...)

166.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Separator      (Package Gtk.Separator)
```

166.2 Types

subtype Gtk_Hseparator **is** Gtk_Separator;

subtype Gtk_Vseparator **is** Gtk_Separator;

166.3 Subprograms

```
procedure Gtk_New_Hseparator
  (Separator          : out   Gtk_Separator);
procedure Gtk_New_Vseparator
  (Separator          : out   Gtk_Separator);
function Get_Type             return Gtk.Gtk_Type;
function Hseparator_Get_Type  return Gtk.Gtk_Type;
function Vseparator_Get_Type  return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Separator.

167 Package Gtk.Separator_Menu_Item

This widget serves as separator between menu items. It is represented graphically as a horizontal line between two items, and is used to group items into meaningful groups.

167.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Item       (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
                  \___ Gtk_Separator_Menu_Item
                          (Package Gtk.Separator_Menu_Item)
```

167.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Separator_Menu_Item);
```

Create a new Gtk_Image_Menu_Item.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

168 Package Gtk.Separator_Tool_Item

This package defines a separator widget that can be inserted in a toolbar, to create groups of widgets in the latter.

168.1 Subprograms

```
procedure Gtk_New
  (Separator          : out   Gtk_Separator_Tool_Item);
function Get_Type               return GType;
```

Return the internal type used for separators

```
procedure Set_Draw
  (Item               : access Gtk_Separator_Tool_Item_Record;
   Draw               :      Boolean);
function Get_Draw
  (Item               : access Gtk_Separator_Tool_Item_Record)
   return Boolean;
```

Sets whether the separator is drawn as a vertical line, or just a blank. Settings this to False along with using Gtk.Tool_Item.Set_Expand is useful to create an item that forces the following items to the end of the toolbar.

169 Package Gtk.Settings

This package contains various subprograms to easily share settings between applications, or even between various parts of your application.

169.1 Subprograms

```
function Get_Default           return Gtk_Settings;
```

Gets the settings object for the default GDK screen, creating it if necessary.

```
function Get_For_Screen
  (Screen             :      Gdk.Gdk_Screen)
   return Gtk_Settings;
```

Gets the settings object for Screen, creating it if necessary.

```
function Get_Type              return Glib.GType;
```

Return the internal type used to identify a Gtk_Settings

```
procedure Install_Property
  (Pspec              :      Glib.Param_Spec);
```

Declares a property that can be shared among various parts of the application

```
procedure Install_Property_Parser
  (Pspec              :      Glib.Param_Spec;
   Parser             :      Gtk.Style.Gtk_Rc_Property_Parser);
```

Install a new parser for the given property. This parser is responsible for reading the property's value in a gtk configuration file, and convert it to a suitable value.

169.1.1 Precoded parsing functions

```
function Parse_Color
  (Pspec              :      Glib.Param_Spec;
   Rc_String          :      Interfaces.C.Strings.chars_ptr;
   Value              : access Glib.Values.GValue)
   return Gboolean;
function Parse_Enum
  (Pspec              :      Glib.Param_Spec;
   Rc_String          :      Interfaces.C.Strings.chars_ptr;
   Value              : access Glib.Values.GValue)
   return Gboolean;
function Parse_Flags
  (Pspec              :      Glib.Param_Spec;
   Rc_String          :      Interfaces.C.Strings.chars_ptr;
   Value              : access Glib.Values.GValue)
   return Gboolean;
function Parse_Requisition
  (Pspec              :      Glib.Param_Spec;
   Rc_String          :      Interfaces.C.Strings.chars_ptr;
   Value              : access Glib.Values.GValue)
   return Gboolean;
function Parse_Border
  (Pspec              :      Glib.Param_Spec;
   Rc_String          :      Interfaces.C.Strings.chars_ptr;
   Value              : access Glib.Values.GValue)
```

```
   return Gboolean;
```

These functions parse some of the predefined property types

169.1.2 Setting predefined properties

```
procedure Set_Property_Value
  (Settings           : access Gtk_Settings_Record;
   Name               :      String;
   Value              :      Glib.Values.GValue;
   Origin             :      String);
procedure Set_String_Property
  (Settings           : access Gtk_Settings_Record;
   Name               :      String;
   Value              :      String;
   Origin             :      String);
procedure Set_Long_Property
  (Settings           : access Gtk_Settings_Record;
   Name               :      String;
   Value              :      Glong;
   Origin             :      String);
procedure Set_Double_Property
  (Settings           : access Gtk_Settings_Record;
   Name               :      String;
   Value              :      Gdouble;
   Origin             :      String);
```

Set the value of a property. This automatically propagates the new
value to all listeners, so that they can refresh themselves. Origin should be something like "filename:line" for rc files, or the name of the function that sets it otherwise

170 Package Gtk.Size_Group

Gtk_Size_Group provides a mechanism for grouping a number of widgets together so they all request the same amount of space. This is typically useful when you want a column of widgets to have the same size, but you can't use a Gtk_Table widget.

Note that size groups only affect the amount of space requested, not the size that the widgets finally receive. If you want the widgets in a Gtk_Size_Group to actually be the same size, you need to pack them in such a way that they get the size they request and not more. For example, if you are packing your widgets into a table, you would not include the Fill flag.

170.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Size_Group        (Package Gtk.Size_Group)
```

170.2 Types

type Property_Size_Group_Mode **is new** Size_Group_Mode_Properties.Property;

type Size_Group_Mode **is**
(None, Horizontal, Vertical, Both);

170.3 Subprograms

```
procedure Gtk_New
  (Size_Group         : out   Gtk_Size_Group;
   Mode               :       Size_Group_Mode := Both);
```

Create a new group.
Initially, it doesn't contain any widget, and you need to add them with the Add_Widget procedure.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Size_Group

```
procedure Set_Mode
  (Size_Group         : access Gtk_Size_Group_Record;
   Mode               :       Size_Group_Mode);
function Get_Mode
  (Size_Group         : access Gtk_Size_Group_Record)
   return Size_Group_Mode;
```

Change the way the group effects the size of its component widgets.

```
procedure Add_Widget
  (Size_Group         : access Gtk_Size_Group_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add a new widget in the group.
Its size will be effected by all other widgets in the group: the size requisition of the widget will be the maximum of its requisition and the requisition of the other widgets in the group (depending on the group mode).

A given widget can belong to only one size group. It is removed from its previous group before being added to Size_Group.

```
procedure Remove_Widget
  (Size_Group         : access Gtk_Size_Group_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Remove a widget from the group.

```
procedure Set_Ignore_Hidden
  (Size_Group         : access Gtk_Size_Group_Record;
   Ignore_Hidden      :      Boolean);

function Get_Ignore_Hidden
  (Size_Group         : access Gtk_Size_Group_Record)
   return Boolean;
```

Whether invisible widgets are ignored when calcuating the size for all widgets in the group.

171 Package Gtk.Socket

Note that this package is currently not supported under Win32 systems.

Together with Gtk_Plug, Gtk_Socket provides the ability to embed widgets from one process into another process in a fashion that is transparent to the user. One process creates a Gtk_Socket widget and, passes the XID of that widget's window to the other process, which then creates a Gtk_Plug window with that XID. Any widgets contained in the Gtk_Plug then will appear inside the first applications window.

The XID of the socket's window is obtained by using the XWindow function provided in this package. Before using this macro, the socket must have been realized, and for hence, have been added to its parent.

Note that if you pass the XID of the socket to another process that will create a plug in the socket, you must make sure that the socket widget is not destroyed until that plug is created. Violating this rule will cause unpredictable consequences, the most likely consequence being that the plug will appear as a separate toplevel window. You can check if the plug has been created by examining the plug_window field of the Gtk_Socket structure. If this field is non-NULL, then the plug has been succesfully created inside of the socket.

When GtkAda is notified that the embedded window has been destroyed, then it will destroy the socket as well. You should always, therefore, be prepared for your sockets to be destroyed at any time when the main event loop is running.

A socket can also be used to swallow arbitrary pre-existing top-level windows using Steal, though the integration when this is done will not be as close as between a Gtk_Plug and a Gtk_Socket. All you need in that case is the X11 window identifier for the external process.

Note that it is recommended that the external window be first hidden before being swallowed, so that Gtk.Socket works with most window managers. If you start with visible windows, some window managers will not be able to correctly merge the two windows (Enlightenment for instance).

171.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Socket      (Package Gtk.Socket)
```

171.2 Signals

- **"plug_added"**

  ```
  procedure Handler (Socket : access Gtk_Socket_Record'Class);
  ```

  Emitted when a client is successfully added to the socket

- **"plug_removed"**

  ```
  function Handler
  (Socket : access Gtk_Socket_Record'Class) return Boolean;
  ```

  Emitted when a client is successfully removed from the socket. The default action is to destroy the socket, so you want to reuse it you must return True.

171.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Socket);
```

Create a new empty GtkSocket.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Socket.

```
procedure Add_Id
  (Socket             : access Gtk_Socket_Record;
   Id                 :      Guint32);
```

Add an XEMBED client, such as a Gtk_Plug, to the Gtk_Socket.
The client may be in the same process or in a different process.

To embed a Gtk_Plug in a Gtk_Socket, you can either create the Gtk_Plug with Gtk_New (0), call Gtk.Plug.Get_Id to get the window ID of the plug, and then pass that to the Gtk.Socket.Add_Id, or you can call Gtk.Socket.Get_Id to get the window ID for the socket, and call Gtk.Plug.Gtk_New passing in that ID.

Id: the XID of a client participating in the XEMBED protocol.

The Gtk_Socket must have already be added into a toplevel window before you can make this call.

```
function Get_Id
  (Socket             : access Gtk_Socket_Record)
   return Guint32;
```

Get the window ID of a Gtk_Socket widget, which can then be used to
create a client embedded inside the socket, for instance with Gtk.Socket.Gtk_New (Id). The Gtk_Socket must have already been added into a toplevel window before you can make this call.

```
function Get_Plug_Window
  (Socket             : access Gtk_Socket_Record)
   return Gdk.Window.Gdk_Window;
```

Return the id of the embedded window.

171.4 Example

```
Obtaining the XID of a socket

with Gtk.Socket;
use Gtk.Socket;

Socket : Gtk_Socket;

Gtk_New (Socket);
Show (Socket);
Add (Parent, Socket);

--  The following call is only necessary if one of
--  the ancestors of the socket is not yet visible.

Realize (Socket);
```

```
Put_Line ("The XID of the sockets window is" &
          Guint32'Image (Get_Id (Socket)));
```

172 Package Gtk.Spin_Button

A Gtk_Spin_Button is a single line text editing widget for text that represents a number. At the right hand side of the text line there are small up- and down arrow buttons for incrementing or decrementing (spinning) the number within a given range. It allows the value to have zero or a number of decimal places and to be incremented/decremented in configurable steps. The action of holding down one of the buttons optionally results in an acceleration of change in the value according to how long it is depressed.

See Gtk.GEntry for a text editing widget without spin buttons.

172.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Editable        (Package Gtk.Editable)
         \___ Gtk_Entry        (Package Gtk.GEntry)
            \___ Gtk_Spin_Button (Package Gtk.Spin_Button)
```

172.2 Signals

- **"change_value"**

  ```
  procedure Handler
  (Spin : access Gtk_Spin_Button_Record'Class;
  Typ  : Gtk_Scroll_Type);
  ```

  You should emit this signal to request a change in the value of the spin button. This is mostly useful as a keybinding, and is bound, by default, to the arrow keys, PageUp, PageDown, Home and End keys.

- **"input"**

  ```
  procedure Handler
  (Spin  : access Gtk_Spin_Button_Record'Class;
  Value : out Gint);
  ```

  ???

- **"output"**

  ```
  procedure Handler (Spin : access Gtk_Spin_Button_Record'Class);
  ```

  ???

- **"value_changed"**

  ```
  procedure Handler (Spin : access Gtk_Spin_Button_Record'Class);
  ```

  Emitted when the value of the spin button has changed.

172.3 Types

```
type Gtk_Spin_Button_Update_Policy is
        (Update_Always,
            -- Update always, errors are ignored while converting text into a
            -- numeric value.

         Update_If_Valid
            -- The spin button's value gets changed if the text input is a numeric
            -- value that is within the range specified by the adjustment.
```

```
      );
```

Determine the update policy of the spin button which affects the behaviour when parsing inserted text and syncing its value with the values of the adjustment. pragma Convention (C, Gtk_Spin_Button_Update_Policy);

```
type Gtk_Spin_Type is
      (Spin_Step_Forward,
       Spin_Step_Backward,
       Spin_Page_Forward,
       Spin_Page_Backward,
       Spin_Home,
       Spin_End,
       Spin_User_Defined);
```

Determine how manual spinning should be done. See also the Spin procedure. pragma Convention (C, Gtk_Spin_Type);

```
type Property_Spin_Button_Update_Policy_Type is new Spin_Button_Update_Policy_Properties.Property;
```

172.4 Subprograms

```
procedure Gtk_New
  (Spin_Button        : out   Gtk_Spin_Button;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment;
   Climb_Rate         :       Gdouble;
   The_Digits         :       Gint);
```

Create a spin button with the given parameters.
Adjustment contains the range, current value, step value and "page" value. The step value is the increment/decrement when pressing mouse button 1 on a button; the page value when mouse button 2 is pressed. Additionally, mouse button 3 can be used to jump directly to the or lower values when used to select one of the buttons. Climb_Rate takes a value between 0.0 and 1.0 and indicates the amount of acceleration that the Spin Button has. The_Digits is the number of digits behind the decimal point to be displayed for the value.

```
procedure Gtk_New
  (Spin_Button        : out   Gtk_Spin_Button;
   Min                :       Gdouble;
   Max                :       Gdouble;
   Step               :       Gdouble);
```

Same as above, but with explicit range instead of an adjustment.
The adjustment associated with Spin_Button is created internally.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Spin_Button.

```
procedure Set_Adjustment
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment);
function Get_Adjustment
  (Spin_Button        : access Gtk_Spin_Button_Record)
```

```
   return Gtk.Adjustment.Gtk_Adjustment;
```

Set or Get the adjustment settings of the spin button.

```
procedure Set_Digits
  (Spin_Button        : access Gtk_Spin_Button_Record;
   The_Digits         :      Guint);

function Get_Digits
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Guint;
```

Set or Get number of decimals of the spin button.

```
procedure Set_Increments
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Step               :      Gdouble;
   Page               :      Gdouble);

procedure Get_Increments
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Step               : out    Gdouble;
   Page               : out    Gdouble);
```

Set or Get the increments for a single step and a page move.

```
procedure Set_Range
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Min                :      Gdouble;
   Max                :      Gdouble);

procedure Get_Range
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Min                : out    Gdouble;
   Max                : out    Gdouble);
```

Set or Get range of the spin button.

```
procedure Set_Value
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Value              :      Gdouble);

function Get_Value
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Gdouble;
```

Set or Get the current value of the spin button in a double.

```
function Get_Value_As_Int
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Gint;
```

Return the current value of the spin button in an integer.

```
procedure Set_Update_Policy
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Policy             :      Gtk_Spin_Button_Update_Policy);

function Get_Update_Policy
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Gtk_Spin_Button_Update_Policy;
```

Set the update policy of the spin button.
See Gtk_Spin_Button_Update_Policy for the meaning of Policy.

```
procedure Set_Numeric
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Numeric            :      Boolean);
```

```
function Get_Numeric
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Boolean;
```

If Numeric is True, then only a numeric value can be typed in the
text entry, otherwise also nonnumeric text.

```
procedure Spin
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Direction          :      Gtk_Spin_Type;
   Step               :      Gdouble);
```

Set the value of the spin button relative to its current value.
Depending on Direction, it will be incremented or decremented with the step value.

```
procedure Set_Wrap
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Wrap               :      Boolean);
function Get_Wrap
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Boolean;
```

Set whether the spin button should "wrap around" when exceeding the
upper and lower limits.

```
procedure Set_Snap_To_Ticks
  (Spin_Button        : access Gtk_Spin_Button_Record;
   Snap_To_Ticks      :      Boolean);
function Get_Snap_To_Ticks
  (Spin_Button        : access Gtk_Spin_Button_Record)
   return Boolean;
```

Set the spin button to round the value to the nearest step value
which is set within its adjustment settings.

```
procedure Update
  (Spin_Button        : access Gtk_Spin_Button_Record);
```

Manually force an update of the spin button.

173 Package Gtk.Status_Bar

A status bar is a special widget in which you can display messages. This type of widget is generally found at the bottom of application windows, and is used to display help or error messages.

This widget works as a stack of messages, ie all older messages are kept when a new one is inserted. It is of course possible to remove the most recent message from the stack. This stack behavior is especially useful when messages can be displayed from several places in your application. Thus, each one subprogram that needs to print a message can simply push it on the stack, and does not need to make sure that the user has had enough time to read the previous message (a timeout can be set to automatically remove the message after a specific delay)

Each message is associated with a specific Context_Id. Each of this context can have a special name, and these context can be used to organize the messages into categories (for instance one for help messages and one for error messages). You can then selectively remove the most recent message of each category.

173.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Box         (Package Gtk.Box)
            \___ Gtk_Status_Bar (Package Gtk.Status_Bar)
```

173.2 Signals

- **"text_popped"**

  ```
  procedure Handler
  (Status_Bar : access Gtk_Status_Bar_Record'Class;
  Context    : Context_Id;
  Text       : Interfaces.C.Strings.chars_ptr);
  ```

  Emitted when a message has been removed from the queue.

- **"text_pushed"**

  ```
  procedure Handler
  (Status_Bar : access Gtk_Status_Bar_Record'Class;
  Context    : Context_Id;
  Text       : Interfaces.C.Strings.chars_ptr);
  ```

  Emitted when a new message has been in the queue.

173.3 Types

type Context_Id **is new** Guint;

subtype Gtk_Statusbar **is** Gtk_Status_Bar;

This is needed by Gate since the C name is GtkStatusbar

```
type Message_Id is new Guint;
```

```
type Status_Bar_Msg is record
        Text    : Interfaces.C.Strings.chars_ptr;
        Context : Context_Id;
        Message : Message_Id;
        end record;
```

A message from the queue. Each of this message is associated with a specific context, and has a specific number.

173.4 Subprograms

```
function Convert
  (Msg                :      Status_Bar_Msg)
   return System.Address;
function Convert
  (Msg                :      System.Address)
   return Status_Bar_Msg;
procedure Gtk_New
  (Statusbar          : out   Gtk_Status_Bar);
```

Create a new status bar, in which messages will be displayed.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Status_Bar.

```
function Get_Context_Id
  (Statusbar          : access Gtk_Status_Bar_Record;
   Context_Description  :      String)
   return Context_Id;
```

Create the context id associated with a special name.
If no context is currently associated with Context_Description, then a new context is created.

```
function Get_Messages
  (Statusbar          : access Gtk_Status_Bar_Record)
   return Messages_List.GSlist;
```

Return a list of all the messages currently stored in the queue.
The first item in the list is the most recent message.

```
function Push
  (Statusbar          : access Gtk_Status_Bar_Record;
   Context            :      Context_Id;
   Text               :      UTF8_String)
   return Message_Id;
```

Push a new message on the queue, associated with a specific context.
This message is directly displayed in the status bar. A new unique message id is associated with this message.

```
procedure Pop
  (Statusbar          : access Gtk_Status_Bar_Record;
   Context            :      Context_Id);
```

Remove the most recent message from a specific context. All other
contexts are ignored, and no error is raised if there is no message in Context.

```
procedure Remove
  (Statusbar          : access Gtk_Status_Bar_Record;
   Context            :      Context_Id;
   Message            :      Message_Id);
```

Remove a message from the list.
The message is only removed if it is in a specific context. Nothing happens if no matching message is found.

```
procedure Set_Has_Resize_Grip
  (Statusbar          : access Gtk_Status_Bar_Record;
   Setting            :      Boolean);
function Get_Has_Resize_Grip
  (Statusbar          : access Gtk_Status_Bar_Record)
   return Boolean;
```

Set or gets the value of the resize_grip attribute for a given status
bar. This indicates whether the status bar has a handle that, when dragged, will resize the toplevel window that contains the status bar.

174 Package Gtk.Stock

These functions provide an applications programmer with default images and buttons for toolbars, menu pixmaps, etc.

See the function Gtk.Widget.Render_Icon for a convenience function that converts a stock icon to an actual pixmap/pixbuf.

174.1 Types

```
type Gtk_Stock_Item is record
        Stock_Id           : Gtkada.Types.Chars_Ptr;
        Label              : Gtkada.Types.Chars_Ptr;
        Modifier           : Gdk.Types.Gdk_Modifier_Type;
        Keyval             : Gdk.Types.Gdk_Key_Type;
        Translation_Domain : Gtkada.Types.Chars_Ptr;
        end record;
```

```
type Gtk_Stock_Item_Access is access all Gtk_Stock_Item;
```

```
type Gtk_Stock_Item_Array is array (Natural range <>) of Gtk_Stock_Item;
```

174.2 Subprograms

```
procedure Gtk_New
  (Item               : out   Gtk_Stock_Item;
   Stock_Id           :       String;
   Label              :       UTF8_String;
   Modifier           :       Gdk.Types.Gdk_Modifier_Type;
   Keyval             :       Gdk.Types.Gdk_Key_Type;
   Translation_Domain  :       String);
```

Create a new stock item.

```
procedure Add
  (Item               :       Gtk_Stock_Item);
```

Register Item.
If an item already exists with the same stock ID as one of the items, the old item gets replaced. The stock item is copied, so GtkAda does not hold any pointer into item and item can be freed. Use Add_Static if item is persistent and GtkAda need not copy the array.

```
procedure Add
  (Items              :       Gtk_Stock_Item_Array);
```

Register each of the stock items in Items.

```
procedure Add_Static
  (Item               :       Gtk_Stock_Item);
```

Same as Add, but do not copy Item, so Item must persist until application exit.

```
procedure Add_Static
  (Items              :      Gtk_Stock_Item_Array);
```

Same as Add, but do not copy Items, so Items must persist until application exit.

```
procedure Lookup
  (Stock_Id           :      String;
   Item               : out  Gtk_Stock_Item;
   Success            : out  Boolean);
```

Fill Item with the registered values for Stock_Id.
Success if set to True of Stock_Id was known.

```
procedure Free
  (Item               : in out Gtk_Stock_Item);
```

Free memory allocated in Item.

175 Package Gtk.Style

This package contains various functions to draw widget parts on the screen. Whenever possible, you should use these functions instead of directly the ones from Gdk.Drawable, since this package will properly take into account the user's theme and color choice.

Consider also using directly the function Gtk.Widget.Modify_Font, Gtk.Widget.Modify_Background,... rather than use the lower level Gtk_Style object.

See Gtk.RC to learn how styles can be defined in external configuration files by the end-user of your application.

175.1 Signals

- **"realize"**

  ```
  procedure Handler (Style : access Gtk_Style_Record'Class);
  ```

  Emitted when the style has been initialized for a particular colormap and depth. Connecting to this signal is probably seldom useful since most of the time applications and widgets only deal with styles that have been already realized.

- **"unrealize"**

  ```
  procedure Handler (Style : access Gtk_Style_Record'Class);
  ```

  Emitted when the aspects of the style specific to a particular colormap and depth are being cleaned up. A connection to this signal can be useful if a widget wants to cache objects like a Gdk_GC as object data on Gtk_Style. This signal provides a convenient place to free such cached objects.

175.2 Types

```
type Gtk_Rc_Property_Parser is access function
        (Pspec     : Glib.Param_Spec;
```

175.3 Subprograms

175.3.1 Styles

```
procedure Gtk_New
  (Style              : out   Gtk_Style);
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Style.

```
function Copy
  (Source             :       Gtk_Style)
   return Gtk_Style;
```

Copy a Gtk_Style

```
function Attach
  (Style              :       Gtk_Style;
   Window             :       Gdk.Window.Gdk_Window)
   return Gtk_Style;
```

```
procedure Detach
  (Style              :      Gtk_Style);
```

Attaches a style to a window; this process allocates the colors and creates the GC's for the style - it specializes it to a particular visual and colormap. The process may involve the creation of a new style if the style has already been attached to a window with a different style and colormap. It returns either Style or a newly allocated style. If a new one is created, the parameter will be Unref once, and the new one Ref once.

```
procedure Set_Background
  (Style              :      Gtk_Style;
   Window             :      Gdk.Window.Gdk_Window;
   State_Type         :      Gtk_State_Type);
```

Set the background color of Window to the background color specified by Style.

```
procedure Apply_Default_Background
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk.Window.Gdk_Window;
   Set_Bg             :      Boolean;
   State_Type         :      Gtk_State_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle;
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Applies the default background from style to the given area in Window

175.3.2 Properties

The style contains a number of properties. Each of these can be set to@* multiple values simulatenously, that will be applied depending on the widget's current state (highlighted, active, inactive,...)

```
procedure Set_Background
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);

procedure Set_Bg
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);

function Get_Background
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.Color.Gdk_Color;

function Get_Bg
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_Color;
```

Set or get the background color that this style uses in the given state

```
procedure Set_Background_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);

procedure Set_Background
```

```
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    GC                 :      Gdk_GC);
procedure Set_Bg
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    GC                 :      Gdk.GC.Gdk_GC);
procedure Set_Bg_GC
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    GC                 :      Gdk.GC.Gdk_GC);
function Get_Background_GC
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type)
    return Gdk.GC.Gdk_GC;
function Get_Bg
   (Style              :      Gtk_Style;
    State_Type         :      Gtk_State_Type)
    return Gdk_GC;
function Get_Background
   (Style              :      Gtk_Style;
    State_Type         :      Gtk_State_Type)
    return Gdk_GC;
function Get_Bg_GC
   (Style              :      Gtk_Style;
    State_Type         :      Gtk_State_Type)
    return Gdk_GC;
```

Set or get the graphic context that the style is using for the background

```
procedure Set_Foreground
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    Color              :      Gdk.Color.Gdk_Color);
procedure Set_Fg
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    Color              :      Gdk.Color.Gdk_Color);
function Get_Foreground
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type)
    return Gdk.Color.Gdk_Color;
function Get_Fg
   (Style              :      Gtk_Style;
    State_Type         :      Gtk_State_Type)
    return Gdk_Color;
```

Set or get the foreground color that the style is using

```
procedure Set_Foreground_GC
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    GC                 :      Gdk.GC.Gdk_GC);
procedure Set_Fg_GC
   (Style              :      Gtk_Style;
    State_Type         :      Enums.Gtk_State_Type;
    GC                 :      Gdk.GC.Gdk_GC);
```

```
procedure Set_Foreground
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
procedure Set_Fg
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Foreground_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.GC.Gdk_GC;
function Get_Foreground
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
function Get_Fg
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
function Get_Fg_GC
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the graphic context used by this style for the foreground

```
procedure Set_Light
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Light
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.Color.Gdk_Color;
procedure Set_Light_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Light_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.GC.Gdk_GC;
procedure Set_Light
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Light
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the lighter color or graphic context that this style is using. This color is used to draw the shadows around rectangles for instance

```
procedure Set_Dark
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
```

```
procedure Set_Dark_GC
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
procedure Set_Dark
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Dark
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk.Color.Gdk_Color;
function Get_Dark_GC
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk.GC.Gdk_GC;
function Get_Dark
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the darker color or graphic context that this style is using.
This color is used to draw the shadows around rectangles for instance.

```
procedure Set_Middle
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
procedure Set_Mid
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Middle
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk.Color.Gdk_Color;
function Get_Mid
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_Color;
procedure Set_Middle_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Middle_GC
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
procedure Set_Middle
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk_GC);
function Get_Middle
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

```
procedure Set_Mid_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk_GC);
function Get_Mid_GC
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
procedure Set_Mid
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Mid
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the middle color. This color should be between the light and dark colors set above.

```
procedure Set_Text
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Text
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.Color.Gdk_Color;
procedure Set_Text_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Text_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.GC.Gdk_GC;
procedure Set_Text
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Text
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the color to use when drawing text.

```
procedure Set_Base
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Base
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.Color.Gdk_Color;
procedure Set_Base_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
```

```
function Get_Base_GC
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.GC.Gdk_GC;
procedure Set_Base
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Base
  (Style              :      Gtk_Style;
   State_Type         :      Gtk_State_Type)
   return Gdk_GC;
```

Set or get the base color

```
procedure Set_Black
  (Style              :      Gtk_Style;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Black
  (Style              :      Gtk_Style)
   return Gdk.Color.Gdk_Color;
procedure Set_Black_GC
  (Style              :      Gtk_Style;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Black_GC
  (Style              :      Gtk_Style)
   return Gdk.GC.Gdk_GC;
procedure Set_Black
  (Style              :      Gtk_Style;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_Black
  (Style              :      Gtk_Style)
   return Gdk.GC.Gdk_GC;
```

Set or get the "black" color. It isn't necessarily black, although most themes will want to use black here.

```
procedure Set_White
  (Style              :      Gtk_Style;
   Color              :      Gdk.Color.Gdk_Color);
function Get_White
  (Style              :      Gtk_Style)
   return Gdk.Color.Gdk_Color;
procedure Set_White_GC
  (Style              :      Gtk_Style;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_White_GC
  (Style              :      Gtk_Style)
   return Gdk.GC.Gdk_GC;
procedure Set_White
  (Style              :      Gtk_Style;
   GC                 :      Gdk.GC.Gdk_GC);
function Get_White
  (Style              :      Gtk_Style)
   return Gdk.GC.Gdk_GC;
```

Set or get the "white" color. It isn't necessarily white, although most themes will want to use white here.

```
procedure Set_Font_Description
  (Style              :      Gtk_Style;
   Desc               :      Pango.Font.Pango_Font_Description);
function Get_Font_Description
  (Style              :      Gtk_Style)
   return Pango.Font.Pango_Font_Description;
```

Set or get the font to use for this style

```
procedure Set_Bg_Pixmap
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type;
   Pixmap             :      Gdk.Pixmap.Gdk_Pixmap);
function Get_Bg_Pixmap
  (Style              :      Gtk_Style;
   State_Type         :      Enums.Gtk_State_Type)
   return Gdk.Pixmap.Gdk_Pixmap;
```

Set or get the pixmap to use for background

```
function X_Thickness
  (Style              :      Gtk_Style)
   return Gint;
```

Width of the vertical scrollbars and ranges when Style is applied.
In fact, this thickness is used for a lot of widgets whose width does not depend on their content, such as rulers,...

```
function Y_Thickness
  (Style              :      Gtk_Style)
   return Gint;
```

Height of the horizontal scrollbars and ranges when Style is applied.

175.3.3 Painting

All the subprograms below have similar profiles.@* Area is always a clipping area. Drawing only takes place within that area, and pixels outside of it are not affected. Detail is a theme-specific detail string. This is generally provided by the application (or rather gtk+ itself) to indicate that the drawing should be slightly different, and Detail describes the exact contact. All drawings are done on Window. Widget is used to draw specific things depending on the widget type.

```
procedure Draw_Insertion_Cursor
  (Widget             : access Gtk.Object.Gtk_Object_Record'Class;
   Drawable           :      Gdk_Drawable;
   Area               :      Gdk.Rectangle.Gdk_Rectangle;
   Location           :      Gdk.Rectangle.Gdk_Rectangle;
   Is_Primary         :      Boolean;
   Direction          :      Gtk_Text_Direction;
   Draw_Arrow         :      Boolean);
```

Draws a text caret on Drawable at Location. This is not a style function
but merely a convenience function for drawing the standard cursor shape. Is_Primary indicates whether the cursor should be the primary cursor color. Direction is the text direction. Draw_Arrow should be true to draw a directional arrow on the cursor. Should be False unless the cursor is split.

```
procedure Paint_Handle
  (Style              :      Gtk_Style;
   Window             :      Gdk.Gdk_Window;
```

```
   State_Type          :      Gtk.Enums.Gtk_State_Type;
   Shadow_Type         :      Gtk.Enums.Gtk_Shadow_Type;
   Area                :      Gdk.Rectangle.Gdk_Rectangle;
   Widget              : access Gtk.Object.Gtk_Object_Record'Class;
   Detail              :      String := "paned";
   X, Y, Width, Height  :       Gint;
   Orientation         :      Gtk.Enums.Gtk_Orientation);
```

Paint the handles as is done in the Gtk_Paned widget (ie a series of small dots in Window, that indicate that Window can be manipulated and resized. If Detail is "paned", only a few dots are painted in the middle of window (aligned either horizontally or vertically depending on Orientation). Any other value for Detail draws points on the whole length of Window. (X, Y, Width, Height) is the area in which the dots should be painted. For the whole window, use (0, 0, -1, -1). Only the area that intersect Area is drawn.

```
procedure Paint_Arrow
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   Arrow_Type         :      Gtk_Arrow_Type;
   Fill               :      Boolean;
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws an arrow in the given rectangle on Window using the given parameters. Arrow_Type determines the direction of the arrow. The default theme engine only recognazied "menu_scroll_arrow_up" for Detail.

```
procedure Paint_Box
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a box on Window with the given parameters.
The default theme engine recognizes the following for Detail: "spinbutton_up", "spinbutton_down", "paned", "optionmenu"

```
procedure Paint_Box_Gap
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
```

```
                    := Gdk.Rectangle.Full_Area;
   Widget           : access Glib.Object.GObject_Record'Class;
   Detail           :      String := "";
   X                :      Gint;
   Y                :      Gint;
   Width            :      Gint;
   Height           :      Gint;
   Gap_Side         :      Gtk_Position_Type;
   Gap_X            :      Gint;
   Gap_Width        :      Gint);
```

Draws a box in Window using the given style and state and shadow type, leaving a gap in one side.

```
procedure Paint_Check
  (Style            : access Gtk_Style_Record;
   Window           :      Gdk_Window;
   State_Type       :      Gtk_State_Type;
   Shadow_Type      :      Gtk_Shadow_Type;
   Area             :      Gdk.Rectangle.Gdk_Rectangle
                    := Gdk.Rectangle.Full_Area;
   Widget           : access Glib.Object.GObject_Record'Class;
   Detail           :      String := "";
   X                :      Gint;
   Y                :      Gint;
   Width            :      Gint;
   Height           :      Gint);
```

Draws a check button indicator in the given rectangle on Window with the given parameters. The default theme handles the following values for detail: "cellcheck", "check"

```
procedure Paint_Diamond
  (Style            : access Gtk_Style_Record;
   Window           :      Gdk_Window;
   State_Type       :      Gtk_State_Type;
   Shadow_Type      :      Gtk_Shadow_Type;
   Area             :      Gdk.Rectangle.Gdk_Rectangle
                    := Gdk.Rectangle.Full_Area;
   Widget           : access Glib.Object.GObject_Record'Class;
   Detail           :      String := "";
   X                :      Gint;
   Y                :      Gint;
   Width            :      Gint;
   Height           :      Gint);
```

Draws a diamond in the given rectangle on Window using the given parameters.

```
procedure Paint_Expander
  (Style            : access Gtk_Style_Record;
   Window           :      Gdk_Window;
   State_Type       :      Gtk_State_Type;
   Area             :      Gdk.Rectangle.Gdk_Rectangle
                    := Gdk.Rectangle.Full_Area;
   Widget           : access Glib.Object.GObject_Record'Class;
   Detail           :      String := "";
   X                :      Gint;
   Y                :      Gint;
   Expander_Style   :      Gtk_Expander_Style);
```

Draws an expander as used in Gtk_Tree_View. X and Y specify the center
the expander. The size of the expander is determined by the "expander-size" style property of Widget. (If widget is not specified or doesn't have an "expander-size" property, an unspecified default size will be used, since the caller doesn't have sufficient information to position the expander, this is likely not useful.) The expander is expander_size pixels tall in the collapsed position and expander_size pixels wide in the expanded position.

```
procedure Paint_Extension
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint;
   Gap_Side           :      Gtk_Position_Type);
```

Draws an extension, i.e. a notebook tab.

```
procedure Paint_Flat_Box
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a flat box on Window with the given parameters.

```
procedure Paint_Focus
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a focus indicator around the given rectangle on Window using the given style.

```
procedure Paint_Hline
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
```

```
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X1                 :      Gint;
   X2                 :      Gint;
   Y                  :      Gint);
```

Draws a horizontal line from (X1, Y) to (X2, Y) in Window using the given style and state.

```
procedure Paint_Layout
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Use_Text           :      Boolean;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Layout             :      Pango.Layout.Pango_Layout);
```

Draws a layout on Window using the given parameters.

```
procedure Paint_Option
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a radio button indicator in the given rectangle on Window with the given parameters.

```
procedure Paint_Polygon
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   Points             :      Gdk.Types.Gdk_Point;
   Npoints            :      Gint;
   Fill               :      Boolean);
```

Draws a polygon on Window with the given parameters.

```
procedure Paint_Resize_Grip
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
```

```
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   Edge               :      Gdk.Window.Gdk_Window_Edge;
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a resize grip in the given rectangle on Window using the given parameters.

```
procedure Paint_Shadow
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws a shadow around the given rectangle in Window using the given style and state and shadow type.

```
procedure Paint_Shadow_Gap
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint;
   Gap_Side           :      Gtk_Position_Type;
   Gap_X              :      Gint;
   Gap_Width          :      Gint);
```

Draws a shadow around the given rectangle in Window
using the given style and state and shadow type, leaving a gap in one side.

```
procedure Paint_Slider
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint;
```

```
   Orientation        :      Gtk_Orientation);
```

Draws a slider in the given rectangle on Window using the given style and orientation.

```
procedure Paint_Tab
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Shadow_Type        :      Gtk_Shadow_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Draws an option menu tab (i.e. the up and down pointing arrows) in the given rectangle on Window using the given parameters.

```
procedure Paint_Vline
  (Style              : access Gtk_Style_Record;
   Window             :      Gdk_Window;
   State_Type         :      Gtk_State_Type;
   Area               :      Gdk.Rectangle.Gdk_Rectangle
                      := Gdk.Rectangle.Full_Area;
   Widget             : access Glib.Object.GObject_Record'Class;
   Detail             :      String := "";
   Y1                 :      Gint;
   Y2                 :      Gint;
   X                  :      Gint);
```

Draws a vertical line from (X, Y1) to (X, Y2) in Window using the given style and state.

176 Package Gtk.Table

A Gtk_Table is a container that can contain any number of children. Each of them is attached to a specific row and a specific column in widget. Every row in the table must have the same height, and every column must have the same width if the table was said as Homogeneous. But you can also decide to have an heterogeneous table, where the width and height are set by the children contained in the table. Check out the Gtk_Sheet widget for a different kind of table that can also contain text and images in a more efficient way.

176.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Table       (Package Gtk.Table)
```

176.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Table;
   Rows               :       Guint;
   Columns            :       Guint;
   Homogeneous        :       Boolean);
```

Create a new table.
The width allocated to the table is divided into Columns columns, which all have the same width if Homogeneous is True. If Homogeneous is False, the width will be calculated with the children contained in the table. Same behavior for the rows.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Table.

```
procedure Resize
  (Table              : access Gtk_Table_Record;
   Rows               :       Guint;
   Columns            :       Guint);
```

Modify the number of rows and columns in the table.

```
procedure Attach
  (Table              : access Gtk_Table_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Left_Attach        :       Guint;
   Right_Attach       :       Guint;
   Top_Attach         :       Guint;
   Bottom_Attach      :       Guint;
   Xoptions           :       Gtk_Attach_Options
                       := Expand or Fill;
   Yoptions           :       Gtk_Attach_Options
                       := Expand or Fill;
   Xpadding           :       Guint := 0;
   Ypadding           :       Guint := 0);
```

Insert a new widget in the table.
All the attachments are relative to the separations between columns and rows (for instance, to insert a widget spanning the first two columns in the table, you should put Left_Attach=0 and Right_Attach=2). Same behavior for the rows. Xoptions and Yoptions indicate the

behavior of the child when the table is resized (whether the child can shrink or expand). See the description in Gtk.Box for more information on the possible values. Xpadding and Ypadding are the amount of space left around the child.

```
procedure Attach_Defaults
  (Table              : access Gtk_Table_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Left_Attach        :      Guint;
   Right_Attach       :      Guint;
   Top_Attach         :      Guint;
   Bottom_Attach      :      Guint);
```

Insert a new widget in the table, with default values.
No padding is put around the child, and the options are set to Expand and Fill. This call is similar to Attach with default values and is only provided for compatibility.

```
procedure Set_Row_Spacing
  (Table              : access Gtk_Table_Record;
   Row                :      Guint;
   Spacing            :      Guint);
```

Set the spacing insert between Row and the next one.
Spacing is in pixels.

```
procedure Set_Col_Spacing
  (Table              : access Gtk_Table_Record;
   Column             :      Guint;
   Spacing            :      Guint);
```

Set the spacing in pixels between Column and the next one.

```
procedure Set_Row_Spacings
  (Table              : access Gtk_Table_Record;
   Spacing            :      Guint);
```

Set the spacing for all the rows.

```
procedure Set_Col_Spacings
  (Table              : access Gtk_Table_Record;
   Spacing            :      Guint);
```

Set the spacing for all the columns.

```
procedure Set_Homogeneous
  (Table              : access Gtk_Table_Record;
   Homogeneous        :      Boolean);
```

Indicate the homogeneous status of the table.
If Homogeneous is True, the rows and columns of the table will all be allocated the same width or height.

```
function Get_Row_Spacing
  (Table              : access Gtk_Table_Record;
   Row                :      Guint)
   return Guint;
```

Return the spacing in pixels between Row and the next one.

```
function Get_Col_Spacing
  (Table              : access Gtk_Table_Record;
   Column             :      Guint)
   return Guint;
```

Return the spacing in pixels between Column and the next one.

```
function Get_Default_Row_Spacing
  (Table              : access Gtk_Table_Record)
   return Guint;
```

Return the default spacing for the rows.

```
function Get_Default_Col_Spacing
  (Table              : access Gtk_Table_Record)
   return Guint;
```

Return the default spacing for the columns.

```
function Get_Homogeneous
  (Table              : access Gtk_Table_Record)
   return Boolean;
```

Return the homogeneous status of the table.
See Set_Homogeneous for more details.

177 Package Gtk.Tearoff_Menu_Item

This package contains a special type of menu item, which is displayed as a hashed line, and which is used to tear off a menu (ie detach it from the menu bar, and into its own toplevel window, so that the user can keep it visible at all time).

177.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Bin          (Package Gtk.Bin)
            \___ Gtk_Item      (Package Gtk.Item)
               \___ Gtk_Menu_Item (Package Gtk.Menu_Item)
                  \___ Gtk_Tearoff_Menu_Item
                          (Package Gtk.Tearoff_Menu_Item)
```

177.2 Subprograms

```
procedure Gtk_New
  (Menu_Item          : out   Gtk_Tearoff_Menu_Item);
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Tearoff_Menu_Item.

178 Package Gtk.Text

This widget displays any given text that can be manipulated by both the user and the programmer. The text can optionally be interactively modified by the user. Different colors and fonts can be used for any given part of the text. The background can have any color, or even be a pixmap.

178.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Old_Editable     (Package Gtk.Old_Editable)
         \___ Gtk_Text          (Package Gtk.Text)
```

178.2 Subprograms

```
procedure Gtk_New
  (Text                : out   Gtk_Text;
   Hadj                : in    Adjustment.Gtk_Adjustment
                        := null;
   Vadj                : in    Adjustment.Gtk_Adjustment
                        := null);
```

Create a new text widget with the given adjustments.
If either or both scrollbars is not provided, the text widget will create its own. You need to insert the Gtk_Text in a Gtk_Scrolled_Window to make the scrollbars visible. Not also that this widget does not currently support horizontal scrollbars.

```
function Get_Type               return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Text.

```
function Get_Text_Area
  (Text                : access Gtk_Text_Record)
   return Gdk.Window.Gdk_Window;
```

Return the specific window into which the text is displayed.
Note that a Gtk_Text is in fact a complex widget, which includes borders on the sides. Thus, whenever you want to convert the mouse coordinates to a position in the text, you should use the Gdk.Window.Get_Pointer function, passing it this text area as the origin window, rather than directly Get_Window (Text). Note that null will be returned while Text hasn't been realized.

```
function Backward_Delete
  (Text                : access Gtk_Text_Record;
   Nchars              : in    Guint)
   return Boolean;
```

Backward delete Nchars characters from the current cursor position.
There must be at least Nchars characters to delete before the pointer, or the operation will not be performed. Return True if the operation was successful, False otherwise.

```
function Forward_Delete
  (Text                : access Gtk_Text_Record;
   Nchars              : in    Guint)
   return Boolean;
```

Forward delete Nchars characters from the current point position.
There must be at least Nchars characters to delete after the pointer, or the operation will not be performed. Return True if the operation was successful, False otherwise.

```
procedure Freeze
  (Text                : access Gtk_Text_Record);
```

Freeze the Gtk_Text widget.
In other words, stop any redrawing of the widget until the Thaw operation is called. This operation is useful when a large number of changes need to be made within the widget. Freezing it during the updates will avoid some flicker seen by the user. Note also that an internal counter is incremented. The updates will be performed only when the same numbers of calls to Thaw has been performed.

Note that you can not call Set_Position while the widget is frozen. This will create a Storage_Error otherwise.

```
procedure Thaw
  (Text                : access Gtk_Text_Record);
```

Cancel the previous call to Freeze.
Allow the widget to be redrawn again, when Thaw has been called as many times as Freeze.

```
function Get_Hadj
  (Text                : access Gtk_Text_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the horizontal scrollbar associated with Text.

```
function Get_Vadj
  (Text                : access Gtk_Text_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Return the vertical scrollbar associated to the given text widget.

```
function Get_Length
  (Text                : access Gtk_Text_Record)
   return Guint;
```

Return the total length of the text contained within the text widget.

```
function Get_Point
  (Text                : access Gtk_Text_Record)
   return Guint;
```

Get the current position of the insertion point (cursor).
Return the number of characters from the upper left corner of the widget.

```
procedure Set_Point
  (Text                : access Gtk_Text_Record;
   Index               : in    Guint);
```

Set the insertion point position.
This does not modify the position of the visible cursor (see Gtk.Editable.Set_Position instead).

```
procedure Insert
  (Text                : access Gtk_Text_Record;
   Font                : in    Gdk.Font.Gdk_Font
                        := Gdk.Font.Null_Font;
   Fore                : in    Gdk.Color.Gdk_Color
                        := Gdk.Color.Null_Color;
   Back                : in    Gdk.Color.Gdk_Color
                        := Gdk.Color.Null_Color;
   Chars               : in    UTF8_String := "";
```

```
   Length             : in    Gint := -1);
```

Insert the given string (Chars) inside the text of the text widget.
Use the specified Font, foreground (Fore) and background (Back) colors. Only the first "Length" characters are inserted, unless Length is set to -1, in which case the complete string is inserted. Note that the colors must be allocated first, and the font loaded. If the default parameters are passed for font and colors, the text widget will use the ones defined in the style for Text (see Gtk.Style for more information about styles).

```
procedure Set_Adjustments
  (Text               : access Gtk_Text_Record;
   Hadj               :      Gtk.Adjustment.Gtk_Adjustment;
   Vadj               :      Gtk.Adjustment.Gtk_Adjustment);
```

Set the horizontal and vertical adjustments associated with Text.

```
procedure Set_Editable
  (Text               : access Gtk_Text_Record;
   Editable           : in    Boolean := True);
```

Toggle the editable state of the given text widget.
This determines whether the user can edit the text or not. Note that the programmer can still perform any update.

```
procedure Set_Line_Wrap
  (Text               : access Gtk_Text_Record;
   Line_Wrap          : in    Boolean := True);
```

Set the Line_Wrap state of the given text widget.
If set to True, the line is broken when it reaches the extent of the widget viewing area and the rest is displayed on the next line. If set to false, the line continues regardless of the size of current viewing area.

```
procedure Set_Word_Wrap
  (Text               : access Gtk_Text_Record;
   Word_Wrap          : in    Boolean := True);
```

Set the Word_Wrap state of the given text widget.
If set to True, words are wrapped down to the next line if they can't be completed on the current line.

179 Package Gtk.Text_Attributes

This package defines the Gtk_Text_Attributes type.

179.1 Types

type Gtk_Text_Appearance **is new** Glib.C_Proxy;

type Gtk_Text_Attributes **is new** Glib.C_Proxy;

179.2 Subprograms

```
procedure Gtk_New
  (Text_Attr          : out   Gtk_Text_Attributes);
```

Create a new Gtk_Text_Attributes structure.

```
function Get_Type               return Glib.GType;
```

Return the internal type used fro a Gtk_Text_Attributes

```
procedure Ref
  (Values             :       Gtk_Text_Attributes);
```

Increase the reference counter of the given Gtk_Text_Attributes by one (this counter is initially set to 1 when this structure is created).

```
procedure Unref
  (Values             :       Gtk_Text_Attributes);
```

Decrease the reference counter by one. When it reaches zero, the Gtk_Text_Attributes is automatically deallocated.

```
function Copy
  (Src                :       Gtk_Text_Attributes)
   return Gtk_Text_Attributes;
```

Create a copy of the given Gtk_Text_Attributes structure.

```
procedure Copy_Values
  (Src                :       Gtk_Text_Attributes;
   Dest               :       Gtk_Text_Attributes);
```

Copy the values from Src into Dest so that Dest has the same values as Src. Free existing values in Dest. Dest's reference counter is preserved.

179.2.1 Text appearance

```
procedure Set_Rise
  (Appearance         :       Gtk_Text_Appearance;
   Rise               :       Gint);
function Get_Rise
  (Appearance         :       Gtk_Text_Appearance)
   return Gint;
```

Offset of the text above the baseline (or below if negative)

```
procedure Set_Underline
  (Appearance         :       Gtk_Text_Appearance;
   Underline          :       Pango.Enums.Underline);
```

```
function Get_Underline
  (Appearance         :      Gtk_Text_Appearance)
   return Pango.Enums.Underline;
```

Set the underline mode

```
procedure Set_Strikethrough
  (Appearance         :      Gtk_Text_Appearance;
   Strikethrough      :      Boolean);
function Get_Strikethrough
  (Appearance         :      Gtk_Text_Appearance)
   return Boolean;
```

Whether to strike through the text

```
procedure Set_Fg_Color
  (Appearance         :      Gtk_Text_Appearance;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Fg_Color
  (Appearance         :      Gtk_Text_Attributes)
   return Gdk.Color.Gdk_Color;
```

The color used to display the text

```
procedure Set_Bg_Color
  (Appearance         :      Gtk_Text_Appearance;
   Color              :      Gdk.Color.Gdk_Color);
function Get_Bg_Color
  (Appearance         :      Gtk_Text_Attributes)
   return Gdk.Color.Gdk_Color;
```

The background color for the text

```
procedure Set_Fg_Stipple
  (Appearance         :      Gtk_Text_Appearance;
   Stipple            :      Gdk.Gdk_Bitmap);
function Get_Fg_Stipple
  (Appearance         :      Gtk_Text_Attributes)
   return Gdk.Gdk_Bitmap;
```

The pattern used in the foreground

```
procedure Set_Bg_Stipple
  (Appearance         :      Gtk_Text_Appearance;
   Stipple            :      Gdk.Gdk_Bitmap);
function Get_Bg_Stipple
  (Appearance         :      Gtk_Text_Attributes)
   return Gdk.Gdk_Bitmap;
```

The pattern used in the background

179.2.2 Attributes

```
procedure Set_Font
  (Text_Attr          :      Gtk_Text_Attributes;
   Font               :      Pango.Font.Pango_Font_Description);
function Get_Font
  (Text_Attr          :      Gtk_Text_Attributes)
   return Pango.Font.Pango_Font_Description;
```

Return the Pango_Font_Description associated to the given Gtk_Text_Attributes.

```
procedure Set_Justification
  (Text_Attr          :      Gtk_Text_Attributes;
   Justification      :      Gtk.Enums.Gtk_Justification);
function Get_Justification
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gtk.Enums.Gtk_Justification;
```

Set the justification for this attributes

```
procedure Set_Direction
  (Text_Attr          :      Gtk_Text_Attributes;
   Direction          :      Gtk.Enums.Gtk_Text_Direction);
function Get_Direction
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gtk.Enums.Gtk_Text_Direction;
```

Set the text direction for this attributes

```
procedure Set_Font_Scale
  (Text_Attr          :      Gtk_Text_Attributes;
   Scale              :      Gdouble);
function Get_Font_Scale
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gdouble;
```

Set the scaling to use for the font

```
procedure Set_Left_Margin
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
function Get_Left_Margin
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Set the left margin

```
procedure Set_Right_Margin
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
function Get_Right_Margin
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Set the right margin

```
procedure Set_Indent
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
function Get_Indent
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Amount to indent the paragraph

```
procedure Set_Pixels_Above_Line
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
function Get_Pixels_Above_Line
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Set the number of blank pixels above paragraphs

```
procedure Set_Pixels_Below_Line
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
```

```
function Get_Pixels_Below_Line
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Set the number of blank pixels below paragraphs

```
procedure Set_Pixels_Inside_Wrap
  (Text_Attr          :      Gtk_Text_Attributes;
   Margin             :      Gint);
function Get_Pixels_Inside_Wrap
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gint;
```

Set the number of pixels between wrapped lines in a paragraph

```
procedure Set_Wrap_Mode
  (Text_Attr          :      Gtk_Text_Attributes;
   Mode               :      Gtk.Enums.Gtk_Wrap_Mode);
function Get_Wrap_Mode
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gtk.Enums.Gtk_Wrap_Mode;
```

Set the wrapping mode

```
procedure Set_Invisible
  (Text_Attr          :      Gtk_Text_Attributes;
   Invisible          :      Boolean);
function Get_Invisible
  (Text_Attr          :      Gtk_Text_Attributes)
   return Boolean;
```

Whether the text is invisible

```
procedure Set_Bg_Full_Height
  (Text_Attr          :      Gtk_Text_Attributes;
   Full_Height        :      Boolean);
function Get_Bg_Full_Height
  (Text_Attr          :      Gtk_Text_Attributes)
   return Boolean;
```

Whether the background occupies the full line height rather than just the area occupied by the text.

```
procedure Set_Editable
  (Text_Attr          :      Gtk_Text_Attributes;
   Editable           :      Boolean);
function Get_Editable
  (Text_Attr          :      Gtk_Text_Attributes)
   return Boolean;
```

Whether the text is editable

```
procedure Set_Tabs
  (Text_Attr          :      Gtk_Text_Attributes;
   Tabs               :      Pango.Tabs.Pango_Tab_Array);
function Get_Tabs
  (Text_Attr          :      Gtk_Text_Attributes)
   return Pango.Tabs.Pango_Tab_Array;
```

Set the default tab stops for paragraphs

```
function Get_Appearance
  (Text_Attr          :      Gtk_Text_Attributes)
   return Gtk_Text_Appearance;
```

Return the appearance of the text. This can be modified with the subprograms above.

180 Package Gtk.Text_Buffer

This is the public representation of a text buffer to be used in coordination with Gtk.Text_View.

180.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Text_Buffer       (Package Gtk.Text_Buffer)
```

180.2 Signals

- **"apply_tag"**

```
procedure Handler
(Widget     : access Gtk_Text_Buffer_Record'Class;
Tag        : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
Start_Char : Gtk.Text_Iter.Gtk_Text_Iter;
End_Char   : Gtk.Text_Iter.Gtk_Text_Iter);
```

- **"begin_user_action"**

```
procedure Handler (Widget : access Gtk_Text_Buffer_Record'Class);
```

- **"changed"**

```
procedure Handler (Widget : access Gtk_Text_Buffer_Record'Class);
```

- **"delete_range"**

```
procedure Handler
(Widget  : access Gtk_Text_Buffer_Record'Class;
Start   : Gtk.Text_Iter.Gtk_Text_Iter;
The_End : Gtk.Text_Iter.Gtk_Text_Iter);
```

- **"end_user_action"**

```
procedure Handler (Widget : access Gtk_Text_Buffer_Record'Class);
```

- **"insert_child_anchor"**

```
procedure Handler
(Widget : access Gtk_Text_Buffer_Record'Class;
Pos    : Gtk.Text_Iter.Gtk_Text_Iter;
Anchor : access Gtk.Text_Child.Gtk_Text_Child_Anchor_Record'Class);
```

- **"insert_pixbuf"**

```
procedure Handler
(Widget : access Gtk_Text_Buffer_Record'Class;
Pos    : Gtk.Text_Iter.Gtk_Text_Iter;
Pixbuf : Gdk.Pixbuf.Gdk_Pixbuf);
```

- **"insert_text"**

```
procedure Handler
(Widget : access Gtk_Text_Buffer_Record'Class;
Pos    : Gtk.Text_Iter.Gtk_Text_Iter;
Text   : UTF8_String;
Length : Gint);
```

- **"mark_deleted"**

```
procedure Handler
(Widget : access Gtk_Text_Buffer_Record'Class;
Mark   : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class);
```

- **"mark_set"**

```
procedure Handler
(Widget   : access Gtk_Text_Buffer_Record'Class;
Location : Gtk.Text_Iter.Gtk_Text_Iter;
Mark     : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class);
```

- **"modified_changed"**

```
procedure Handler (Widget : access Gtk_Text_Buffer_Record'Class);
```

- **"remove_tag"**

```
procedure Handler
(Widget     : access Gtk_Text_Buffer_Record'Class;
Tag        : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
Start_Char : Gtk.Text_Iter.Gtk_Text_Iter;
End_Char   : Gtk.Text_Iter.Gtk_Text_Iter);
```

180.3 Subprograms

```
procedure Gtk_New
  (Buffer             : out   Gtk_Text_Buffer;
   Table              :      Gtk.Text_Tag_Table.Gtk_Text_Tag_Table
                       := null);

function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Text_Buffer.

```
function Get_Line_Count
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Gint;
```

Return the number of lines in the buffer.
This value is cached, so the function is very fast.

```
function Get_Char_Count
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Gint;
```

Return the number of characters in the buffer.
Note that characters and bytes are not the same, you can't e.g. expect the contents of the buffer in string form to be this many bytes long. The character count is cached, so this function is very fast.

180.3.1 Modifiying the buffer

```
procedure Set_Modified
  (Buffer             : access Gtk_Text_Buffer_Record;
   Setting            :      Boolean := True);

function Get_Modified
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Boolean;
```

Used to keep track of whether the buffer has been modified since the
last time it was saved. Whenever the buffer is saved to disk, call Set_Modified (Buffer, False). When the buffer is modified, it will automatically toggled on the modified bit again. When the modified bit flips, the buffer emits a "modified_changed" signal.

```
procedure Set_Text
  (Buffer             : access Gtk_Text_Buffer_Record;
   Text               :      UTF8_String);
```

Delete current contents of Buffer, and insert Text instead.
If Text doesn't end with a newline, a newline is added; Gtk_Text_Buffer contents must always end with a newline. If Text ends with a newline, the new buffer contents will be exactly Text. Text: UTF-8 format text to insert.

```
procedure Insert
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      UTF8_String);
procedure Insert
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      Gtkada.Types.Chars_Ptr);
```

Insert Text at position Iter.
Emit the "insert_text" signal; insertion actually occurs in the default handler for the signal. Iter is invalidated when insertion occurs (because the buffer contents change), but the default signal handler revalidates it to point to the end of the inserted text. Text: UTF-8 format text to insert.

```
procedure Insert_With_Tags
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      UTF8_String;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag);
procedure Insert_With_Tags
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      Gtkada.Types.Chars_Ptr;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag);
```

Same as Insert, but specifies the tag to apply to the range.

```
procedure Insert_With_Tags_By_Name
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      UTF8_String;
   Tag_Name           :      String);
```

Same as Insert_With_Tags, but the tag is specified by its name

```
procedure Insert_At_Cursor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Text               :      UTF8_String);
```

Call Buffer_Insert, using the current cursor position
as the insertion point. Text: UTF-8 format text to insert.

```
procedure Insert_At_Cursor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Text               :      Gtkada.Types.Chars_Ptr;
   Len                :      Gint := -1);
```

Call Buffer_Insert, using the current cursor position
as the insertion point. Text: UTF-8 format C string to insert.

```
procedure Insert_Interactive
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Text               :      UTF8_String;
   Default_Editable   :      Boolean;
   Result             : out   Boolean);
```

Like Insert, but the insertion will not occur if Iter is at a
non-editable location in the buffer. Usually you want to prevent insertions at ineditable locations if the insertion results from a user action (is interactive).

Default_Editable indicates the editability of text that doesn't have a tag affecting editability applied to it. Typically the result of Gtk.Text_View.Get_Editable is appropriate here. Text: UTF-8 format text to insert. Result: whether text was actually inserted.

```
function Insert_Interactive_At_Cursor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Text               :      UTF8_String;
   Default_Editable   :      Boolean)
   return Boolean;
```

Call Insert_Interactive at the cursor position.
Text: UTF-8 format text to insert. Return value: whether text was actually inserted.

```
procedure Insert_Range
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Copy text, tags, and pixbufs between Start and End.
The order of Start and End doesn't matter. Also insert the copy at Iter. Used instead of simply getting/inserting text because it preserves images and tags. If Start and End are in a different buffer from Buffer, the two buffers must share the same tag table. Implemented via emissions of the insert_text and apply_tag signals, so expect those.

```
procedure Insert_Range_Interactive
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter;
   Default_Editable   :      Boolean;
   Result             : out   Boolean);
```

Like Insert_Range, does nothing if the insertion point isn't editable.
The Default_Editable parameter indicates whether the text is editable at Iter if no tags enclosing Iter affect editability. Typically the result of Gtk.Text_View.Get_Editable is appropriate here. Result: whether an insertion was possible at Iter

```
procedure Insert_Pixbuf
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               :      Gtk.Text_Iter.Gtk_Text_Iter;
   Pixbuf             :      Gdk.Pixbuf.Gdk_Pixbuf);
```

Insert an image into the text buffer at Iter.
The image will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 16#FFFC#. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for pixbufs, but the "text" variants do not. e.g. see Get_Slice and Get_Text.

```
procedure Delete
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              : in out Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            : in out Gtk.Text_Iter.Gtk_Text_Iter);
```

Delete text between Start and End.
The order of Start and End is not actually relevant; Delete will reorder them. This function

actually emits the "delete_range" signal, and the default handler of that signal deletes the text. Because the buffer is modified, all outstanding iterators become invalid after calling this function; however, the Start and End will be re-initialized to point to the location where text was deleted.

Note that the final newline in the buffer may not be deleted; a Gtk_Text_Buffer always contains at least one newline. You can safely include the final newline in the range [Start,End) but it won't be affected by the deletion.

```
procedure Delete_Interactive
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start_Iter         : in out Gtk.Text_Iter.Gtk_Text_Iter;
   End_Iter           : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Default_Editable   :      Boolean;
   Result             : out   Boolean);
```

Delete all editable text in the given range.
Call Delete for each editable sub-range of [Start,End). Start and End are revalidated to point to the location of the last deleted range, or left untouched if no text was deleted. Result: whether some text was actually deleted

```
function Backspace
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               :      Gtk.Text_Iter.Gtk_Text_Iter;
   Interactive        :      Boolean;
   Default_Editable   :      Boolean)
   return Boolean;
```

Performs the appropriate action as if the user hit the delete key with
the cursor at the position specified by Iter. In the normal case a single character will be deleted, but when combining accents are involved, more than one character can be deleted, and when precomposed character and accent combinations are involved, less than one character will be deleted. Because the buffer is modified, all outstanding iterators become invalid after calling this function; however, Iter will be re-initialized to point to the location where text was deleted. Interactive should be true if the deletion is caused by user interaction. Default_Editable: Whether the buffer is editable by default. Returns True if the buffer was modified.

180.3.2 Reading the buffer contents

```
function Get_Text
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter;
   Include_Hidden_Chars  :      Boolean := False)
   return UTF8_String;
```

Return the text in the range [Start,End).
Exclude undisplayed text (text marked with tags that set the invisibility attribute) if Include_Hidden_Chars is False. Does not include characters representing embedded images, so byte and character indexes into the returned string do not correspond to byte and character indexes into the buffer. Contrast with Get_Slice. Return value: an allocated UTF-8 string

```
function Get_Text
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter;
```

```
   Include_Hidden_Chars  :      Boolean := False)
   return Gtkada.Types.Chars_Ptr;
```

Same as Get_Text above, but return a pointer to a C string, for
efficiency. The caller is responsible for freeing (using Gtkada.Types.g_free) the returned pointer.

```
function Get_Slice
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter;
   Include_Hidden_Chars  :      Boolean := False)
   return UTF8_String;
```

Return the text in the range [Start,End).
Exclude undisplayed text (text marked with tags that set the invisibility attribute) if Include_Hidden_Chars is False. The returned string includes a 16#FFFC# character whenever the buffer contains embedded images, so byte and character indexes into the returned string do correspond to byte and character indexes into the buffer. Contrast with Get_Text. Note that 16#FFFC# can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the buffer. Return value: an allocated UTF-8 string

```
function Get_Slice
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter;
   Include_Hidden_Chars  :      Boolean := False)
   return Gtkada.Types.Chars_Ptr;
```

Same as Get_Slice above, but return a pointer to a C string, for
efficiency. The caller is responsible for freeing (using Gtkada.Types.g_free) the returned pointer.

180.3.3 Marks

See Gtk.Text_Mark

```
function Create_Mark
  (Buffer             : access Gtk_Text_Buffer_Record;
   Mark_Name          :      String := "";
   Where              :      Gtk.Text_Iter.Gtk_Text_Iter;
   Left_Gravity       :      Boolean := True)
   return Gtk.Text_Mark.Gtk_Text_Mark;
```

Create a mark at position Where.
If Mark_Name is null, the mark is anonymous; otherwise, the mark can be retrieved by name using Get_Mark. If a mark has left gravity, and text is inserted at the mark's current location, the mark will be moved to the left of the newly-inserted text. If the mark has right gravity (Left_Gravity = False), the mark will end up on the right of newly-inserted text. The standard left-to-right cursor is a mark with right gravity (when you type, the cursor stays on the right side of the text you're typing).

The caller of this function does not own a reference to the returned Gtk_Text_Mark, so you can ignore the return value if you like. Marks are owned by the buffer and go away when the buffer does. Emit the "mark_set" signal as notification of the mark's initial placement.

Return value: the new Gtk_Text_Mark object.

```
procedure Move_Mark
  (Buffer             : access Gtk_Text_Buffer_Record;
```

```
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class;
   Where              :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Move Mark to the new location Where.
Emit the "mark_set" signal as notification of the move.

```
procedure Delete_Mark
  (Buffer             : access Gtk_Text_Buffer_Record;
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class);
```

Delete Mark, so that it's no longer located anywhere in the
buffer. Remove the reference the buffer holds to the mark, so if you haven't called Ref on the mark, it will be freed. Even if the mark isn't freed, most operations on Mark become invalid. There is no way to undelete a mark. Gtk.Text_Mark.Get_Deleted will return True after this function has been called on a mark; Gtk.Text_Mark.Get_Deleted indicates that a mark no longer belongs to a buffer. The "mark_deleted" signal will be emitted as notification after the mark is deleted.

```
function Get_Mark
  (Buffer             : access Gtk_Text_Buffer_Record;
   Name               :      String)
   return Gtk.Text_Mark.Gtk_Text_Mark;
```

Return the mark named Name in Buffer
or null if no such mark exists in the buffer.

```
procedure Move_Mark_By_Name
  (Buffer             : access Gtk_Text_Buffer_Record;
   Name               :      String;
   Where              :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Move the mark named Name (which must exist) to location Where.
See Move_Mark for details.

```
procedure Delete_Mark_By_Name
  (Buffer             : access Gtk_Text_Buffer_Record;
   Name               :      String);
```

Delete the mark named Name
The mark must exist. See Delete_Mark for details.

```
function Get_Insert
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Gtk.Text_Mark.Gtk_Text_Mark;
```

Return the mark that represents the cursor (insertion point).
Equivalent to calling Get_Mark to get the mark name "insert", but slightly more efficient, and involves less typing.

```
function Get_Selection_Bound
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Gtk.Text_Mark.Gtk_Text_Mark;
```

Return the mark that represents the selection bound.
Equivalent to calling Get_Mark to get the mark name "selection_bound", but very slightly more efficient, and involves less typing.

The currently-selected text in Buffer is the region between the "selection_bound" and "insert" marks. If "selection_bound" and "insert" are in the same place, then there is no current selection. Get_Selection_Bounds is another convenient function for handling the selection, if you just want to know whether there's a selection and what its bounds are.

```
function Get_Buffer
  (Mark               :      Gtk.Text_Mark.Gtk_Text_Mark)
   return Gtk_Text_Buffer;
```

Return the buffer associated to the given mark

180.3.4 Cursor

The cursor is a special mark in the buffer

```
procedure Place_Cursor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Where              :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Move the "insert" and "selection_bound" marks simultaneously.
If you move them to the same place in two steps with Move_Mark, you will temporarily select a region in between their old and new locations, which can be pretty inefficient since the temporarily-selected region will force stuff to be recomputed. This function moves them as a unit, which can be optimized.

If you want to get the position of the cursor, the simplest way is Get_Iter_At_Mark (Buffer, Iter, Get_Insert (Buffer));

180.3.5 Tags

Tags can be applied to change the properties of a range of text

```
function Create_Tag
  (Buffer             : access Gtk_Text_Buffer_Record;
   Tag_Name           :      String := "")
   return Gtk.Text_Tag.Gtk_Text_Tag;
```

Creates a tag and adds it to the tag table for Buffer. Equivalent to
calling gtk.text_tag.gtk_new and then adding the tag to the buffer's tag table. The returned tag is owned by the buffer's tag table, so the ref count will be equal to one.

If Tag_Name is NULL, the tag is anonymous, otherwise a tag called Tag_Name must not already exist in the tag table for this buffer.

```
function Get_Tag_Table
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Gtk.Text_Tag_Table.Gtk_Text_Tag_Table;
```

Get the Gtk_Text_Tag_Table associated with this buffer.

```
procedure Apply_Tag
  (Buffer             : access Gtk_Text_Buffer_Record;
   Tag                : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Emit the "apply_tag" signal on Buffer.
The default handler for the signal applies Tag to the given range. Start and End do not have to be in order.

```
procedure Remove_Tag
  (Buffer             : access Gtk_Text_Buffer_Record;
   Tag                : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Emit the "remove_tag" signal.
The default handler for the signal removes all occurrences of Tag from the given range. Start and End don't have to be in order.

```
procedure Remove_All_Tags
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Remove all tags in the range between Start and End.
Note that this procedure should be used carefully, as it might be removing tags that were added from another section of the code.

```
procedure Apply_Tag_By_Name
  (Buffer             : access Gtk_Text_Buffer_Record;
   Name               :      String;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Call Gtk.Text_Tag_Table.Lookup on the buffer's tag table to get a Gtk_Text_Tag, then call Apply_Tag.

```
procedure Remove_Tag_By_Name
  (Buffer             : access Gtk_Text_Buffer_Record;
   Name               :      String;
   Start              :      Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            :      Gtk.Text_Iter.Gtk_Text_Iter);
```

Call Gtk.Text_Tag_Table.Lookup on the buffer's tag table to get a Gtk_Text_Tag, then call Remove_Tag.

180.3.6 Iterators

```
procedure Get_Iter_At_Line_Offset
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Line_Number        :      Gint;
   Char_Offset        :      Gint := 0);
```

Obtain an iterator pointing to Char_Offset within the given line. The Char_Offset must exist, offsets off the end of the line are not allowed. Note characters, not bytes; UTF-8 may encode one character as multiple bytes. If the Line_Number is an existing line but the Char_Offset is past the last offset, the iter pointing at the beginning of the line is returned. If the Line_Number is not valid, the behavior is undetermined.

```
procedure Get_Iter_At_Line_Index
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Line_Number        :      Gint;
   Byte_Index         :      Gint := 0);
```

Obtain an iterator pointing to Byte_Index within the given line.
Byte_Index must be the start of a UTF-8 character, and must not be beyond the end of the line. Note bytes, not characters; UTF-8 may encode one character as multiple bytes.

```
procedure Get_Iter_At_Offset
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Char_Offset        :      Gint);
```

Initialize Iter to a position Char_Offset chars from the start of the entire buffer. Char_Offset: char offset from start of buffer, counting from 0.

```
procedure Get_Iter_At_Line
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
```

```
   Line_Number        :      Gint);
```

Initialize Iter to the start of the given line.
Line_Number: line number counting from 0.

```
procedure Get_Start_Iter
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter);
```

Initialize Iter with the first position in the text buffer. This is the same as using Get_Iter_At_Offset with Offset set to 0.

```
procedure Get_End_Iter
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter);
```

Initialize Iter with the "end iterator", one past the last valid character in the text buffer. If dereferenced with Gtk.Text_Iter.Get_Char, the end iterator has a character value of 0. The entire buffer lies in the range from the first position in the buffer (call Get_Iter_At_Offset to get character position 0) to the end iterator.

```
procedure Get_Bounds
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              : out   Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            : out   Gtk.Text_Iter.Gtk_Text_Iter);
```

Retrieve the first and last iterators in the buffer.
The entire buffer lies within the range [Start,End).

```
procedure Get_Iter_At_Mark
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class);
```

Initialize Iter with the current position of Mark.

```
function Get_Buffer
  (Iter               :      Gtk.Text_Iter.Gtk_Text_Iter)
   return Gtk_Text_Buffer;
```

Return the buffer associated to the given Gtk_Text_Iterator.

180.3.7 Widgets

Widgets can be put in the buffer at specific places. See@* Gtk.Text_Child

```
procedure Get_Iter_At_Child_Anchor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Anchor             : access Gtk.Text_Child.Gtk_Text_Child_Anchor_Record'Class);
```

Obtains the location of Anchor within Buffer.

```
procedure Insert_Child_Anchor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Anchor             : access Gtk.Text_Child.Gtk_Text_Child_Anchor_Record'Class);
```

Insert a child widget anchor into the text buffer at Iter.
The anchor will be counted as one character in character counts, and when obtaining the buffer contents as a string, will be represented by the Unicode "object replacement character" 16#FFFC#. Note that the "slice" variants for obtaining portions of the buffer as a string include this character for pixbufs, but the "text" variants do not. e.g. see Get_Slice and Get_Text. Consider Create_Child_Anchor as a more convenient alternative to this function. The buffer will add a reference to the anchor, so you can unref it after insertion.

```
procedure Create_Child_Anchor
  (Buffer             : access Gtk_Text_Buffer_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Gtk.Text_Child.Gtk_Text_Child_Anchor);
```

Convenience function which simply creates a child anchor with Gtk.Text_Child.Gtk_New and inserts it into the buffer with Insert_Child_Anchor. Result: the created child anchor.

180.3.8 Clipboard and selection

```
procedure Add_Selection_Clipboard
  (Buffer             : access Gtk_Text_Buffer_Record;
   Clipboard          :      Gtk.Clipboard.Gtk_Clipboard);
```

Adds Clipboard to the list of clipboards in which the selection contents of Buffer are available. In most cases, Clipboard will be the clipboard corresponding to SELECTION_PRIMARY. You generally do not have to call this procedure yourself unless you are creating your own clipboards.

```
procedure Remove_Selection_Clipboard
  (Buffer             : access Gtk_Text_Buffer_Record;
   Clipboard          :      Gtk.Clipboard.Gtk_Clipboard);
```

Removes a Clipboard added with Add_Selection_Clipboard

```
procedure Cut_Clipboard
  (Buffer             : access Gtk_Text_Buffer_Record;
   Clipboard          :      Gtk.Clipboard.Gtk_Clipboard;
   Default_Editable   :      Boolean := True);
```

Copy the currently-selected text to the clipboard, then delete it if editable. Default_Editable: default editability of the buffer.

```
procedure Copy_Clipboard
  (Buffer             : access Gtk_Text_Buffer_Record;
   Clipboard          :      Gtk.Clipboard.Gtk_Clipboard);
```

Copy the currently-selected text to the clipboard.

```
procedure Paste_Clipboard
  (Buffer             : access Gtk_Text_Buffer_Record;
   Clipboard          :      Gtk.Clipboard.Gtk_Clipboard;
   Override_Location  :      Gtk.Text_Iter.Gtk_Text_Iter_Access
                      := null;
   Default_Editable   :      Boolean := True);
```

Paste the clipboard contents at the insertion point, or at Override_Location if this parameter is not null. (Note: pasting is asynchronous, that is, we'll ask for the paste data and return, and at some point later after the main loop runs, the paste data will be inserted.)

```
function Selection_Exists
  (Buffer             : access Gtk_Text_Buffer_Record)
   return Boolean;
```

Return True if some text in the buffer is currently selected.

```
procedure Select_Range
  (Buffer             : access Gtk_Text_Buffer_Record;
   Ins                :      Gtk.Text_Iter.Gtk_Text_Iter;
   Bound              :      Gtk.Text_Iter.Gtk_Text_Iter);
```

This function moves the "insert" and "selection_bound" marks simultaneously. If you move them in two steps with Move_Mark, you will temporarily

select region in between their old and new locations, which can be pretty inefficient since the temporarily-selected region will force stuff to be recalculated. This function moves them as a unit, which can be optimized.

```
procedure Get_Selection_Bounds
  (Buffer             : access Gtk_Text_Buffer_Record;
   Start              : out   Gtk.Text_Iter.Gtk_Text_Iter;
   The_End            : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Boolean);
```

Place the bounds of the selection in Start and End. If the selection
has length 0, then Start and End are filled in with the same value. Start and End will be in ascending order. Result: whether the selection has nonzero length.

```
function Delete_Selection
  (Buffer             : access Gtk_Text_Buffer_Record;
   Interactive        :      Boolean;
   Default_Editable   :      Boolean)
   return Boolean;
```

Delete the range between the "insert" and "selection_bound" marks,
that is, the currently-selected text. If Interactive is True, the editability of the selection will be considered (users can't delete uneditable text). Return value: whether there was a non-empty selection to delete.

180.3.9 User actions

```
procedure Begin_User_Action
  (Buffer             : access Gtk_Text_Buffer_Record);
```

Called to indicate that the buffer operations between here and a
call to End_User_Action are part of a single user-visible operation. The operations between Begin_User_Action and End_User_Action can then be grouped when creating an undo stack. Gtk_Text_Buffer maintains a count of calls to Begin_User_Action that have not been closed with a call to End_User_Action, and emits the "begin_user_action" and "end_user_action" signals only for the outermost pair of calls. This allows you to build user actions from other user actions.

The "interactive" buffer mutation functions, such as Insert_Interactive, automatically call begin/end user action around the buffer operations they perform, so there's no need to add extra calls if your user action consists solely of a single call to one of those functions.

```
procedure End_User_Action
  (Buffer             : access Gtk_Text_Buffer_Record);
```

Should be paired with a call to Begin_User_Action.
See that function for a full explanation.

181 Package Gtk.Text_Child

A GtkTextChildAnchor is a spot in the buffer where child widgets can be "anchored" (inserted inline, as if they were characters). The anchor can have multiple widgets anchored, to allow for multiple views.

181.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Text_Child_Anchor   (Package Gtk.Text_Child_Anchor)
```

181.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Text_Child_Anchor);
function Get_Type             return Glib.GType;
```

Return the internal value associated with a Gtk_Text_Child_Anchor.

```
function Get_Widgets
  (Anchor             : access Gtk_Text_Child_Anchor_Record)
   return Gtk.Widget.Widget_List.Glist;
```

Return the list of widgets attached at anchor. The returned list should be freed by the caller.

```
function Get_Deleted
  (Anchor             : access Gtk_Text_Child_Anchor_Record)
   return Boolean;
```

Determines whether a child anchor has been deleted from the buffer. Keep in mind that the child anchor will be unreferenced when removed from the buffer, so you need to hold your own reference (with Ref()) if you plan to use this function; otherwise all deleted child anchors will also be finalized.

182 Package Gtk.Text_Child_Anchor

183 Package Gtk.Text_Iter

A Gtk_Text_Iter represents a location in the text. It becomes invalid if the characters/pixmaps/widgets (indexable objects) in the text buffer are changed.

183.1 Types

type Data_Type **is private**;

type Gtk_Text_Char_Predicate **is access function**
(Ch : Gunichar; User_Data : Data_Type) **return** Boolean;

type Gtk_Text_Iter **is limited private**;

type Gtk_Text_Iter_Access **is access all** Gtk_Text_Iter;

type Gtk_Text_Search_Flags **is mod** 2 ** 8;

183.2 Subprograms

function Get_Type **return** Glib.GType;

Return the internal type used for a Gtk_Text_Iter

procedure Copy
(Source : Gtk_Text_Iter;
Dest : **out** Gtk_Text_Iter);

Create a copy of Source.

183.2.1 Characters and bytes

The basic component of a Gtk_Text_Buffer is a character. Since these are@* encoded in Unicode's UTF8, a character can be stored as multiple bytes in fact, and gtk+ therefore provides function to either take bytes or characters into account. The latter is generally the form that you should use in your applications

procedure Forward_Char
(Iter : **in out** Gtk_Text_Iter;
Result : **out** Boolean);

Move Iter forward by one character offset.
Note that images embedded in the buffer occupy 1 character slot, so Forward_Char may actually move onto an image instead of a character, if you have images in your buffer. If Iter is the end iterator or one character before it, Iter will now point at the end iterator, and Forward_Char returns False for convenience when writing loops.

```
procedure Backward_Char
  (Iter               : in out Gtk_Text_Iter;
   Result             : out   Boolean);
```

Move backward by one character offset.
Return True if movement was possible; if Iter was the first in the buffer (character offset 0), return False for convenience when writing loops.

```
procedure Forward_Chars
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out   Boolean);
```

Move Count characters if possible.
If Count would move past the start or end of the buffer, move to the start or end of the buffer). Result indicates whether the new position of Iter is different from its original position, and dereferenceable (the last iterator in the buffer is not dereferenceable). If Count is 0, this procedure does nothing and returns False.

```
procedure Backward_Chars
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out   Boolean);
```

Move Count characters backward, if possible.
If Count would move past the start or end of the buffer, moves to the start or end of the buffer). Result indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't move, or moved onto the end iterator, then False is returned. If Count is 0, the function does nothing and returns False.

```
procedure Set_Offset
  (Iter               : in out Gtk_Text_Iter;
   Char_Offset        :      Gint);

function Get_Offset
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Set or return the character offset of an iterator.
Each character in a Gtk_Text_Buffer has an offset, starting with 0 for the first character in the buffer. Use Gtk.Text_Buffer.Get_Iter_At_Offset to convert an offset back into an iterator.

183.2.2 Words

Characters are grouped together into words. Their exact definition@* depends on the current language (see Pango.Language).

```
function Starts_Word
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter begins a natural-language word.
Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms.

```
function Ends_Word
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter ends a natural-language word.
Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

```
function Inside_Word
  (Iter                :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter is inside a natural-language word (as opposed to
say inside some whitespace). Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

```
procedure Forward_Word_End
  (Iter                : in out Gtk_Text_Iter;
   Result              : out    Boolean);
```

Move forward to the next word end.
If Iter is currently on a word end, move forward to the next one after that. Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

```
procedure Forward_Word_Ends
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out    Boolean);
```

Call Forward_Word_End up to Count times.

```
procedure Forward_Visible_Word_End
  (Iter                : in out Gtk_Text_Iter;
   Result              : out    Boolean);
```

Moves forward to the next visible word end. (If Iter is currently on
word end, moves forward to the next one after that.) Word breaks are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

```
procedure Forward_Visible_Word_Ends
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out    Boolean);
```

Calls Forward_Visible_Word_End up to Count times

```
procedure Backward_Word_Start
  (Iter                : in out Gtk_Text_Iter;
   Result              : out    Boolean);
```

Move backward to the next word start.
If Iter is currently on a word start, move backward to the next one after that.

```
procedure Backward_Word_Starts
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out    Boolean);
```

Call Backward_Word_Start up to Count times.

```
procedure Backward_Visible_Word_Start
  (Iter                : in out Gtk_Text_Iter;
   Result              : out    Boolean);
```

Moves backward to the previous visible word start. (If Iter is currently
on a word start, moves backward to the next one after that.) Word breaks are determined

by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango word break algorithms).

```
procedure Backward_Visible_Word_Starts
  (Iter               :      Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : in out Boolean);
```

Move backward up to Count previous visible word starts.

183.2.3 Sentences

Words are then grouped together into sentences.

```
function Starts_Sentence
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter begins a sentence.
Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

```
function Ends_Sentence
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter ends a sentence.
Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

```
function Inside_Sentence
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Determine whether Iter is inside a sentence (as opposed to in between two sentences, e.g. after a period and before the first letter of the next sentence). Sentence boundaries are determined by Pango and should be correct for nearly any language (if not, the correct fix would be to the Pango text boundary algorithms).

```
procedure Forward_Sentence_End
  (Iter               : in out Gtk_Text_Iter;
   Result             : out    Boolean);
```

Move forward to the next sentence end.
If Iter is at the end of a sentence, move to the next end of sentence.

```
procedure Backward_Sentence_Start
  (Iter               : in out Gtk_Text_Iter;
   Result             : out    Boolean);
```

Move backward to the next sentence start.
If Iter is already at the start of a sentence, move backward to the next one.

```
procedure Forward_Sentence_Ends
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out    Boolean);
```

Call Forward_Sentence_End up to Count times.

```
procedure Backward_Sentence_Starts
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out    Boolean);
```

Call Backward_Sentence_Starts up to Count times.

183.2.4 Lines and paragraphs

Sentences are grouped together to form lines and paragraphs. The@* definition of these is language-dependent

```
procedure Set_Line
  (Iter               : in out Gtk_Text_Iter;
   Line_Number        :      Gint);

function Get_Line
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Set or return the line number containing the iterator.
Lines in a Gtk_Text_Buffer are numbered beginning with 0 for the first line in the buffer.

```
procedure Set_Line_Offset
  (Iter               : in out Gtk_Text_Iter;
   Char_On_Line       :      Gint);

function Get_Line_Offset
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Move Iter within a line, to a new character (not byte) offset.
The given character offset must be less than or equal to the number of characters in the line; if equal, Iter moves to the start of the next line. See Set_Line_Index if you have a byte index rather than a character offset. The first character on the line has offset 0.

```
procedure Set_Line_Index
  (Iter               : in out Gtk_Text_Iter;
   Byte_On_Line       :      Gint);

function Get_Line_Index
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Same as Set_Line_Offset, but work with a byte index.
The given byte index must be at the start of a character, it can't be in the middle of a UTF-8 encoded character. Remember that Gtk_Text_Buffer encodes text in UTF-8, and that characters can require a variable number of bytes to represent.

```
procedure Set_Visible_Line_Offset
  (Iter               : in out Gtk_Text_Iter;
   Char_On_Line       :      Gint);

function Get_Visible_Line_Offset
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Sets or returns the offset in characters from the start of the line to
the given Iter, not counting characters that are invisible due to tags with the "invisible" flag toggled on.

```
procedure Set_Visible_Line_Index
  (Iter               : in out Gtk_Text_Iter;
   Byte_On_Line       :      Gint);

function Get_Visible_Line_Index
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Set or returns the number of bytes from the start of the line to the
given Iter, not counting bytes that are invisible due to tags with the "invisible" flag toggled on.

```
function Starts_Line
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Return True if Iter begins a paragraph. i.e. if Get_Line_Offset would return 0. However this function is potentially more efficient than Get_Line_Offset because it doesn't have to compute the offset, it just has to see whether it's 0.

```
function Ends_Line
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Return True if Iter points to the start of the paragraph delimiter characters for a line (delimiters will be either a newline, a carriage return, a carriage return followed by a newline, or a Unicode paragraph separator character). Note that an iterator pointing to the ASCII.LF of a ASCII.CR & ASCII.LF pair will not be counted as the end of a line, the line ends before the ASCII.CR.

```
function Get_Chars_In_Line
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Return the number of characters in the line containing Iter, including the paragraph delimiters.

```
function Get_Bytes_In_Line
  (Iter               :      Gtk_Text_Iter)
   return Gint;
```

Return the number of bytes in the line containing Iter, including the paragraph delimiters.

```
procedure Forward_Line
  (Iter               : in out Gtk_Text_Iter;
   Result             : out    Boolean);
```

Move Iter to the start of the next line. Return True if there was a next line to move to, and False if iter was simply moved to the end of the buffer and is now not dereferenceable, or if Iter was already at the end of the buffer.

```
procedure Forward_Lines
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out    Boolean);
```

Call Forward_Line, up to Count times.

```
procedure Forward_Visible_Line
  (Iter               : in out Gtk_Text_Iter;
   Result             : out    Boolean);
```

Moves Iter to the start of the next visible line. Returns True if there was a next line to move to, and False if Iter was simply moved to the end of the buffer and is now not dereferenceable, or if Iter was already at the end of the buffer.

```
procedure Forward_Visible_Lines
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out    Boolean);
```

Moves Count visible lines forward, if possible (if Count would move past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't

move, or moved onto the end iterator, then False is returned. If Count is 0, the function does nothing and returns False. If Count is negative, moves backward by 0 - Count lines.

```
procedure Forward_To_Line_End
  (Iter               : in out Gtk_Text_Iter;
   Result             : out   Boolean);
```

Move the iterator to point to the paragraph delimiter characters,
which will be either a newline, a carriage return, a carriage return/newline in sequence, or the Unicode paragraph separator character. If the iterator is already at the paragraph delimiter characters, move to the paragraph delimiter characters for the next line.

```
procedure Backward_Line
  (Iter               : in out Gtk_Text_Iter;
   Result             : out   Boolean);
```

Move Iter to the start of the previous line.
Return True if Iter could be moved; i.e. if Iter was at character offset 0, this function returns False. Therefore if Iter was already on line 0, but not at the start of the line, Iter is snapped to the start of the line and the function returns True. (Note that this implies that in a loop calling this function, the line number may not change on every iteration, if your first iteration is on line 0)

```
procedure Backward_Lines
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out   Boolean);
```

Call Backward_Line, up to Count times.

```
procedure Backward_Visible_Line
  (Iter               : in out Gtk_Text_Iter;
   Result             : out   Boolean);
```

Moves Iter to the start of the previous visible line. Returns True if
Iter could be moved; i.e. if Iter was at character offset 0, this function returns False. Therefore if Iter was already on line 0, but not at the start of the line, Iter is snapped to the start of the line and the function returns True. (Note that this implies that in a loop calling this function, the line number may not change on every iteration, if your first iteration is on line 0.)

```
procedure Backward_Visible_Lines
  (Iter               : in out Gtk_Text_Iter;
   Count              :      Gint := 1;
   Result             : out   Boolean);
```

Moves Count visible lines backward, if possible (if Count would move
past the start or end of the buffer, moves to the start or end of the buffer). The return value indicates whether the iterator moved onto a dereferenceable position; if the iterator didn't move, or moved onto the end iterator, then False is returned. If Count is 0, the function does nothing and returns False. If Count is negative, moves forward by 0 - Count lines.

183.2.5 Buffer

When grouped together, lines and paragraph made up the whole buffer.

```
function Is_End
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Return True if Iter is the end iterator.
i.e. one past the last dereferenceable iterator in the buffer. This is the most efficient way to check whether an iterator is the end iterator.

```
function Is_Start
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Return True if Iter is the first iterator in the buffer, that is
if Iter has a character offset of 0.

```
procedure Forward_To_End
  (Iter               : in out Gtk_Text_Iter);
```

Move Iter forward to the "end iterator", which points one past the last
valid character in the buffer. Get_Char called on the end iterator returns 0, which is convenient for writing loops.

183.2.6 Reading buffer contents

```
function Get_Char
  (Iter               :      Gtk_Text_Iter)
   return Gunichar;
```

Return the character immediately following Iter. If Iter is at the
end of the buffer, then return ASCII.NUL.

```
function Get_Char
  (Iter               :      Gtk_Text_Iter)
   return Character;
```

Return the character immediately following Iter. If Iter is at the
end of the buffer, then return ASCII.NUL. Note that this function assumes that the text is encoded in ASCII format. If this is not the case, use the Get_Char function that returns a Gunichar instead.

```
function Get_Slice
  (Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return UTF8_String;
```

Return the text in the given range.
A "slice" is an array of characters encoded in UTF-8 format, including the Unicode "unknown" character 16#FFFC# for iterable non-character elements in the buffer, such as images. Because images are encoded in the slice, byte and character offsets in the returned array will correspond to byte offsets in the text buffer. Note that 16#FFFC# can occur in normal text as well, so it is not a reliable indicator that a pixbuf or widget is in the buffer.

```
function Get_Slice
  (Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return Interfaces.C.Strings.chars_ptr;
```

Same as above, but returns the row C string.
The caller is responsible for freeing the string returned.

```
function Get_Text
  (Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return UTF8_String;
```

Return text in the given range.
If the range contains non-text elements such as images, the character and byte offsets in

the returned string will not correspond to character and byte offsets in the buffer. If you want offsets to correspond, see Get_Slice.

```
function Get_Visible_Slice
  (Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return UTF8_String;
```

Like Get_Slice, but invisible text is not included.
Invisible text is usually invisible because a Gtk_Text_Tag with the "invisible" attribute turned on has been applied to it.

```
function Get_Visible_Text
  (Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return UTF8_String;
```

Like Get_Text, but invisible text is not included.
Invisible text is usually invisible because a Gtk_Text_Tag with the "invisible" attribute turned on has been applied to it.

```
function Get_Pixbuf
  (Iter               :      Gtk_Text_Iter)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

If the location pointed to by Iter contains a pixbuf, the pixbuf
is returned (with no new reference count added). Otherwise, null is returned.

183.2.7 Tags

Iterators can be used to move among tags. These tags are used to@* set some specific attributes on the text.

```
function Begins_Tag
  (Iter               :      Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                      := null)
   return Boolean;
```

Return True if Tag is toggled on at exactly this point.
If Tag is null, return True if any tag is toggled on at this point. Return True if Iter is the start of the tagged range; Has_Tag tells you whether an iterator is within a tagged range.

```
function Ends_Tag
  (Iter               :      Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                      := null)
   return Boolean;
```

Return True if Tag is toggled off at exactly this point.
If Tag is null, return True if any tag is toggled off at this point. Note that the Ends_Tag return True if Iter is the end of the tagged range; Has_Tag tells you whether an iterator is within a tagged range.

```
function Toggles_Tag
  (Iter               :      Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                      := null)
   return Boolean;
```

Whether a range with Tag applied to it begins or ends at Iter.
Equivalent to "Begins_Tag (Iter, Tag) or else Ends_Tag (Iter, Tag)".

```
function Has_Tag
  (Iter               :      Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                       := null)
   return Boolean;
```

Return True if Iter is within a range tagged with Tag.

```
function Get_Tags
  (Iter               :      Gtk_Text_Iter)
   return Gtk.Text_Tag.Text_Tag_List.GSlist;
```

Return a list of tags that apply to Iter, in ascending order of priority (highest-priority tags are last). The Gtk_Text_Tag in the list don't have a reference added, but you have to free the list itself.

```
procedure Forward_To_Tag_Toggle
  (Iter               : in out Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                       := null;
   Result             : out   Boolean);
```

Move forward to the next toggle (on or off) of the Gtk_Text_Tag Tag, or to the next toggle of any tag if Tag is null. If no matching tag toggles are found, return False, otherwise True. Do not return toggles located at Iter, only toggles after Iter. Set Iter to the location of the toggle, or to the end of the buffer if no toggle is found.

```
procedure Backward_To_Tag_Toggle
  (Iter               : in out Gtk_Text_Iter;
   Tag                :      Gtk.Text_Tag.Gtk_Text_Tag
                       := null;
   Result             : out   Boolean);
```

Move backward to the next toggle (on or off) of the Gtk_Text_Tag Tag, or to the next toggle of any tag if Tag is null. If no matching tag toggles are found, return False, otherwise True. Do not return toggles located at Iter, only toggles before Iter. Set Iter to the location of the toggle, or the start of the buffer if no toggle is found.

```
function Get_Toggled_Tags
  (Iter               :      Gtk_Text_Iter;
   Toggled_On         :      Boolean)
   return Glib.Object.Object_List.GSlist;
```

Returns a list of #GtkTextTag that are toggled on or off at this point. (If Toggled_On is True, the list contains tags that are toggled on.) If a tag is toggled on at Iter, then some non-empty range of characters following Iter has that tag applied to it. If a tag is toggled off, then some non-empty range following Iter does not have the tag applied to it. The returned list should be freed by the caller.

183.2.8 Attributes

The tags are used to change the attributes of parts of the buffer. For@* convenience, a number of wrapper subprograms are provided to make the use of tags easier.

```
function Editable
  (Iter               :      Gtk_Text_Iter;
   Default_Setting    :      Boolean := True)
   return Boolean;
```

Return whether Iter is within an editable region of text. Non-editable text is "locked" and can't be changed by the user via Gtk_Text_View. This

function is simply a convenience wrapper around Get_Attributes. If no tags applied to this text affect editability, Default_Setting will be returned.

```
function Can_Insert
  (Iter               :      Gtk_Text_Iter;
   Default_Editability  :      Boolean)
   return Boolean;
```

Return whether text inserted at Iter would be editable.
Considering the default editability of the buffer, and tags that affect editability, determines whether text inserted at Iter would be editable. If text inserted at Iter would be editable then the user should be allowed to insert text at Iter. Gtk.Text_Buffer.Insert_Interactive uses this function to decide whether insertions are allowed at a given position.

```
function Get_Language
  (Iter               :      Gtk_Text_Iter)
   return UTF8_String;
```

A convenience wrapper around Get_Attributes,
which returns the language in effect at Iter. If no tags affecting language apply to Iter, the return value is identical to that of Gtk.Get_Default_Language.

```
procedure Get_Attributes
  (Iter               :      Gtk_Text_Iter;
   Values             : in out Gtk.Text_Attributes.Gtk_Text_Attributes;
   Modified           : out   Boolean);
```

Computes the effect of any tags applied to this spot in the text. The
Values parameter should be initialized to the default settings you wish to use if no tags are in effect. You'd typically obtain the defaults from gtk.text_view.get_default_attributes.

Get_Attributes will modify Values, applying the effects of any tags present at Iter. If any tags affected Values, the function returns True.

183.2.9 Cursor

The cursor is a special position in the buffer that indicates where the@* user will interactively insert new characters. In some languages, you can put the cursor between certain chars. Also you can't put the cursor between \r and \n on Windows-line ending files.

```
function Is_Cursor_Position
  (Iter               :      Gtk_Text_Iter)
   return Boolean;
```

Return True if the cursor can be placed at Iter.
See Forward_Cursor_Position for details on what a cursor position is.

```
procedure Forward_Cursor_Position
  (Iter               : in out Gtk_Text_Iter;
   Result             : out   Boolean);
```

Move Iter forward by a single cursor position.
Cursor positions are (unsurprisingly) positions where the cursor can appear. Perhaps surprisingly, there may not be a cursor position between all characters. The most common example for European languages would be a carriage return/newline sequence. For some Unicode characters, the equivalent of say the letter "a" with an accent mark will be represented as two characters, first the letter then a "combining mark" that causes the accent to be rendered; so the cursor can't go between those two characters.

```
procedure Backward_Cursor_Position
  (Iter               : in out Gtk_Text_Iter;
```

```
   Result              : out   Boolean);
```

Like Forward_Cursor_Position, but moves backward.

```
procedure Forward_Cursor_Positions
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out   Boolean);
```

Call Forward_Cursor_Position up to Count times.

```
procedure Forward_Visible_Cursor_Position
  (Iter                : in out Gtk_Text_Iter;
   Result              : out   Boolean);
```

Moves Iter forward to the next visible cursor position. Return True if the new position is valid

```
procedure Forward_Visible_Cursor_Positions
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out   Boolean);
```

Moves up to Count visible cursor positions. See Forward_Cursor_Position for details. Return True if the cursor could be moved.

```
procedure Backward_Cursor_Positions
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out   Boolean);
```

Call Backward_Cursor_Position up to Count times.

```
procedure Backward_Visible_Cursor_Position
  (Iter                : in out Gtk_Text_Iter;
   Result              : out   Boolean);
```

Moves Iter backward to the previous visible cursor position. Return True if the new position is valid.

```
procedure Backward_Visible_Cursor_Positions
  (Iter                : in out Gtk_Text_Iter;
   Count               :      Gint := 1;
   Result              : out   Boolean);
```

Moves up to Count visible cursor positions. Return True if the new position is valid.

183.2.10 Children

The buffer can contain many widgets. They are all attached to specific@* anchors (see Gtk.Text_Child)

```
function Get_Child_Anchor
  (Iter                :      Gtk_Text_Iter)
   return Gtk.Text_Child.Gtk_Text_Child_Anchor;
```

If the location pointed to by Iter contains a child anchor, the anchor is returned (with no new reference count added). Otherwise, null is returned.

```
function Get_Marks
  (Iter                :      Gtk_Text_Iter)
   return Glib.Object.Object_List.GSlist;
```

Returns a list of all Gtk_Text_Mark at this location. Because marks are not iterable (they don't take up any "space" in the buffer, they are just marks in between iterable locations), multiple marks can exist in the same place. The returned list is not in any meaningful order.

183.2.11 Searching

```
procedure Forward_Search
  (Iter               :      Gtk_Text_Iter;
   Str                :      UTF8_String;
   Flags              :      Gtk_Text_Search_Flags;
   Match_Start        : out  Gtk_Text_Iter;
   Match_End          : out  Gtk_Text_Iter;
   Limit              :      Gtk_Text_Iter;
   Result             : out  Boolean);
```

Search forward for Str.
Any match is returned as the range Match_Start, Match_End. If you specify Visible_Only or Slice, the match may have invisible text, pixbufs, or child widgets interspersed in Str. Iter: start of search Str: a search string Match_Start: return location for start of match, or null Match_End: return location for end of match, or null Limit: bound for the search, or null for the end of the buffer Result: whether a match was found.

```
procedure Backward_Search
  (Iter               :      Gtk_Text_Iter;
   Str                :      UTF8_String;
   Flags              :      Gtk_Text_Search_Flags;
   Match_Start        : out  Gtk_Text_Iter;
   Match_End          : out  Gtk_Text_Iter;
   Limit              :      Gtk_Text_Iter;
   Result             : out  Boolean);
```

Same as Forward_Search, but move backward.

```
function Forward_Find_Char
  (Iter               :      Gtk_Text_Iter;
   Pred               :      Gtk_Text_Char_Predicate;
   User_Data          :      Data_Type;
   Limit              :      Gtk_Text_Iter)
   return Boolean;
```

Advances Iter, calling Pred on each character. If Pred returns True,
returns True and stops scanning. If Pred never returns True, Iter is set to Limit if Limit is not Null_Iter, otherwise to the end iterator.

```
function Backward_Find_Char
  (Iter               :      Gtk_Text_Iter;
   Pred               :      Gtk_Text_Char_Predicate;
   User_Data          :      Data_Type;
   Limit              :      Gtk_Text_Iter)
   return Boolean;
```

Same as Forward_Find_Char, but goes backward from Iter

183.2.12 Comparisons

```
function Equal
  (Lhs                :      Gtk_Text_Iter;
   Rhs                :      Gtk_Text_Iter)
   return Boolean;
```

Test whether two iterators are equal, using the fastest possible
mechanism. This function is very fast; you can expect it to perform better than e.g. getting the character offset for each iterator and comparing the offsets yourself. Also, it's a bit faster than Compare.

```
function Compare
```

```
(Lhs                 :      Gtk_Text_Iter;
 Rhs                 :      Gtk_Text_Iter)
 return Gint;
```

A quick sort-style function that return negative if Lhs is less than
Rhs, positive if Lhs is greater than Rhs, and 0 if they're equal. Ordering is in character offset order, i.e. the first character in the buffer is less than the second character in the buffer.

```
function In_Range
  (Iter               :      Gtk_Text_Iter;
   Start              :      Gtk_Text_Iter;
   The_End            :      Gtk_Text_Iter)
   return Boolean;
```

Start and End must be in order, unlike most text buffer functions, for
efficiency reasons. Return True if Iter falls in the range [Start, End)

```
procedure Order
  (First              : in out Gtk_Text_Iter;
   Second             : in out Gtk_Text_Iter);
```

Swap the value of First and Second if Second comes before First in the
buffer. That is, ensures that First and Second are in sequence. Most text buffer functions that take a range call this automatically on your behalf, so there's no real reason to call it yourself in those cases. There are some exceptions, such as In_Range, that expect a pre-sorted range.

183.2.13 Converting to/from GValue

```
procedure Set_Text_Iter
  (Val                : in out Glib.Values.GValue;
   Iter               :      Gtk_Text_Iter);
```

Set the value of the given GValue to Iter.
Note that Iter is stored by reference, which means no copy of Iter is made. Iter should remain allocated as long as Val is being used.

```
procedure Get_Text_Iter
  (Val                :      Glib.Values.GValue;
   Iter               : out   Gtk_Text_Iter);
```

Extract the iterator from the given GValue.
Note that the iterator returned is a copy of the iterator referenced by the give GValue. Modifying the iterator returned does not modify the iterator referenced by the GValue.

183.2.14 Moving around the buffer

```
function C_Gtk_Text_Iter_Size  return Gint;
```

184 Package Gtk.Text_Mark

Marks are positions in a buffer which move when the buffer is modified, so that they always point to the same place in the buffer. They are automatically destroyed when the buffer is destroyed, unless you have explicitly call Ref on the mark. See Gtk.Text_Buffer for various functions dealing with marks. In particular, Gtk.Text_Buffer.Get_Buffer can be used to retrieve the buffer from a mark.

184.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Text_Mark         (Package Gtk.Text_Mark)
```

184.2 Subprograms

```
function Get_Type              return Glib.GType;
```

Return the internal value associated with a Gtk_Label.

```
procedure Set_Visible
  (Mark               : access Gtk_Text_Mark_Record;
   Setting            :      Boolean := True);
function Get_Visible
  (Mark               : access Gtk_Text_Mark_Record)
   return Boolean;
```

Set the visibility of Mark.
The insertion point is normally visible, i.e. you can see it as a vertical bar. Also, the text widget uses a visible mark to indicate where a drop will occur when dragging-and-dropping text. Most other marks are not visible. Marks are not visible by default.

```
function Get_Name
  (Mark               : access Gtk_Text_Mark_Record)
   return String;
```

Return the mark name; Return "" for anonymous marks.

```
function Get_Deleted
  (Mark               : access Gtk_Text_Mark_Record)
   return Boolean;
```

Returns True if the mark has been removed from its buffer with Gtk.Text_Buffer.Delete_Mark. Marks can't be used once deleted.

```
function Get_Left_Gravity
  (Mark               : access Gtk_Text_Mark_Record)
   return Boolean;
```

Return True if the mark has left gravity, False otherwise.

184.2.1 Converting to/from GValue

```
procedure Set_Text_Mark
  (Val                : in out Glib.Values.GValue;
   Mark               : access Gtk_Text_Mark_Record);
function Get_Text_Mark
  (Val                :      Glib.Values.GValue)
   return Gtk_Text_Mark;
```

Set the value of the given GValue to Mark.

185 Package Gtk.Text_Tag

A tag is a set of properties that can be associated with a range of text. See also Gtk.Text_Attributes. Tags should be in a Gtk_Text_Tag_Table for a given before before they are used in that buffer.

185.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Text_Tag          (Package Gtk.Text_Tag)
```

185.2 Signals

- **"event"**

```
function Handler
(Tag          : access Gtk_Text_Tag_Record'Class;
Event_Object : out GObject;
Event        : Gdk.Event.Gdk_Event;
Iter         : access Gtk.Text_Iter.Gtk_Text_Iter_Record'Class)
return Gint;
```

???

185.3 Subprograms

```
function Convert
  (W                  :      Gtk_Text_Tag)
   return System.Address;
function Convert
  (W                  :      System.Address)
   return Gtk_Text_Tag;
procedure Gtk_New
  (Widget             : out   Gtk_Text_Tag;
   Name               :      String := "");
```

Create a new Gtk_Text_Tag.
Newly created tags must be added to the tags table for the buffer you intend to use them in. Gtk.Text_Tag_Table.Add (Get_Tag_Table (Buffer), Tag); See also Gtk.Text_Buffer.Create_Tag which is a more convenient way of creating a tag.

```
function Get_Type              return Glib.GType;
```

Return the internal value associated with this widget.

```
procedure Set_Priority
  (Tag                : access Gtk_Text_Tag_Record;
   Priority           :      Gint);
function Get_Priority
  (Tag                : access Gtk_Text_Tag_Record)
   return Gint;
```

Set the priority of a Gtk_Text_Tag.
Valid priorities start at 0 and go to one less than Table_Size. Each tag in a table has a unique priority; setting the priority of one tag shifts the priorities of all the other tags in the table to maintain a unique priority for each tag. Higher priority tags "win" if two tags both set the same text attribute. When adding a tag to a tag table, it will be assigned the

highest priority in the table by default; so normally the precedence of a set of tags is the order in which they were added to the table, or created with Gtk.Text_Buffer.Create_Tag, which adds the tag to the buffer's table automatically.

186 Package Gtk.Text_Tag_Table

A table is a collection of tags where you can Add, Remove, Lookup or traverse (Foreach) a tag.

186.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Text_Tag_Table    (Package Gtk.Text_Tag_Table)
```

186.2 Signals

- **"tag_added"**

```
procedure Handler
(Widget : access Gtk_Text_Tag_Table_Record'Class;
Tag    : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class);
```

- **"tag_changed"**

```
procedure Handler
(Widget       : access Gtk_Text_Tag_Table_Record'Class;
Tag          : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
Size_Changed : Boolean);
```

- **"tag_removed"**

```
procedure Handler
(Widget : access Gtk_Text_Tag_Table_Record'Class;
Tag    : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class);
```

186.3 Types

type Data_Type **is private**;

type Data_Type_Access **is access all** Data_Type;

```
type Gtk_Text_Tag_Table_Proc is access procedure
          (Tag  : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class;
           Data : Data_Type_Access);
```

186.4 Subprograms

```
procedure Gtk_New
  (Table              : out   Gtk_Text_Tag_Table);
```

Create a new Text_Tag_Table.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Text_Tag_Table.

```
procedure Add
  (Table              : access Gtk_Text_Tag_Table_Record;
   Tag                : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class);
```

Add a tag to the table.
The tag is assigned the highest priority in the table. You must Unref the Tag on exit

```
procedure Remove
  (Table              : access Gtk_Text_Tag_Table_Record;
   Tag                : access Gtk.Text_Tag.Gtk_Text_Tag_Record'Class);
```

Remove a tag from the table.
This will remove the table's reference to the tag, so be careful - the tag will end up destroyed if you don't have a reference to it.

```
function Lookup
  (Table              : access Gtk_Text_Tag_Table_Record;
   Name               :      String)
   return Gtk.Text_Tag.Gtk_Text_Tag;
```

Look up a named tag.
Return the tag or null if none by that name is in the table.

```
function Get_Size
  (Table              : access Gtk_Text_Tag_Table_Record)
   return Gint;
```

Return the size of the table (number of tags).

```
procedure Foreach
  (Table              : access Gtk_Text_Tag_Table_Record;
   Proc               :      Gtk_Text_Tag_Table_Proc;
   Data               :      Data_Type_Access);
```

Call Proc on each tag in Table, with user data Data.

187 Package Gtk.Text_View

This widget displays a view of a Gtk_Text_Buffer. Multiple views can be set on a given buffer.

187.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Text_View   (Package Gtk.Text_View)
```

187.2 Signals

- **"copy_clipboard"**

```
procedure Handler (Widget : access Gtk_Text_View_Record'Class);
```

- **"cut_clipboard"**

```
procedure Handler (Widget : access Gtk_Text_View_Record'Class);
```

- **"delete_from_cursor"**

```
procedure Handler
(Widget   : access Gtk_Text_View_Record'Class;
The_Type : Gtk_Delete_Type;
Count    : Gint);
```

- **"insert_at_cursor"**

```
procedure Handler
(Widget : access Gtk_Text_View_Record'Class; Str : UTF8_String);
```

- **"move_cursor"**

```
procedure Handler
(Widget           : access Gtk_Text_View_Record'Class;
Step             : Gtk_Movement_Step;
Count            : Gint;
Extend_Selection : Boolean);
```

- **"paste_clipboard"**

```
procedure Handler (Widget : access Gtk_Text_View_Record'Class);
```

- **"populate_popup"**

```
procedure Handler
(Widget : access Gtk_Text_View_Record'Class;
Menu   : access Gtk.Menu.Gtk_Menu_Record'Class);
```

- **"set_anchor"**

```
procedure Handler (Widget : access Gtk_Text_View_Record'Class);
```

- **"set_scroll_adjustments"**

```
procedure Handler
(Widget      : access Gtk_Text_View_Record'Class;
Hadjustment : access Gtk.Adjustment.Gtk_Adjustment_Record'Class;
Vadjustment : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
```

- **"toggle_overwrite"**

```
procedure Handler (Widget : access Gtk_Text_View_Record'Class);
```

187.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Text_View;
   Buffer             :       Gtk.Text_Buffer.Gtk_Text_Buffer
                       := null);

function Get_Type                return Glib.GType;
```

Return the internal value associated with this widget.

```
procedure Set_Buffer
  (Text_View          : access Gtk_Text_View_Record;
   Buffer             : access Gtk.Text_Buffer.Gtk_Text_Buffer_Record'Class);
```

Set Buffer as the buffer being displayed by Text_View.
The previous buffer displayed by the text view is unreferenced, and a reference is added to Buffer. If you owned a reference to Buffer before passing it to this function, you must remove that reference yourself; Gtk_Text_View will not "adopt" it.

```
function Get_Buffer
  (Text_View          : access Gtk_Text_View_Record)
   return Gtk.Text_Buffer.Gtk_Text_Buffer;
```

Return the Gtk_Text_Buffer being displayed by this text view.
The reference count on the buffer is not incremented; the caller of this function won't own a new reference.

```
function Scroll_To_Iter
  (Text_View          : access Gtk_Text_View_Record;
   Iter               :       Gtk.Text_Iter.Gtk_Text_Iter;
   Within_Margin      :       Gdouble;
   Use_Align          :       Boolean;
   Xalign             :       Gdouble;
   Yalign             :       Gdouble)
   return Boolean;
```

Scroll Text_View so that Iter is on the screen in the position
indicated by Xalign and Yalign. An alignment of 0.0 indicates left or top, 1.0 indicates right or bottom, 0.5 means center. If Use_Align is False, the text scrolls the minimal distance to get the mark onscreen, possibly not scrolling at all. The effective screen for purposes of this function is reduced by a margin of size Within_Margin. Note: This function uses the currently-computed height of the lines in the text buffer. Note that line heights are computed in an idle handler; so this function may not have the desired effect if it's called before the height computations. To avoid oddness, consider using Scroll_To_Mark which saves a point to be scrolled to after line validation.

```
procedure Scroll_To_Mark
  (Text_View          : access Gtk_Text_View_Record;
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class;
   Within_Margin      :       Gdouble := 0.0;
   Use_Align          :       Boolean := False;
   Xalign             :       Gdouble := 0.0;
   Yalign             :       Gdouble := 0.0);
```

Scroll Text_View so that Mark is on the screen in the position indicated
by Xalign and Yalign. An alignment of 0.0 indicates left or top, 1.0 indicates right or bottom, 0.5 means center. If Use_Align is False, the text scrolls the minimal distance to get the mark onscreen, possibly not scrolling at all. The effective screen for purposes of this function is reduced by a margin of size Within_Margin.

```
procedure Scroll_Mark_Onscreen
  (Text_View          : access Gtk_Text_View_Record;
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class);
```

Same as the above with the default values

```
function Move_Mark_Onscreen
  (Text_View          : access Gtk_Text_View_Record;
   Mark               : access Gtk.Text_Mark.Gtk_Text_Mark_Record'Class)
   return Boolean;
```

Move a mark within the buffer so that it's located within the currently-visible text area. Return value: True if the mark moved (wasn't already onscreen).

```
function Place_Cursor_Onscreen
  (Text_View          : access Gtk_Text_View_Record)
   return Boolean;
```

Move the cursor to the currently visible region of the buffer, if it isn't there already. Return value: True if the cursor had to be moved.

```
procedure Get_Visible_Rect
  (Text_View          : access Gtk_Text_View_Record;
   Visible_Rect       : out   Gdk.Rectangle.Gdk_Rectangle);
```

Fill Visible_Rect with the currently-visible region of the buffer, in buffer coordinates. Convert to window coordinates with Buffer_To_Window_Coords.

```
procedure Get_Iter_Location
  (Text_View          : access Gtk_Text_View_Record;
   Iter               :      Gtk.Text_Iter.Gtk_Text_Iter;
   Location           : out   Gdk.Rectangle.Gdk_Rectangle);
```

Get a rectangle which roughly contains the character at iter. The rectangle position is in buffer coordinates; use Buffer_To_Window_Coords to convert these coordinates to coordinates for one of the windows in the text view.

```
procedure Get_Iter_At_Location
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   X                  :      Gint;
   Y                  :      Gint);
```

Retrieve the iterator at buffer coordinates X and Y. Buffer coordinates are coordinates for the entire buffer, not just the currently-displayed portion. If you have coordinates from an event, you have to convert those to buffer coordinates with Window_To_Buffer_Coords.

```
procedure Get_Iter_At_Position
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Trailing           : out   Gint;
   X                  :      Gint;
   Y                  :      Gint);
```

Retrieves the iterator pointing to the character at buffer coordinates X and Y. Buffer coordinates are coordinates for the entire buffer, not just the currently-displayed portion. If you have coordinates from an event, you have to convert those to buffer coordinates with Window_To_Buffer_Coords. Note that this is different from Get_Iter_At_Location(), which returns cursor locations, i.e. positions between characters) Trailing is set to indicate where in the grapheme the user clicked. It will be either 0, or the number of characters in the grapheme. 0 represents the trailing edge of the grapheme.

```
procedure Get_Line_Yrange
  (Text_View          : access Gtk_Text_View_Record;
   Iter               :      Gtk.Text_Iter.Gtk_Text_Iter;
   Y                  : out   Gint;
   Height             : out   Gint);
```

Get the Y coordinate of the top of the line containing Iter,
and the Height of the line. The coordinate is a buffer coordinate; convert to window coordinates with Buffer_To_Window_Coords.

```
procedure Get_Line_At_Y
  (Text_View          : access Gtk_Text_View_Record;
   Target_Iter        : out   Gtk.Text_Iter.Gtk_Text_Iter;
   Y                  :      Gint;
   Line_Top           : out   Gint);
```

Get the Gtk_Text_Iter at the start of the line containing the
coordinate Y. Y is in buffer coordinates, convert from window coordinates with Window_To_Buffer_Coords. Line_Top will be filled with the coordinate of the top edge of the line.

```
procedure Buffer_To_Window_Coords
  (Text_View          : access Gtk_Text_View_Record;
   Win                :      Gtk.Enums.Gtk_Text_Window_Type;
   Buffer_X           :      Gint;
   Buffer_Y           :      Gint;
   Window_X           : out   Gint;
   Window_Y           : out   Gint);
```

Convert coordinate (Buffer_X, Buffer_Y) to coordinates for the window
Win, and store the result in (Window_X, Window_Y).

```
procedure Window_To_Buffer_Coords
  (Text_View          : access Gtk_Text_View_Record;
   Win                :      Gtk.Enums.Gtk_Text_Window_Type;
   Window_X           :      Gint;
   Window_Y           :      Gint;
   Buffer_X           : out   Gint;
   Buffer_Y           : out   Gint);
```

Convert coordinates on the window identified by Win to buffer
coordinates, storing the result in (Buffer_X, Buffer_Y).

```
function Get_Window
  (Text_View          : access Gtk_Text_View_Record;
   Win                :      Gtk.Enums.Gtk_Text_Window_Type)
   return Gdk.Window.Gdk_Window;
```

Retrieve the Gdk_Window corresponding to an area of the text view;
possible windows include the overall widget window, child windows on the left, right, top, bottom, and the window that displays the text buffer. Windows are null and nonexistent if their width or height is 0, and are nonexistent before the widget has been realized.

```
function Get_Window_Type
  (Text_View          : access Gtk_Text_View_Record;
   Window             :      Gdk.Window.Gdk_Window)
   return Gtk.Enums.Gtk_Text_Window_Type;
```

Usually used to find out which window an event corresponds to.
If you connect to an event signal on Text_View, this function should be called on Get_Window (Event) to see which window it was.

```
procedure Set_Border_Window_Size
  (Text_View          : access Gtk_Text_View_Record;
   The_Type           :      Gtk.Enums.Gtk_Text_Window_Type;
   Size               :      Gint);
function Get_Border_Window_Size
  (Text_View          : access Gtk_Text_View_Record;
   The_Type           :      Gtk.Enums.Gtk_Text_Window_Type)
   return Gint;
```

Set the width of Text_Window_Left or Text_Window_Right,
or the height of Text_Window_Top or Text_Window_Bottom. Automatically destroy the corresponding window if the size is set to 0, and create the window if the size is set to non-zero.

187.3.1 Iterators

You can manipulate iterators either through the buffer directly (thus@* independently of any display properties), or through the property (if you need to reference to what the user is actually seeing on the screen)

```
procedure Forward_Display_Line
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Boolean);
procedure Forward_Display_Line_End
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Boolean);
```

Moves the given Iter forward by one display (wrapped) line. A
display line is different from a paragraph. Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the Gtk_Text_Buffer. Returns True if Iter was moved and is not on the end iterator.

```
procedure Backward_Display_Line
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Boolean);
procedure Backward_Display_Line_Start
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Result             : out   Boolean);
```

Moves the given Iter backward by one display (wrapped) line. A display
line is different from a paragraph. Paragraphs are separated by newlines or other paragraph separator characters. Display lines are created by line-wrapping a paragraph. If wrapping is turned off, display lines and paragraphs will be the same. Display lines are divided differently for each view, since they depend on the view's width; paragraphs are the same in all views, since they depend on the contents of the Gtk_Text_Buffer. Returns True if Iter was moved and is not on the end iterator

```
function Starts_Display_Line
  (Text_View          : access Gtk_Text_View_Record;
   Iter               :      Gtk.Text_Iter.Gtk_Text_Iter)
```

```
   return Boolean;
```

Determines whether Iter is at the start of a display line. See Forward_Display_Line for an explanation of display lines vs. paragraphs. Returns true if Iter begins a wrapped line.

```
procedure Move_Visually
  (Text_View          : access Gtk_Text_View_Record;
   Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
   Count              :      Gint;
   Result             : out   Boolean);
```

Move the iterator a given number of characters visually, treating it as the strong cursor position. If Count is positive, then the new strong cursor position will be Count positions to the right of the old cursor position. If Count is negative then the new strong cursor position will be Count positions to the left of the old cursor position.

In the presence of bidirection text, the correspondence between logical and visual order will depend on the direction of the current run, and there may be jumps when the cursor is moved off of the end of a run.

Returns True if Iter moved and is not on the end iterator

187.3.2 Children widgets

Any widget can be put in a text_view, for instance to provide an@* interactive area.

```
procedure Add_Child_In_Window
  (Text_View          : access Gtk_Text_View_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Which_Window       :      Gtk.Enums.Gtk_Text_Window_Type;
   Xpos               :      Gint;
   Ypos               :      Gint);
```

Adds a child at fixed coordinates in one of the text widget's windows. The window must have nonzero size (see Set_Border_Window_Size). Note that the child coordinates are given relative to the Gdk_Window in question, and that these coordinates have no sane relationship to scrolling. When placing a child in GTK_TEXT_WINDOW_WIDGET, scrolling is irrelevant, the child floats above all scrollable areas. But when placing a child in one of the scrollable windows (border windows or text window), you'll need to compute the child's correct position in buffer coordinates any time scrolling occurs or buffer changes occur, and then call Move_Child() to update the child's position. Unfortunately there's no good way to detect that scrolling has occurred, using the current API; a possible hack would be to update all child positions when the scroll adjustments change or the text buffer changes.

```
procedure Add_Child_At_Anchor
  (Text_View          : access Gtk_Text_View_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Anchor             : access Gtk.Text_Child.Gtk_Text_Child_Anchor_Record'Class);
```

Adds a child widget in the text buffer, at the given Anchor.

```
procedure Move_Child
  (Text_View          : access Gtk_Text_View_Record;
   Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
   Xpos               :      Gint;
   Ypos               :      Gint);
```

Updates the position of a child, as for Add_Child_In_Window. Child must already have been added to the text_view.

187.3.3 Attributes

```
function Get_Default_Attributes
  (Text_View          : access Gtk_Text_View_Record)
   return Gtk.Text_Attributes.Gtk_Text_Attributes;
```

Obtains a copy of the default text attributes. These are the attributes
used for text unless a tag overrides them. You'd typically pass the default attributes in to Gtk.Text_Iter.Get_Attributes in order to get the attributes in effect at a given text position. The returned value is a copy and should be freed by the caller.

```
procedure Set_Cursor_Visible
  (Text_View          : access Gtk_Text_View_Record;
   Setting            :      Boolean := True);
function Get_Cursor_Visible
  (Text_View          : access Gtk_Text_View_Record)
   return Boolean;
```

Toggle whether the insertion point is displayed.
A buffer with no editable text probably shouldn't have a visible cursor, so you may want to turn the cursor off.

```
procedure Set_Wrap_Mode
  (Text_View          : access Gtk_Text_View_Record;
   Wrap_Mode          :      Gtk.Enums.Gtk_Wrap_Mode);
function Get_Wrap_Mode
  (Text_View          : access Gtk_Text_View_Record)
   return Gtk.Enums.Gtk_Wrap_Mode;
```

Set the line wrapping for the view.

```
procedure Set_Editable
  (Text_View          : access Gtk_Text_View_Record;
   Setting            :      Boolean := True);
function Get_Editable
  (Text_View          : access Gtk_Text_View_Record)
   return Boolean;
```

Set the default editability of the Gtk_Text_View.
You can override this default setting with tags in the buffer, using the "editable" attribute of tags.

```
procedure Set_Pixels_Above_Lines
  (Text_View          : access Gtk_Text_View_Record;
   Pixels_Above_Lines  :      Gint);
function Get_Pixels_Above_Lines
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default number of blank pixels above paragraphs in Text_View.
Tags in the buffer for Text_View may override the defaults.

```
procedure Set_Pixels_Below_Lines
  (Text_View          : access Gtk_Text_View_Record;
   Pixels_Below_Lines  :      Gint);
function Get_Pixels_Below_Lines
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default number of pixels of blank space
to put below paragraphs in Text_View. May be overridden by tags applied to Text_View's buffer.

```
procedure Set_Pixels_Inside_Wrap
  (Text_View          : access Gtk_Text_View_Record;
   Pixels_Inside_Wrap  :      Gint);

function Get_Pixels_Inside_Wrap
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default number of pixels of blank space to leave between display/wrapped lines within a paragraph. May be overridden by tags in Text_View's buffer.

```
procedure Set_Justification
  (Text_View          : access Gtk_Text_View_Record;
   Justification      :      Gtk.Enums.Gtk_Justification);

function Get_Justification
  (Text_View          : access Gtk_Text_View_Record)
   return Gtk.Enums.Gtk_Justification;
```

Sets the default justification of text in Text_View. Tags in the view's buffer may override the default.

```
procedure Set_Left_Margin
  (Text_View          : access Gtk_Text_View_Record;
   Left_Margin        :      Gint);

function Get_Left_Margin
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default left margin for text in Text_View. Tags in the buffer may override the default.

```
procedure Set_Right_Margin
  (Text_View          : access Gtk_Text_View_Record;
   Right_Margin       :      Gint);

function Get_Right_Margin
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default right margin for text in the text view. Tags in the buffer may override the default.

```
procedure Set_Indent
  (Text_View          : access Gtk_Text_View_Record;
   Indent             :      Gint);

function Get_Indent
  (Text_View          : access Gtk_Text_View_Record)
   return Gint;
```

Sets the default indentation for paragraphs in Text_View. Tags in the buffer may override the default.

```
procedure Set_Tabs
  (Text_View          : access Gtk_Text_View_Record;
   Tabs               :      Pango.Tabs.Pango_Tab_Array);

function Get_Tabs
  (Text_View          : access Gtk_Text_View_Record)
   return Pango.Tabs.Pango_Tab_Array;
```

Sets the default tab stops for paragraphs in Text_View. Tags in the buffer may override the default The returned array will be Null_Tab_Array if "standard" (8-space) tabs are used. Free the return value Pango.Tabs.Free

```
procedure Set_Overwrite
  (Text_View          : access Gtk_Text_View_Record;
   Overwrite          :      Boolean);
function Get_Overwrite
  (Text_View          : access Gtk_Text_View_Record)
   return Boolean;
```

Changes the Text_View overwrite mode.

```
procedure Set_Accepts_Tab
  (Text_View          : access Gtk_Text_View_Record;
   Accepts_Tab        :      Boolean);
function Get_Accepts_Tab
  (Text_View          : access Gtk_Text_View_Record)
   return Boolean;
```

Sets the behavior of the text widget when the Tab key is pressed. If Accepts_Tab is True a tab character is inserted, otherwise the keyboard focus is moved to the next widget in the focus chain.

187.4 Example

```
--  The following example creates an empty text view, and puts it in a
--  scrolling area so that if more text is added, scrollbars are created
--  automatically.

declare
   View     : Gtk_Text_View;
   Buffer   : Gtk_Text_Buffer;
   Scrolled : Gtk_Scrolled_Window;
begin
   Gtk_New (Scrolled);
   Set_Policy (Scrolled, Policy_Automatic, Policy_Automatic);
   Gtk_New (Buffer);
   Gtk_New (View, Buffer);
   Add (Scrolled, View);
end;
```

188 Package Gtk.Toggle_Action

A Gtk_Toggle_Action corresponds roughly to a Gtk_Check_Menu_Item. It has an "active" state specifying whether the action has been checked or not.

188.1 Signals

- **"toggled"**

  ```
  procedure Handler (Toggled : access Gtk_Toggle_Action_Record'Class);
  ```

  Called when the state of the action is toggled.

188.2 Subprograms

```
procedure Gtk_New
  (Action             : out   Gtk_Toggle_Action;
   Name               :       String;
   Label              :       String := "";
   Tooltip            :       String := "";
   Stock_Id           :       String := "");
function Get_Type               return GType;
```

Return the internal type associated with Gtk_Toggle_Action.

```
procedure Set_Active
  (Action             : access Gtk_Toggle_Action_Record;
   Is_Active          :       Boolean);
function Get_Active
  (Action             : access Gtk_Toggle_Action_Record)
   return Boolean;
```

Returns the checked state of the toggle action.

```
procedure Set_Draw_As_Radio
  (Action             : access Gtk_Toggle_Action_Record;
   Draw_As_Radio      :       Boolean);
function Get_Draw_As_Radio
  (Action             : access Gtk_Toggle_Action_Record)
   return Boolean;
```

Returns whether the action should have proxies like a radio action. This changes the display of widgets associated with that action.

188.2.1 Signals

```
procedure Toggled
  (Action             : access Gtk_Toggle_Action_Record);
```

Emits the "toggled" signal on the toggle action.

189 Package Gtk.Toggle_Button

A Gtk_Toggle_Button is like a regular button, but can be in one of two states, "active" or "inactive". Its visual aspect is modified when the state is changed.

You should consider using a Gtk_Check_Button instead, since it looks nicer and provides more visual clues that the button can be toggled.

189.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Bin           (Package Gtk.Bin)
            \___ Gtk_Button     (Package Gtk.Button)
               \___ Gtk_Toggle_Button (Package Gtk.Toggle_Button)
```

189.2 Signals

- **"toggled"**

  ```
  procedure Handler (Toggle : access Gtk_Toggle_Button_Record'Class);
  ```

  This signal is emitted every time the state of the button is modified.

189.3 Subprograms

```
procedure Gtk_New
  (Toggle_Button      : out   Gtk_Toggle_Button;
   Label              :       UTF8_String := "");
```

Initialize a button.
If Label is "", then no label is created inside the button and you will have to provide your own child through a call to Gtk.Container.Add. This is the recommended way to put a pixmap inside a toggle button.

```
procedure Gtk_New_With_Mnemonic
  (Toggle_Button      : out   Gtk_Toggle_Button;
   Label              :       UTF8_String);
```

Create a Gtk_Toggle_Button containing Label. The Label will be created
using Gtk.Label.Gtk_New_With_Mnemonic, so underscores in Label indicate the mnemonic for the button.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Toggle_Button.

```
procedure Set_Mode
  (Toggle_Button      : access Gtk_Toggle_Button_Record;
   Draw_Indicator     :       Boolean);
function Get_Mode
  (Toggle_Button      : access Gtk_Toggle_Button_Record)
   return Boolean;
```

Change the mode of the button.
If Draw_Indicator is False, then the button is hidden.

```
procedure Set_Active
  (Toggle_Button      : access Gtk_Toggle_Button_Record;
   Is_Active          :       Boolean);
```

```
function Get_Active
  (Toggle_Button      : access Gtk_Toggle_Button_Record)
   return Boolean;
```

Change the state of the button.
When Is_Active is True, the button is drawn as a pressed button.

```
procedure Set_Inconsistent
  (Toggle_Button      : access Gtk_Toggle_Button_Record;
   Setting            :      Boolean := True);
function Get_Inconsistent
  (Toggle_Button      : access Gtk_Toggle_Button_Record)
   return Boolean;
```

If the user has selected a range of elements (such as some text or spreadsheet cells) that are affected by a toggle button, and the current values in that range are inconsistent, you may want to display the toggle in an "in between" state. This function turns on "in between" display. Normally you would turn off the inconsistent state again if the user toggles the toggle button. This has to be done manually, Set_Inconsistent only affects visual appearance, it doesn't affect the semantics of the button.

189.3.1 Signals emission

```
procedure Toggled
  (Toggle_Button      : access Gtk_Toggle_Button_Record);
```

Emit the toggled signal on this widget.
Note that the state of the button is not changed, only the callbacks are called.

189.4 Example

```
--  This example creates a toggle button with a pixmap in it

with Gtk.Toggle_Button, Gdk.Pixmap, Gdk.Bitmap, Gtk.Pixmap;
with Gtk.Style, Gtk.Enums;

procedure Toggle is
   Toggle    : Gtk.Toggle_Button.Gtk_Toggle_Button;
   Style     : Gtk.Style.Gtk_Style;
   Pixmap    : Gdk.Pixmap.Gdk_Pixmap;
   Mask      : Gdk.Bitmap.Gdk_Bitmap;
   PixmapWid : Gtk.Pixmap.Gtk_Pixmap;
begin
   --  Do not specify a label
   Gtk.Toggle_Button.Gtk_New (Toggle);

   Style := Gtk.Toggle_Button.Get_Style (Toggle);
   Gdk.Pixmap.Create_From_Xpm
     (Pixmap,
      Gtk.Toggle_Button.Get_Window (Toggle),
      Mask,
      Gtk.Style.Get_Bg (Style, Gtk.Enums.State_Normal),
```

```
         "icon.xpm");
      Gtk.Pixmap.Gtk_New (PixmapWid, Pixmap, Mask);

      --  Add the pixmap to the button
      Gtk.Toggle_Button.Add (Toggle, PixmapWid);
   end Toggle;
```

190 Package Gtk.Toggle_Tool_Button

This package defines the base class for all items that can be added into a toolbar (see gtk-toolbar.ads). See also Gtk.Tool_Button (gtk-tool_button.ads). See also Gtk.Separator_Tool_Item (gtk-separator_tool_item).

190.1 Signals

- **"toggled"**

```
procedure Handler
(Button     : access Gtk_Toggle_Tool_Button_Record'Class);
```

  Emitted whenever the toggle button changes state

190.2 Subprograms

```
procedure Gtk_New
  (Button             : out   Gtk_Toggle_Tool_Button);
procedure Gtk_New_From_Stock
  (Button             : out   Gtk_Toggle_Tool_Button;
   Stock_Id           :       String);
function Get_Type                return GType;
```

Internal type representing this class of widgets

```
procedure Set_Active
  (Button             : access Gtk_Toggle_Tool_Button_Record;
   Is_Active          :       Boolean);
function Get_Active
  (Button             : access Gtk_Toggle_Tool_Button_Record)
   return Boolean;
```

Sets whether the button should be selected

191 Package Gtk.Tool_Button

This package defines a special kind of Gtk.Toolbar child that embeds a button. See also gtk-toggle_tool_button.ads, gtk-radio_tool_button.ads and gtk-menu_tool_button.ads

191.1 Signals

- **"clicked"**

```
procedure Handler
(Button : access Gtk_Tool_Button_Record'Class);
```

Emitted when the button is clicked with the mouse or activated with the keyboard.

191.2 Subprograms

191.2.1 Creating buttons

```
procedure Gtk_New
  (Button             : out   Gtk_Tool_Button;
   Icon_Widget        :       Gtk.Widget.Gtk_Widget := null;
   Label              :       String := "");
procedure Gtk_New_From_Stock
  (Button             : out   Gtk_Tool_Button;
   Stock_Id           :       String);
function Get_Type               return GType;
```

Return the internal type used for this widget class

```
procedure Set_Icon_Name
  (Button             : access Gtk_Tool_Button_Record;
   Icon_Name          :       String);
function Get_Icon_Name
  (Button             : access Gtk_Tool_Button_Record)
   return String;
```

Sets the icon for the tool button from a named themed icon.
See the docs for Gtk.Icon_them for more details. The "icon_name" property only has an effect if not overriden by non-null "label", "icon_widget" or "stock_id" properties

```
procedure Set_Icon_Widget
  (Button             : access Gtk_Tool_Button_Record;
   Icon_Widget        :       Gtk.Widget.Gtk_Widget := null);
function Get_Icon_Widget
  (Button             : access Gtk_Tool_Button_Record)
   return Gtk.Widget.Gtk_Widget;
```

Sets or gets the widget used as icon on Button.
If Icon_Widget is null, the icon used for the button is determined by the "stock_id" property. If the latter is also null, the button has no icon

```
procedure Set_Label
  (Button             : access Gtk_Tool_Button_Record;
   Label              :       String);
function Get_Label
  (Button             : access Gtk_Tool_Button_Record)
   return String;
```

Sets or gets the label used for the button. The "label" property only
has an effect if not overridden by a non-null "label_widget" property. If both are null, the

label comes from the "stock_id" properties. If also null, the button has no label. Get_Label only returns the value of the "labeL" property.

```
procedure Set_Label_Widget
  (Button             : access Gtk_Tool_Button_Record;
   Label_Widget       :      Gtk.Widget.Gtk_Widget := null);
function Get_Label_Widget
  (Button             : access Gtk_Tool_Button_Record)
   return Gtk.Widget.Gtk_Widget;
```

Sets Label_Widget as the widget that will be used as the label for the button. If this is null, the "label" property is used as label.

```
procedure Set_Stock_Id
  (Button             : access Gtk_Tool_Button_Record;
   Stock_Id           :      String);
function Get_Stock_Id
  (Button             : access Gtk_Tool_Button_Record)
   return String;
```

Sets the name of the stock item. This property has no effect if overriden by non-null "label" or "icon_widget" properties.

```
procedure Set_Use_Underline
  (Button             : access Gtk_Tool_Button_Record;
   Use_Underline      :      Boolean := True);
function Get_Use_Underline
  (Button             : access Gtk_Tool_Button_Record)
   return Boolean;
```

If Use_Underline is true, an underline in the label property indicates that the next character should be used a mnemonic accelerator key in the overflow menu of the toolbar. For instance, if the label is "_Open", the item in the overflow menu can be activated with alt-O. Labels shown on tool buttons never have mnemonics on them.

192 Package Gtk.Tool_Item

This package defines the base class for all items that can be added into a toolbar (see gtk-toolbar.ads). See also Gtk.Tool_Button (gtk-tool_button.ads). See also Gtk.Separator_Tool_Item (gtk-separator_tool_item).

192.1 Signals

- **"create_menu_proxy"**

  ```
  function Handler
  (Item : access Gtk_Tool_Item_Record'Class) return Boolean;
  ```

  Emitted when the toolbar needs information from the item about whether the item should appear in the toolbar overflow menu. In response, the item should either: - call Set_Proxy_Menu_Item with a null parameter, and return True, to indicate that the item should not appear - call Set_Proxy_Menu_Item with a new menu item, and return True - return False to indicate that the signal wasn't handled. The item will not appear in the overflow menu unless a later handler installs a menu item The toolbar may cache the result of this signal. See Rebuild_Menu to invalidate the cache.

- **"set_tooltip"**

  ```
  function Handler
  (Item        : access Gtk_Tool_Item_Record'Class;
  Tooltips    : access Gtk_Tooltips_Record'Class;
  Tip         : String;
  Tip_Private : String) return Boolean;
  ```

  Emitted when the item's tooltip has changed through Set_Tooltip. Should return True if the signal was handled.

- **"toolbar_reconfigured"**

  ```
  procedure Handler (Item : access Gtk_Tool_Item_Record'Class);
  ```

  Emitted when some property of the toolbar that Item belongs to has changed.

192.2 Subprograms

192.2.1 Creating items

```
procedure Gtk_New
  (Item               : out   Gtk_Tool_Item);

function Get_Type               return GType;
```

Return the internal value associated with a Gtk_Button.

```
procedure Set_Expand
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Expand             :      Boolean);

function Get_Expand
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Boolean;
```

Sets whether Tool_Item is allocated extra space when there is more room
on the toolbar than needed for the items. The effect is that the item gets bigger when the toolbar gets bigger.

```
procedure Set_Homogeneous
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Homogeneous        :      Boolean);

function Get_Homogeneous
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Boolean;
```

Sets whether Tool_Item is to be allocated the same size as other homogeneous items. The effect is that all homogeneous items will have the same width as the widest of the items.

```
function Get_Icon_Size
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Gtk.Enums.Gtk_Icon_Size;
```

Returns the icon size used for Tool_Item. Custom subclasses of Gtk_Tool_Item_Record should call this function to find out what size icons they should use. This settings depends on the toolbar that contains the item

```
procedure Set_Is_Important
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Is_Important       :      Boolean);

function Get_Is_Important
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Boolean;
```

Sets whether Tool_Item should be considered important. The Gtk_Tool_Button class uses this property to determine whether to show or hide its label when the toolbar style is Toolbar_Both_Horiz. The result is that only tool buttons with the "is_important" property set have labels, an effect known as "priority text".

```
function Get_Orientation
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Gtk.Enums.Gtk_Orientation;
```

Returns the orientation used for Tool_Item.

```
procedure Set_Proxy_Menu_Item
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Menu_Item_Id       :      String;
   Menu_Item          :      Gtk.Menu_Item.Gtk_Menu_Item);

function Get_Proxy_Menu_Item
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Menu_Item_Id       :      String)
   return Gtk.Menu_Item.Gtk_Menu_Item;
```

Sets the menu item used in the toolbar overflow menu. Menu_Item_Id is used to identify the caller of this function and should also be used with Get_Proxy_Menu_Item. Custom subclasses of Gtk_Tool_Item_Record should use this function to update their menu item when the tool item changes. See also Gtk.Toolbar.Set_Show_Arrow.

```
function Retrieve_Proxy_Menu_Item
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Gtk.Menu_Item.Gtk_Menu_Item;
```

Returns the menu item that was last set by Set_Proxy_Menu_Item, ie the menu item that will appear in the overflow menu. This might be different from the one set through Set_Proxy_Menu_Item, if someone else has overriden the menu afterward.

```
procedure Rebuild_Menu
  (Tool_Item          : access Gtk_Tool_Item_Record);
```

Calling this function signals to the toolbar that the
overflow menu item for Tool_Item has changed. If the overflow menu is visible when this function it called, the menu will be rebuilt.

```
procedure Set_Tooltip
  (Tool_Item          : access Gtk_Tool_Item_Record;
   Tooltips           : access Gtk.Tooltips.Gtk_Tooltips_Record'Class;
   Tip_Text           :      String;
   Tip_Private        :      String := "");
```

Sets the tooltips object to be used for Tool item, the text to be
displayed as tooltip on the item and the private text to be used

```
procedure Set_Visible_Vertical
  (Toolitem           : access Gtk_Tool_Item_Record;
   Visible_Vertical   :      Boolean);
function Get_Visible_Vertical
  (Toolitem           : access Gtk_Tool_Item_Record)
   return Boolean;
```

Sets whether Toolitem is visible when the toolbar is docked
vertically. Some tool items, such as text entries, are too wide to be useful on a vertically docked toolbar. If visible_vertical is False Toolitem will not appear on toolbars that are docked vertically.

```
procedure Set_Visible_Horizontal
  (Toolitem           : access Gtk_Tool_Item_Record;
   Visible_Horizontal  :      Boolean);
function Get_Visible_Horizontal
  (Toolitem           : access Gtk_Tool_Item_Record)
   return Boolean;
```

Same as Set_Visible_Vertical, but for a horizontal orientation

```
procedure Set_Use_Drag_Window
  (Toolitem           : access Gtk_Tool_Item_Record;
   Use_Drag_Window    :      Boolean);
function Get_Use_Drag_Window
  (Toolitem           : access Gtk_Tool_Item_Record)
   return Boolean;
```

Sets whether Toolitem has a drag window. When True the
toolitem can be used as a drag source through gtk_drag_source_set(). When Toolitem has a drag window it will intercept all events, even those that would otherwise be sent to a child of Toolitem.

```
function Get_Relief_Style
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Gtk.Enums.Gtk_Relief_Style;
```

Get the relief style of the item

```
function Get_Toolbar_Style
  (Tool_Item          : access Gtk_Tool_Item_Record)
   return Gtk.Enums.Gtk_Toolbar_Style;
```

Get the style of the toolbar that contains the item

193 Package Gtk.Toolbar

A toolbar groups a number of items (buttons, combo boxes,...), generally at the top of the application window, just below the menu bar. It provides quick access to the most commonly used features of your application. It is common for an application to have multiple toolbars.

193.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Toolbar     (Package Gtk.Toolbar)
```

193.2 Signals

- **"focus_home_or_end"**

  ```
  function Handler
  (Toolbar    : access Gtk_Toolbar_Record'Class;
  Focus_Home : Boolean) return Boolean;
  ```

  A keybinding signal used internally by GTK+. This signal can't be used in application code

- **"move_focus"**

- **"orientation-changed"**

  ```
  procedure Handler
  (Toolbar     : access Gtk_Toolbar_Record'Class;
  Orientation : Gtk_Orientation);
  ```

  Emitted when the orientation of the toolbar changes

- **"popup_context_menu"**

  ```
  function Handler
  (Toolbar      : access Gtk_Toolbar_Record'Class;
  X, Y, Button : Gint) return Boolean;
  ```

  Emitted when the user right-clicks the toolbar or uses the keybinding to display a popup menu. Application developers should handle this signal if they want to display a context menu on the toolbar. The context-menu should appear at the coordinates given by (x, y). The mouse button number is given by the Button parameter (set to -1 if popped up with the keyboard). Return value is True if the signal was handled.

- **"style-changed"**

  ```
  procedure Handler
  (Toolbar     : access Gtk_Toolbar_Record'Class;
  Style       : Gtk_Toolbar_Style);
  ```

  Emitted when the style of the toolbar changes

193.3 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Toolbar);
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Toolbar.

193.3.1 Items

```
procedure Insert
  (Toolbar            : access Gtk_Toolbar_Record;
   Item               : access Gtk.Tool_Item.Gtk_Tool_Item_Record'Class;
   Pos                :      Gint := -1);
```

Insert a new item anywhere in the toolbar.
If Pos is negative, the item is inserted at the end. If Pos is 0, the item is inserted first in the toolbar

```
function Get_Item_Index
  (Toolbar            : access Gtk_Toolbar_Record;
   Item               : access Gtk.Tool_Item.Gtk_Tool_Item_Record'Class)
   return Gint;
```

Get the position of Item within the toolbar

```
function Get_N_Items
  (Toolbar            : access Gtk_Toolbar_Record)
   return Gint;
```

Return the number of items in the toolbar

```
function Get_Nth_Item
  (Toolbar            : access Gtk_Toolbar_Record;
   N                  :      Gint)
   return Gtk.Tool_Item.Gtk_Tool_Item;
```

Return the n-th item in the toolbar

```
procedure Set_Drop_Highlight_Item
  (Toolbar            : access Gtk_Toolbar_Record;
   Tool_Item          : access Gtk.Tool_Item.Gtk_Tool_Item_Record'Class;
   Index              :      Gint);
```

Highlights Toolbar to give an idea of what it would look like
if Item was added at the position indicated by Index. If Item is %NULL, highlighting is turned off. In that case Index is ignored.

The item passed to this function must not be part of any widget hierarchy. When an item is set as drop highlight item it can not be added to any widget hierarchy or used as highlight item for another toolbar.

193.3.2 Style functions

```
procedure Set_Orientation
  (Toolbar            : access Gtk_Toolbar_Record;
   Orientation        :      Gtk_Orientation);
function Get_Orientation
  (Toolbar            : access Gtk_Toolbar_Record)
   return Gtk_Orientation;
```

Set or get the orientation (horizontal, vertical) for the toolbar

```
procedure Set_Style
  (Toolbar            : access Gtk_Toolbar_Record;
   Style              :      Gtk_Toolbar_Style);
function Get_Style
  (Toolbar            : access Gtk_Toolbar_Record)
   return Gtk_Toolbar_Style;
```

Set the style of the toolbar: text only, images only, or both

```
procedure Unset_Style
  (Toolbar            : access Gtk_Toolbar_Record);
```

Unsets a toolbar style set with Set_Style, so that user preferences will be used to determine the toolbar style. These user preferences are defined through the current gtk+ theme

```
procedure Set_Tooltips
  (Toolbar            : access Gtk_Toolbar_Record;
   Enable             :      Boolean);
function Get_Tooltips
  (Toolbar            : access Gtk_Toolbar_Record)
   return Boolean;
```

Sets whether tooltips should be enabled for items in the toolbar

```
function Get_Relief_Style
  (Toolbar            : access Gtk_Toolbar_Record)
   return Gtk_Relief_Style;
```

Returns the relief style of buttons on Toolbar. See Gtk.Button.Set_Relief for more information on reliefs.

```
procedure Set_Show_Arrow
  (Toolbar            : access Gtk_Toolbar_Record;
   Show_Arrow         :      Boolean := True);
function Get_Show_Arrow
  (Toolbar            : access Gtk_Toolbar_Record)
   return Boolean;
```

Sets or Gets whether to show an overflow arrow when the toolbar doesn't have room for all items on it. If True, the items that have no room are still available to the user.

```
function Get_Icon_Size
  (Toolbar            : access Gtk_Toolbar_Record)
   return Gtk_Icon_Size;
```

Returns the icon size used in this toolbar

193.3.3 Misc

```
function Get_Drop_Index
  (Toolbar            : access Gtk_Toolbar_Record;
   X                  :      Gint;
   Y                  :      Gint)
   return Gint;
```

Returns the position corresponding to the indicated point on Toolbar. This is useful when dragging items to the toolbar: this function returns the position a new item should be inserted. (X, Y) are the coordinates, in pixels, within the toolbar

194 Package Gtk.Tooltips

Tooltips are the small text windows that popup when the mouse rests over a widget, and that provide a quick help for the user.

In GtkAda, all tooltips belong to a group (a Gtk_Tooltips). All the individual tooltips in a group can be disabled or enabled at the same time. Likewise, the colors and style of a tooltip can be set on a group basis.

See the example at the end for how to change the default colors used for tooltips.

194.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Tooltips          (Package Gtk.Tooltips)
```

194.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Tooltips);
```

Create a new group of tooltips.

```
function Get_Type             return Glib.GType;
```

Return the internal value associated with a Gtk_Tooltips.

```
procedure Enable
  (Tooltips           : access Gtk_Tooltips_Record);
```

Enable all the tooltips in the group.
From now on, when the mouse rests over a widget for a short period of time, the help text is automatically displayed.

```
procedure Disable
  (Tooltips           : access Gtk_Tooltips_Record);
```

Disable all the tooptips in the group.
From now on, no tooltips in this group will appear, unless they are re-enabled.

```
procedure Set_Tip
  (Tooltips           : access Gtk_Tooltips_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
   Tip_Text           :      UTF8_String;
   Tip_Private        :      UTF8_String := "");
```

Add a new tooltip to Widget.
The message that appears in the tooltip is Tip_Text, and the tooltip belongs to the group Tooltips. Tip_Private contains more information, that can be displayed by a Gtk_Tips_Query widget through the "widget_selected" signal. In most cases, Tip_Private should simply keep its default empty value.

```
function Get_Data
  (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
   return Tooltips_Data;
```

Return the tooltip data associated with the Widget.
If there is none, the two text fields in the returned structure have a length 0.

```
procedure Force_Window
  (Widget             : access Gtk_Tooltips_Record);
```

Make sure the window in which the tooltips will be displayed is created. This is useful if you want to modify some characteristics of that window.

```
procedure Set_Markup
  (Tooltips           : access Gtk_Tooltips_Record;
   Text               :      UTF8_String);
```

Sets the text of the tooltip to be markup, which is marked up with the Pango text markup language. If markup is empty string, the label will be hidden.

```
procedure Set_Icon_From_Stock
  (Tooltips           : access Gtk_Tooltips_Record;
   Stock_Id           :      String;
   Size               :      Gtk.Enums.Gtk_Icon_Size);
```

Sets the icon of the tooltip (which is in front of the text) to be the stock item indicated by stock_id with the size indicated by size. If stock_id is emtry string, the image will be hidden.

194.3 Example

```
--  This example demonstrates how you can change the color scheme used
--  for tooltips.
--  This is of course done through styles. However, you can not directly
--  associate a Gtk_Tooltips with a style, so you have to do the follow-
ing.

--  Note also that this choice should probably left to the user, who can
--  modify it through a RC file that contains the following:
--      style "postie"
--      {
--         bg[NORMAL]={1.0, 0.93, 0.22}
--      }
--      widget "gtk-tooltips*" style "postie"

with Gtk.Tooltips, Gtk.Style, Gtk.Enums, Gtk.Widget, Gdk.Color;
use Gtk.Tooltips, Gtk.Style, Gtk.Enums, Gtk.Widget, Gdk.Color;

procedure Tooltips is
   Style : Gtk_Style;
   Tips  : Gtk_Tooltips;
   Color : Gdk_Color;
begin
   Gtk_New (Style);

   --  blue foreground
   Set_Rgb (Color, 255, 255, 65535);
   Alloc (Get_Default_Colormap, Color);
   Set_Foreground (Style, State_Normal, Color);

   --  green background
```

```
      Set_Rgb (Color, 255, 65535, 255);
      Alloc (Get_Default_Colormap, Color);
      Set_Background (Style, State_Normal, Color);

      Gtk_New (Tips);
      Force_Window (Tips);
   end Tooltips;
```

195 Package Gtk.Tree_Dnd

GTK+ supports Drag-and-Drop in tree views with a high-level and a low-level API.

The low-level API consists of the GTK+ DND API, augmented by some treeview utility functions: Gtk.Tree_View.Set_Drag_Dest_Row, Gtk.Tree_View.Get_Drag_Dest_Row, Gtk.Tree_View.Get_Dest_Row_At_Pos, Gtk.Tree_View.Create_Row_Drag_Icon, Set_Row_Drag_Data and Get_Row_Drag_Data. This API leaves a lot of flexibility, but nothing is done automatically, and implementing advanced features like hover-to-open-rows or autoscrolling on top of this API is a lot of work.

On the other hand, if you write to the high-level API, then all the bookkeeping of rows is done for you, as well as things like hover-to-open and auto-scroll, but your models have to implement the Gtk_Tree_Drag_Source and Gtk_Tree_Drag_Dest interfaces.

195.1 Types

```
type Gtk_Tree_Drag_Dest is new Glib.Types.GType_Interface;
```

```
type Gtk_Tree_Drag_Source is new Glib.Types.GType_Interface;
```

195.2 Subprograms

```
function Drag_Dest_Get_Type    return GType;
function Drag_Source_Get_Type  return GType;
```

Return the low-level types associated with the interfaces

```
function Drag_Dest_Drag_Data_Received
  (Drag_Dest          :      Gtk_Tree_Drag_Dest;
   Dest               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Selection_Data     :      Gtk.Selection.Selection_Data)
   return Boolean;
```

Asks the Drag_Dest to insert a row before the path Dest,
deriving the contents of the row from Selection_Data. If Dest is outside the tree so that inserting before it is impossible, False will be returned. Also, False may be returned if the new row is not created for some model-specific reason. Should robustly handle a Dest no longer found in the model!

```
function Drag_Dest_Row_Drop_Possible
  (Drag_Dest          :      Gtk_Tree_Drag_Dest;
   Dest_Path          :      Gtk.Tree_Model.Gtk_Tree_Path;
   Selection_Data     :      Gtk.Selection.Selection_Data)
   return Boolean;
```

Determines whether a drop is possible before the given Dest_Path,
at the same depth as Dest_Path. i.e., can we drop the data in Selection_Data at that location. Dest_Path does not have to exist; the return value will almost certainly be False if the parent of Dest_Path doesn't exist, though.

```
function Drag_Source_Drag_Data_Delete
  (Drag_Source        :      Gtk_Tree_Drag_Source;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Asks the Drag_Source to delete the row at Path, because
it was moved somewhere else via drag-and-drop. Returns False if the deletion fails because Path no longer exists, or for some model-specific reason. Should robustly handle a Path no longer found in the model!

```
function Drag_Source_Drag_Data_Get
  (Drag_Source        :      Gtk_Tree_Drag_Source;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Selection_Data     :      Gtk.Selection.Selection_Data)
   return Boolean;
```

Asks the Drag_Source to fill in Selection_Data with a
representation of the row at Path. Get_Target (Selection_Data) gives the required type of the data. Should robustly handle a Path no longer found in the model!

```
function Drag_Source_Row_Draggable
  (Drag_Source        :      Gtk_Tree_Drag_Source;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Asks the Drag_Source whether a particular row can be used as
the source of a DND operation. If the source doesn't implement this interface, the row is assumed draggable.

```
procedure Get_Row_Drag_Data
  (Selection_Data     :      Gtk.Selection.Selection_Data;
   Tree_Model         : out  Gtk.Tree_Model.Gtk_Tree_Model;
   Path               : out  Gtk.Tree_Model.Gtk_Tree_Path;
   Success            : out  Boolean);
```

Obtains a Tree_Model and Path from selection data of target type
GTK_TREE_MODEL_ROW. Normally called from a drag_data_received handler.

This function can only be used if Selection_Data originates from the same process that's calling this function, because a pointer to the tree model is being passed around. If you aren't in the same process, then you'll get memory corruption. In the Gtk_Tree_Drag_Dest drag_data_received handler, you can assume that selection data of type GTK_TREE_MODEL_ROW is in from the current process.

The returned path must be freed with

```
function Set_Row_Drag_Data
  (Selection_Data     :      Gtk.Selection.Selection_Data;
   Tree_Model         : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Sets selection data of target type GTK_TREE_MODEL_ROW. Normally used
in a drag_data_get handler.

196 Package Gtk.Tree_Model

The type Gtk_Tree_Model defined in this model defines an abstract interface to represent sets of data that will be displayed in a Gtk_Tree_View. Various default implementations are provided in the Gtk.Tree_Store and Gtk.List_Store packages.

Data are considered as being organized into a tree-like structure.

This package also defines a number of other types to manipulate these models:

A Gtk_Tree_Path is a textual pointer to a specific row/node in the model. It is a column separate list of numbers, that indicate the index of the child they point to. For instance, "10:4:0" would points to the first (0) child of the fifth (4) child of the eleventh child of the root. The depth of this path is 3.

A Gtk_Tree_Iter is similar to a path, but is a direct pointer to the actual data. It is also more efficient to use than paths.

A Gtk_Row_Reference is an object that tracks model changes, so that it always refere to the same row. A Gtk_Tree_Path refers to a position in the model, not a fixed row.

196.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Tree_Model        (Package Gtk.Tree_Model)
```

196.2 Signals

- **"row_changed"**

  ```
  procedure Handler
  (Tree_Model : access Gtk_Tree_Model_Record'Class;
  Path        : Gtk_Tree_Path;
  Iter        : Gtk_Tree_Iter);
  ```

  This signal should be emitted every time the contents of a row (any column) has changed. This forces the tree_view to refresh the display.

- **"row_deleted"**

  ```
  procedure Handler
  (Tree_Model : access Gtk_Tree_Model_Record'Class;
  Path        : Gtk_Tree_Path);
  ```

  This should be emitted by models after the child state of a node changes.

- **"row_has_child_toggled"**

  ```
  procedure Handler
  (Tree_Model : access Gtk_Tree_Model_Record'Class;
  Path        : Gtk_Tree_Path;
  Iter        : Gtk_Tree_Iter);
  ```

  This should be emitted by models after the child state of a node changes.

- **"row_inserted"**

  ```
  procedure Handler
  (Tree_Model : access Gtk_Tree_Model_Record'Class;
  Path        : Gtk_Tree_Path;
  Iter        : Gtk_Tree_Iter);
  ```

  This signal should be emitted every time a new row has been inserted.

- **"rows_reordered"**

```
procedure Handler
(Tree_Model : access Gtk_Tree_Model_Record'Class;
Path        : Gtk_Tree_Path;
Iter        : Gtk_Tree_Iter;
New_Order   : Gint_Array);
```

  This should be emitted when the rows have been reordered

196.3 Types

```
type Gtk_Tree_Iter is private;
```

```
type Gtk_Tree_Model_Foreach_Func is access function
        (Model      : access Gtk_Tree_Model_Record'Class;
```

```
type Gtk_Tree_Path is new Glib.C_Proxy;
```

```
type Gtk_Tree_Row_Reference is new Glib.C_Proxy;
```

```
type Tree_Model_Flags is mod 2 ** 32;
```

196.4 Subprograms

196.4.1 Tree models

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Tree_Model.

```
function Get_Flags
  (Model              : access Gtk_Tree_Model_Record)
   return Tree_Model_Flags;
```

Return a set of flags supported by this interface. The flags supported should not change during the lifecycle of the tree_model. The flags should not change in the lifetime of the model.

```
function Get_N_Columns
  (Tree_Model         : access Gtk_Tree_Model_Record)
   return Gint;
```

Return the number of columns supported by Tree_Model.

```
function Get_Column_Type
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Index              :      Gint)
   return GType;
```

Return the type of the Index-th column in the model.

```
procedure Foreach
  (Model              : access Gtk_Tree_Model_Record;
   Func               :      Gtk_Tree_Model_Foreach_Func;
   User_Data          :      System.Address);
```

Calls func on each node in model in a depth-first fashion. If func returns True, then the tree ceases to be walked, and Foreach returns.

196.4.2 Paths manipulation

```
function Gtk_New
  (Path               :      String := "")
   return Gtk_Tree_Path;
```

Create a new Gtk_Tree_Path from a path string.
Path should have the format described above, like "10:4:0". If it is the empty string, then a Gtk_Tree_Path of depth 0 is returned. The memory allocated for the path must be freed explicitely by calling Path_Free below.

```
function Gtk_New_First          return Gtk_Tree_Path;
```

Return a new path pointed to the first row in the model. The string representation is "0"

```
function Path_Get_Type          return Glib.GType;
```

Return the internal type used for Gtk_Tree_Path

```
function To_String
  (Path               :      Gtk_Tree_Path)
   return String;
```

Generate a string representation of the path.
This string is a colon-separated list of numbers, as described above.

```
procedure Append_Index
  (Path               :      Gtk_Tree_Path;
   Index              :      Gint);
```

Append a new index to a path.
As a result, the depth of the path is increased. See Path_Up for the opposite operation.

```
procedure Prepend_Index
  (Path               :      Gtk_Tree_Path;
   Index              :      Gint);
```

Prepend a new index to a path. As a result, the depth of the path is increased.

```
function Get_Depth
  (Path               :      Gtk_Tree_Path)
   return Gint;
```

Return the current depth of Path.

```
function Get_Indices
  (Path               :      Gtk_Tree_Path)
   return Glib.Gint_Array;
```

Return the list of indices from the path. This is an array of integers, each representing a node in a tree, as described in the path format.

```
procedure Path_Free
  (Path               :      Gtk_Tree_Path);
```

Free the memory allocated for Path.

```
function Copy
  (Path               :      Gtk_Tree_Path)
   return Gtk_Tree_Path;
```

Create a new Gtk_Tree_Path as a copy of Path. The memory allocated for the new path must be freed by a call to Path_Free.

```
function Compare
  (A, B               :      Gtk_Tree_Path)
   return Gint;
```

Compare two paths. If A appears before B in a tree, then -1 is returned. If B appears before A, then 1 is returned. If the two nodes are equal, then 0 is returned.

```
procedure Next
  (Path               :      Gtk_Tree_Path);
```

Move the Path to point to the next node at the current depth. In effect, it increments the last indice of the path. Note that the path might become invalid if there is no more node at this depth.

```
function Prev
  (Path               :      Gtk_Tree_Path)
   return Boolean;
```

Move Path to point to the previous node at the current depth, if it exists. Return True if Path has a previous node, and the move was made. If it returns False, then Path has not been changed.

```
function Up
  (Path               :      Gtk_Tree_Path)
   return Boolean;
```

Moves the Path to point to it's parent node, if it has a parent. Return True if Path has a parent, and the move was made. In practive, the depth of Path is decreased by 1.

```
procedure Down
  (Path               :      Gtk_Tree_Path);
```

Moves Path to point to the first child of the current path.

```
function Is_Ancestor
  (Path, Descendant   :      Gtk_Tree_Path)
   return Boolean;
```

Return True if Descendant is contained inside Path.

```
function Is_Descendant
  (Path, Ancestor     :      Gtk_Tree_Path)
   return Boolean;
```

Return True if Path is contained inside Ancestor.

```
procedure Convert;
```

```
procedure Convert;
```

196.4.3 Row_Reference manipulation

```
function Gtk_New
  (Model              : access Gtk_Tree_Model_Record;
   Path               :      Gtk_Tree_Path)
   return Gtk_Tree_Row_Reference;
```

Create a row reference based on Path. This reference will keep pointing
to the node pointed to by Path, so long as it exists. It listens to all signals on model, and updates it's path appropriately. If Path isn't a valid path in Model, then null is returned.

```
function Row_Reference_Get_Typereturn Glib.GType;
```

Return the internal type used for row reference.

```
function Get_Path
  (Reference          :      Gtk_Tree_Row_Reference)
   return Gtk_Tree_Path;
```

Return the path that Reference currently points to.
null is returned if Reference is no longer valid. The caller must free the returned path.

```
function Valid
  (Reference          :      Gtk_Tree_Row_Reference)
   return Boolean;
```

Return True if Reference is non null and is still valid.

```
function Row_Reference_Copy
  (Ref                :      Gtk_Tree_Row_Reference)
   return Gtk_Tree_Row_Reference;
```

Return a newly allocated copy of Ref

```
procedure Row_Reference_Free
  (Reference          :      Gtk_Tree_Row_Reference);
```

Free the memory occupied by Reference.

```
function Get_Model
  (Reference          :      Gtk_Tree_Row_Reference)
   return Gtk_Tree_Model;
```

Returns the model which Reference is monitoring in order to
appropriately the path.

196.4.4 Iterators

??? Need to be able to access the user_data fields, so that new models@* can define their own iterators

```
function Iter_Get_Type          return Glib.GType;
```

Return the internal type used for iterators

```
procedure Iter_Copy
  (Source             :      Gtk_Tree_Iter;
   Dest               : out  Gtk_Tree_Iter);
```

Create a copy of Source.
You can also copy tree iters simply by using the ":=" Ada construct.

```
procedure Set_Tree_Iter
  (Val                : in out Glib.Values.GValue;
   Iter               :      Gtk_Tree_Iter);
```

Set the value of the given GValue to Iter.
Note that Iter is stored by reference, which means no copy of Iter is made. Iter should remain allocated as long as Val is being used.

```
procedure Get_Tree_Iter
  (Val                :      Glib.Values.GValue;
   Iter               : out  Gtk_Tree_Iter);
```

Extract the iterator from the given GValue.
Note that the iterator returned is a copy of the iterator referenced by the give GValue. Modifying the iterator returned does not modify the iterator referenced by the GValue.

```
function Get_Iter
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Path               :      Gtk_Tree_Path)
   return Gtk_Tree_Iter;
```

Return an iterator pointing to Path.
Null_Iter is returned if Path was invalid or no iterator could be set.

```
function Get_Iter_From_String
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Path_String        :      String)
   return Gtk_Tree_Iter;
```

Return an iterator pointing to Path_String.
Null_Iter is returned if Path was invalid or no iterator could be set.

```
function Get_String_From_Iter
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter)
   return String;
```

Generates a string representation of the iter. This string is a ':'
separated list of numbers. For example, "4:10:0:3" would be an acceptable return value for this string.

```
function Get_Iter_First
  (Tree_Model         : access Gtk_Tree_Model_Record)
   return Gtk_Tree_Iter;
```

Return an iterator pointing to the root of Tree_Model.
Null_Iter is returned if Tree_Model is empty.

```
function Get_Path
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter)
   return Gtk_Tree_Path;
```

Return a newly created Gtk_Tree_Path referenced by Iter.
This path must be freed with Path_Free.

```
procedure Next
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               : in out Gtk_Tree_Iter);
```

Sets Iter to point to the node following it at the current level.
If there is none, Iter is set to Null_Iter.

```
function Children
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Parent             :      Gtk_Tree_Iter)
   return Gtk_Tree_Iter;
```

Return the first child of Parent.
If Parent has no children, return Null_Iter. Parent will remain a valid node after this function has been called.

```
function Has_Child
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter)
   return Boolean;
```

Return True if Iter has children, False otherwise.

```
function N_Children
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter := Null_Iter)
   return Gint;
```

Return the number of children that Iter has.
As a special case, if Iter is Null_Iter, then the number of toplevel nodes is returned.

```
function Nth_Child
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Parent             :      Gtk_Tree_Iter;
   N                  :      Gint)
   return Gtk_Tree_Iter;
```

Return the child of Parent, using the given index.
The First index is 0. If Index is too big, or Parent has no children, return Null_Iter. If Parent is Null_Iter, then the nth root node is set.

```
function Parent
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Child              :      Gtk_Tree_Iter)
   return Gtk_Tree_Iter;
```

Return the parent of Child.
If Child is at the toplevel, and doesn't have a parent, then Null_Iter is returned.

```
procedure Ref_Node
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter);
```

Let the tree reference the node.
This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons. This function is primarily meant as a way for views to let caching model know when nodes are being displayed (and hence, whether or not to cache that node). For example, a file-system based model would not want to keep the entire file-hierarchy in memory, just the sections that are currently being displayed by every current view.

```
procedure Unref_Node
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter);
```

Let the tree unref the node.
This is an optional method for models to implement. To be more specific, models may ignore this call as it exists primarily for performance reasons. For more information on what this means, please see Tree_Model_Ref_Node. Please note that nodes that are deleted are not unreferenced.

```
procedure Get_Value
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint;
   Value              : out   Glib.Values.GValue);
```

Get a value from the model, at column Column and line Iter.
Value must be freed by the caller.

```
function Get_Int
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return Gint;
```

Get the int value of one cell in the row referenced by Iter.

```
function Get_Boolean
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return Boolean;
```

Get the boolean value of one cell in the row referenced by Iter.

```
function Get_Object
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return Glib.Object.GObject;
```

Get the object value of one cell in the row referenced by Iter.

```
function Get_C_Proxy
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return Glib.C_Proxy;
```

Get the address value of one cell in the row referenced by Iter.

```
function Get_String
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return UTF8_String;
```

Get the string stored at a specific location in the model.

```
function Get_Address
  (Tree_Model         : access Gtk_Tree_Model_Record;
   Iter               :      Gtk_Tree_Iter;
   Column             :      Gint)
   return System.Address;
```

Get the pointer stored at a specific location in the model.

196.4.5 Signals

```
procedure Row_Changed
  (Tree_Model         : access Gtk_Tree_Model_Record'Class;
   Path               :      Gtk_Tree_Path;
   Iter               :      Gtk_Tree_Iter);
```

Emit the "row_changed" signal.

```
procedure Row_Inserted
  (Tree_Model         : access Gtk_Tree_Model_Record'Class;
   Path               :      Gtk_Tree_Path;
   Iter               :      Gtk_Tree_Iter);
```

Emit the "row_inserted" signal.

```
procedure Row_Has_Child_Toggled
  (Tree_Model         : access Gtk_Tree_Model_Record'Class;
   Path               :      Gtk_Tree_Path;
   Iter               :      Gtk_Tree_Iter);
```

Emit the "row_has_child_toggled" signal.

```
procedure Row_Deleted
  (Tree_Model         : access Gtk_Tree_Model_Record'Class;
   Path               :      Gtk_Tree_Path);
```

Emit the "row_has_child_toggled" signal.

```
procedure Rows_Reordered
  (Tree_Model         : access Gtk_Tree_Model_Record'Class;
   Path               :      Gtk_Tree_Path;
   Iter               :      Gtk_Tree_Iter;
   New_Order          :      Gint_Array);
```

Emit the "rows_reordered" signal

197 Package Gtk.Tree_Model_Filter

A Gtk_Tree_Model_Filter is a tree model which wraps another tree model, and can do the following things:

@itemize @bullet @item Filter specific rows, based on data from a "visible column", a column storing booleans indicating whether the row should be filtered or not, or based on the return value of a "visible function", which gets a model, iter and user_data and returns a boolean indicating whether the row should be filtered or not.

@item Modify the "appearance" of the model, using a modify function. This is extremely powerful and allows for just changing some values and also for creating a completely different model based on the given child model.

@item Set a different root node, also known as a "virtual root". You can pass in a Gtk_Tree_Path indicating the root node for the filter at construction time.

@end itemize

197.1 Types

type Data_Type **is private**;

type Destroy_Notify **is access procedure**
(Data : **in out** Data_Type);

Destroys the memory allocated for Data

type Gtk_Tree_Model_Filter_Visible_Func **is access function**
(Model : **access** Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;

197.2 Subprograms

```
procedure Gtk_New
  (Model              : out   Gtk_Tree_Model_Filter;
   Child_Model        : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;
   Root               :      Gtk.Tree_Model.Gtk_Tree_Path
                      := null);
function Get_Type               return Glib.GType;
```

Returns the internal type used for a Gtk_Tree_Model_Filter

197.2.1 Child model

The tree model filter wraps another model, and offers functions to@* convert from one to the other. Generally speaking, you can change data on either of the two models, and these changes will be reflected graphically automatically.

```
function Get_Model
  (Filter             : access Gtk_Tree_Model_Filter_Record)
```

```
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Returns a pointer to the child model of Filter.

```
procedure Convert_Child_Iter_To_Iter
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Filter_Iter        : out   Gtk.Tree_Model.Gtk_Tree_Iter;
   Child_Iter         :       Gtk.Tree_Model.Gtk_Tree_Iter);
```

Sets Filter_Iter to point to the row in Filter that corresponds to the row pointed at by Child_Iter.

```
function Convert_Child_Path_To_Path
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Child_Path         :       Gtk.Tree_Model.Gtk_Tree_Path)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Converts Child_Path to a path relative to Filter. That is, Child_Path points to a path in the child model. The returned path will point to the same row in the filtered model. If Child_Path isn't a valid path on the child model, then null is returned. The returned value must be freed with Path_Free.

```
procedure Convert_Iter_To_Child_Iter
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Child_Iter         : out   Gtk.Tree_Model.Gtk_Tree_Iter;
   Filter_Iter        :       Gtk.Tree_Model.Gtk_Tree_Iter);
```

Sets Child_Iter to point to the row pointed to by Filter_Iter.

```
function Convert_Path_To_Child_Path
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Filter_Path        :       Gtk.Tree_Model.Gtk_Tree_Path)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Converts Filter_Path to a path on the child model of Filter. That is, Filter_Path points to a location in Filter. The returned path will point to the same location in the model not being filtered. If Filter_Path does not point to a location in the child model, null is returned. The returned value must be freed with Path_Free.

197.2.2 Changing visibility

One of the capabilities of a Gtk_Tree_Model_Filter is to hide some of@* the rows of its child model, so that they are not visible on the screen.

```
procedure Set_Visible_Column
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Column             :       Gint);
```

Sets Column of the child_model to be the column where Filter should look for visibility information. Columns should be a column of type GType_Boolean, where True means that a row is visible, and False if not.

```
procedure Set_Visible_Func
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Func               :       Gtk_Tree_Model_Filter_Visible_Func);
```

Sets the visible function used when filtering the Filter to be Func. The function should return True if the given row should be visible and False otherwise. If the condition calculated by the function changes over time (e.g. because it depends on some global parameters), you must call Refilter to keep the visibility information of the model uptodate.

```
procedure Set_Visible_Func
  (Filter             : access Gtk_Tree_Model_Filter_Record'Class;
   Func               :      Gtk_Tree_Model_Filter_Visible_Func;
   Data               :      Data_Type;
   Destroy            :      Destroy_Notify := null);
```

Same as above, but the application can pass addition data to the function

```
procedure Refilter
  (Filter             : access Gtk_Tree_Model_Filter_Record);
```

Emits row_changed for each row in the child model, which causes the filter to re-evaluate whether a row is visible or not.

197.2.3 Modifying displayed values

The other capability of a Gtk_Tree_Model_Filter is to modify on the fly@* the displayed value (ie we do not display directly what is in the child model, but change the value in memory, not in the model, on the fly)

```
procedure Set_Modify_Func
  (Filter             : access Gtk_Tree_Model_Filter_Record;
   Types              :      Glib.GType_Array;
   Func               :      Gtk_Tree_Model_Filter_Modify_Func);
```

Types can be used to override the column types that will be made visible to the parent model/view. Func is used to specify the modify function. The modify function will get called for *each* data access, the goal of the modify function is to return the data which should be displayed at the location specified using the parameters of the modify function.

```
procedure Set_Modify_Func
  (Filter             : access Gtk_Tree_Model_Filter_Record'Class;
   Types              :      Glib.GType_Array;
   Func               :      Gtk_Tree_Model_Filter_Modify_Func;
   Data               :      Data_Type;
   Destroy            :      Destroy_Notify := null);
```

Same as above, but the application can pass extra data to the function.

197.2.4 Misc

```
procedure Clear_Cache
  (Filter             : access Gtk_Tree_Model_Filter_Record);
```

This function should almost never be called. It clears the Filter of any cached iterators that haven't been reffed with Gtk.Tree_Model.Ref_Node. This might be useful if the child model being filtered is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unrefed iters will be invalid.

197.2.5 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Tree_Drag_Source" This interface allows this widget to act as a dnd source @end itemize

```
function "+"
  (Model               : access Gtk_Tree_Model_Filter_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Source;
function "-"
  (Drag_Source        :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Source)
   return Gtk_Tree_Model_Filter;
```

Converts to and from the Gtk_Tree_Drag_Source interface

198 Package Gtk.Tree_Model_Sort

The Gtk_Tree_Model_Sort is a model which implements the Gtk_Tree_Sortable interface. It does not hold any data itself, but rather is created with child model and proxies its data. It has identical column types to this child model, and the changes in the child are propagated. The primary purpose of this model is to provide a way to sort a different model without modifying it. Note that the sort function used by Gtk_Tree_Model_Sort is not guaranteed to be stable.

The use of this is best demonstrated through an example. In the following sample code we create two Gtk_Tree_View widgets each with a view of the same data. As the model is wrapped here by a Gtk_Tree_Model_Sort, the two Gtk_Tree_Views can each sort their view of the data without affecting the other. By contrast, if we simply put the same model in each widget, then sorting the first would sort the second.

declare Tree_View1, Tree_View2 : Gtk_Tree_View; Sort_Model1, Sort_Model2 : Gtk_Tree_Model_Sort; Child_Model : Gtk_Tree_Model; begin Child_Model := Get_My_Model; – Your own implementation

– Create the first tree Gtk_New_With_Model (Sort_Model1, Child_Model); Gtk_New (Tree_View1, Sort_Model1); Set_Sort_Column_Id (Sort_Model1, COLUMN1, Sort_Ascending);

– Create the second tree Gtk_New_With_Model (Sort_Model2, Child_Model); Gtk_New (Tree_View2, Sort_Model2); Set_Sort_Column_Id (Sort_Model2, COLUMN1, Sort_Descending); end;

To demonstrate how to access the underlying child model from the sort model, the next example will be a callback for the Gtk_Tree_Selection "changed" signal. In this callback, we get a string from COLUMN_1 of the model. We then modify the string, find the same selected row on the child model, and change the row there.

procedure Selection_Changed (Selection : access Gtk_Tree_Selection_Record'Class) is Sort_Model, Child_Model : Gtk_Tree_Model; Sort_Iter, Child_Iter : Gtk_Tree_Iter; begin – Get the currently selected row and the model Get_Selected (Selection, Sort_Model, Sort_Iter); if Sort_Iter = Null_Iter then return; end if;

– Lookup the current value on the selected row declare Some_Data : constant String := Get_String (Sort_Model, Sort_Iter, COLUMN1); begin – Get an iterator on the child model instead of the sort model Convert_Iter_To_Child_Iter (Sort_Model, Child_Iter, Sort_Iter);

– Get the child model and change the value in the row – In this example, the model is a Gtk_List_Store, but it could be – anything Child_Model := Get_Model (Gtk_Sort_Model (Sort_Model)); Set (Ctk_List_Store (Child_Model), Child_Iter, COLUMN1, "data"); end; end Selection_Changed;

198.1 Subprograms

```
procedure Gtk_New_With_Model
  (Sort_Model         : out   Gtk_Tree_Model_Sort;
   Child_Model        : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class);

function Get_Type               return Glib.GType;
```

Return the internal type associated with a Gtk_Tree_Model_Sort.

```
function Get_Model
  (Tree_Model          : access Gtk_Tree_Model_Sort_Record)
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Return the model the Gtk_Tree_Model_Sort is sorting.

```
function Convert_Child_Path_To_Path
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record;
   Child_Path          :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Convert Child_Path to a path relative to Tree_Model_Sort.
That is, Child_Path points to a path in the child model. The returned path will point to the same row in the sorted model. If Child_Path isn't a valid path on the child model, then Null is returned. The returned value must be freed with Path_Free.

```
procedure Convert_Child_Iter_To_Iter
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record;
   Sort_Iter           : out   Gtk.Tree_Model.Gtk_Tree_Iter;
   Child_Iter          :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Set Sort_Iter to point to the row in Tree_Model_Sort that
corresponds to the row pointed at by Child_Iter.

```
function Convert_Path_To_Child_Path
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record;
   Sorted_Path         :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Gtk.Tree_Model.Gtk_Tree_Path;
```

Convert Sort_Path to a path on the child model of Tree_Model_Sort.
That is, Sort_Path points ot a location in Tree_Model_Sort. The returned path will point to the same location in the model not being sorted.

```
procedure Convert_Iter_To_Child_Iter
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record;
   Child_Iter          : out   Gtk.Tree_Model.Gtk_Tree_Iter;
   Sorted_Iter         :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Set Child_Iter to point to the row pointed to by Sorted_Iter.

```
procedure Reset_Default_Sort_Func
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record);
```

This resets the default sort function to be in the 'unsorted' state.
That is, it is in the same order as the child model. It will re-sort the model to be in the same order as the child model only if the Gtk_Tree_Model_Sort is in 'unsorted' state.

```
procedure Clear_Cache
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record);
```

This function should almost never be called. It clears the
tree_model_sort of any cached iterators that haven't been reffed with gtk.tree_model.ref_node. This might be useful if the child model being sorted is static (and doesn't change often) and there has been a lot of unreffed access to nodes. As a side effect of this function, all unreffed iters will be invalid.

```
function Iter_Is_Valid
  (Tree_Model_Sort     : access Gtk_Tree_Model_Sort_Record;
   Iter                :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Boolean;
```

WARNING: this function is slow. Only use if for debugging and/or
testing purposes. Checks if the given iter is a valid iter for this model.

198.1.1 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Tree_Sortable" This interface allows you to specify your own sort function

@item "Gtk_Tree_Drag_Source" This interface allows this widget to act as a dnd source @end itemize

```
function "+"
  (Model              : access Gtk_Tree_Model_Sort_Record'Class)
   return Gtk.Tree_Sortable.Gtk_Tree_Sortable;
function "-"
  (Sortable           :      Gtk.Tree_Sortable.Gtk_Tree_Sortable)
   return Gtk_Tree_Model_Sort;
```

Converts to and from the Gtk_Tree_Sortable interface

```
function "+"
  (Model              : access Gtk_Tree_Model_Sort_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Source;
function "-"
  (Drag_Source        :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Source)
   return Gtk_Tree_Model_Sort;
```

Converts to and from the Gtk_Tree_Drag_Source interface

199 Package Gtk.Tree_Selection

The Gtk_Tree_Selection object is a helper object to manage the selection for a Gtk_Tree_View widget. The Gtk_Tree_Selection object is automatically created when a new Gtk_Tree_View widget is created, and cannot exist independentally of this widget. The primary reason the Gtk_Tree_Selection objects exists is for cleanliness of code and API. That is, there is no conceptual reason all these functions could not be methods on the Gtk_Tree_View widget instead of separate function.

The Gtk_Tree_Selection object is gotten from a Gtk_Tree_View by calling Gtk.Tree_View.Get_Selection. It can be manipulated to check the selection status of the tree, as well as select and deselect individual rows. Selection is done completely view side. As a result, multiple views of the same model can have completely different selections. Additionally, you cannot change the selection of a row on the model that is not currently displayed by the view without expanding its parents first.

One of the important things to remember when monitoring the selection of a view is that the "changed" signal is mostly a hint. That is, it may only emit one signal when a range of rows is selected. Additionally, it may on occasion emit a "changed" signal when nothing has happened (mostly as a result of programmers calling select_row on an already selected row).

199.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Tree_Selection    (Package Gtk.Tree_Selection)
```

199.2 Signals

- **"changed"**

  ```
  procedure Handler
  (Widget : access Gtk_Tree_Selection_Record'Class'Class);
  ```

  Emitted whenever the selection has (possibly) changed. Please note that this signal is mostly a hint. It may only be emitted once when a range of rows are selected, and it may occasionally be emitted when nothing has happened.

199.3 Types

type Data_Type **is private**;

type Data_Type_Access **is access all** Data_Type;

type Foreach_Func **is access procedure**

199.4 Subprograms

```
function Get_Type               return Glib.GType;
```

Return the internal type associated with Gtk_Tree_Selection.

```
procedure Set_Mode
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   The_Type           :      Gtk_Selection_Mode);
function Get_Mode
  (Selection          : access Gtk_Tree_Selection_Record'Class)
   return Gtk_Selection_Mode;
```

Set the selection mode of the Selection.
If the previous type was Gtk_Selection_Multiple, then the anchor is kept selected, if it was previously selected.

```
function Get_Tree_View
  (Selection          : access Gtk_Tree_Selection_Record'Class)
   return Gtk.Widget.Gtk_Widget;
```

Return the tree view associated with Selection.

```
function Count_Selected_Rows
  (Selection          : access Gtk_Tree_Selection_Record)
   return Gint;
```

Returns the number of rows that have been selected.

```
procedure Get_Selected
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Model              : out   Gtk.Tree_Model.Gtk_Tree_Model;
   Iter               : out   Gtk.Tree_Model.Gtk_Tree_Iter);
```

Set Iter to the currently selected node if Selection
is set to Gtk_Selection_Single or Gtk_Selection_Browse. Iter is set to Null_Iter if no node is currently selected. Model is filled with the current model as a convenience. This function will not work if Selection is set to Gtk_Selection_Multiple.

```
procedure Get_Selected_Rows
  (Selection          : access Gtk_Tree_Selection_Record;
   Model              : out   Gtk.Tree_Model.Gtk_Tree_Model;
   Path_List          : out   Gtk.Tree_Model.Gtk_Tree_Path_List.Glist);
```

Creates a list of path of all selected rows. Additionally, if you are
planning on modifying the model after calling this function, you may want to convert the returned list into a list of Gtk_Tree_Row_Reference.

You must free the resulting list by calling Path_Free on each item, and then freeing the list itself.

```
procedure Selected_Foreach
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Func               :      Foreach_Func;
   Data               :      Data_Type_Access);
```

Call Func for each selected node.

```
procedure Select_Path
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
procedure Unselect_Path
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Selects or unselects the row at path.

```
function Path_Is_Selected
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Return True if the row pointed to by path is currently selected.
If path does not point to a valid location, False is returned

```
procedure Select_Iter
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter);
procedure Unselect_Iter
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Selects or unselects the row pointed to by the specified iterator.

```
function Iter_Is_Selected
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Boolean;
```

Return True if the row pointed to by path is currently selected.

```
procedure Select_All
  (Selection          : access Gtk_Tree_Selection_Record'Class);
procedure Unselect_All
  (Selection          : access Gtk_Tree_Selection_Record'Class);
```

Selects or unselects all the nodes.
Selection must be set to Gtk_Selection_Multiple mode.

```
procedure Select_Range
  (Selection          : access Gtk_Tree_Selection_Record'Class;
   Start_Path         :      Gtk.Tree_Model.Gtk_Tree_Path;
   End_Path           :      Gtk.Tree_Model.Gtk_Tree_Path);
procedure Unselect_Range
  (Selection          : access Gtk_Tree_Selection_Record;
   Start_Path         :      Gtk.Tree_Model.Gtk_Tree_Path;
   End_Path           :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Selects or unselects a range of nodes, determined by Start_Path and
End_Path inclusive

200 Package Gtk.Tree_Sortable

Gtk_Tree_Sortable is an interface to be implemented by tree models which support sorting. The Gtk_Tree_View uses the methods provided by this interface to sort the model.

200.1 Signals

- **"sort_column_changed"**

  ```
  procedure Handler (Sortable : Gtk_Tree_Sortable);
  ```

  Emitted when the sort column is changed through Set_Sort_Column_Id

200.2 Types

```
type Data_Type is private;
```

```
type Destroy_Notify is access procedure
        (Data : in out Data_Type);
```

Free the memory used by Data

```
type Gtk_Tree_Iter_Compare_Func is access function
        (Model      : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;
```

```
type Gtk_Tree_Sortable is new Glib.Types.GType_Interface;
```

200.3 Subprograms

```
function Get_Type                return Glib.GType;
```

Returns the internal type used for a Gtk_Tree_Sortable

```
procedure Set_Sort_Column_Id
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Column_Id     :      Gint;
   Order              :      Gtk.Enums.Gtk_Sort_Type);
procedure Get_Sort_Column_Id
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Column_Id     : out  Gint;
   Order              : out  Gtk.Enums.Gtk_Sort_Type);
```

Sets the current sort column to be Sort_Column_Id. The Sortable will resort itself to reflect this change, after emitting sort_column_changed signal. If Sort_Column_Id is Default_Sort_Column_Id, then the default sort function will be used, if it is set.

```
procedure Set_Default_Sort_Func
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Func          :      Gtk_Tree_Iter_Compare_Func);
```

```
function Has_Default_Sort_Func
  (Sortable           :      Gtk_Tree_Sortable)
   return Boolean;
```

Sets the default comparison function used when sorting to be Sort_Func.
If the current sort column id of Sortable is Default_Sort_Column_Id, then the model will sort using this function. If Sort_Func is null, then there will be no default comparison function. This means that once the model has been sorted, it can't go back to the default state. In this case, when the current sort column id of Sortable is Default_Sort_Column_Id, the model will be unsorted.

```
procedure Set_Sort_Func
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Column_Id     :      Gint;
   Sort_Func          :      Gtk_Tree_Iter_Compare_Func);
```

Sets the comparison function used when sorting to be Sort_Func. If the current sort column id of Sortable is the same as Sort_Column_Id, then the model will sort using this function.

```
procedure Set_Default_Sort_Func
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Func          :      Gtk_Tree_Iter_Compare_Func;
   User_Data          :      Data_Type;
   Destroy            :      Destroy_Notify := null);
procedure Set_Sort_Func
  (Sortable           :      Gtk_Tree_Sortable;
   Sort_Column_Id     :      Gint;
   Sort_Func          :      Gtk_Tree_Iter_Compare_Func;
   User_Data          :      Data_Type;
   Destroy            :      Destroy_Notify := null);
```

Same as above, but an additional user data can be passed to the sort function.

200.3.1 Signals

The following new signals are defined for this widget:

```
procedure Sort_Column_Changed
  (Sortable           :      Gtk_Tree_Sortable);
```

Emits sort_column_changed signal

201 Package Gtk.Tree_Store

This package implements a specific model to store your data in. It is basically similar to a small database, in that each field can contain any number of columns.

Each column can contain a different type of data, specified when the model is created.

Adding new values in the model is done as in the example at the end.

201.1 Types

type Data_Type **is private**;

type Data_Type_Access **is access all** Data_Type;

201.2 Subprograms

```
procedure Gtk_New
  (Tree_Store         : out   Gtk_Tree_Store;
   Types              :       GType_Array);
```

Create a new tree store using Types to fill the columns.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with this widget.

```
procedure Set_Column_Types
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Types              :       GType_Array);
```

This function is meant primarily for GObjects that inherit from Gtk_Tree_Store, and should only be used when constructing a new Gtk_Tree_Store. It will not function after a row has been added, or a method on the Gtk_Tree_Model interface is called.

```
procedure Set_Value
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Glib.Values.GValue);
```

Set a new value in the model. The value is added in the column Column, and in the line Iter. This is the most general of the Set procedures, since it allows you to control what should be done when the cell is freed (useful for instance for reference-controlled types). In particular, you would create a GValue of a special type derived from Boxed (see Glib.Value.Set_Boxed).

The type of the column must be of the type stored in the GValue itself. Referencing the example given for Set_Boxed, this would be the value in "Typ".

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record'Class;
   Iter               :       Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :       Gint;
   Value              :       Data_Type_Access);
```

Generic procedure used to store access objects in the model.
For GObject and all of its descendents (including all widgets), you should use the Set procedure below that takes a GObject as parameter.

Please see the example at the end for more information on how to create your own Set procedures adapted to your model. Also consider using Set_Value for complex cases

```
function Get
  (Tree_Store         : access Gtk_Tree_Store_Record'Class;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint)
   return Data_Type_Access;
```

Generic procedure used to get access objects back from the model.
For GObject and all of its descendents (including all widgets), you should use the Get_Object function defined in Gtk-Tree_Model that returns a GObject.

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Value              :      UTF8_String);
```

Same as Generic_Set, but tailored to use with a string.

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Value              :      Boolean);
```

Same as Generic_Set, but tailored to use with a boolean.

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Value              :      Gint);
```

Same as Generic_Set, but tailored to use with an integer.

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Value              :      Glib.C_Proxy);
```

Same as Generic_Set, but tailored for Gdk types.

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Address            :      System.Address);
```

Same as Generic_Set, for a generic address

```
procedure Set
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Column             :      Gint;
   Value              :      Glib.Object.GObject);
```

Same as Generic_Set, but tailored to objects/widgets.

```
procedure Remove
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter);
```

Remove Iter from Tree_Store.
After being removed, Iter is set to Null_Iter.

```
procedure Insert
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Position           :      Gint);
```

Create a new row at Position.
If parent is non-null, then the row will be made a child of Parent. Otherwise, the row will be created at the toplevel. If Position is larger than the number of rows at that level, then the new row will be inserted to the end of the list. Iter will be changed to point to this new row. The row will be empty before this function is called. To fill in values, you need to call Set_Value.

```
procedure Insert_Before
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Sibling            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Insert a new row before Sibling.
If Sibling is Null_Iter, then the row will be appended to the beginning of the Parent's children. If Parent and Sibling are Null_Iter, then the row will be appended to the toplevel. If both Sibling and Parent are set, then Parent must be the parent of Sibling. When Sibling is set, Parent is optional. Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value.

```
procedure Insert_After
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Sibling            :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Insert a new row after Sibling.
If Sibling is Null_Iter, then the row will be prepended to the beginning of the Parent's children. If Parent and Sibling are Null_Iter, then the row will be prepended to the toplevel. If both Sibling and Parent are set, then Parent must be the parent of Sibling. When Sibling is set, Parent is optional. Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value.

```
procedure Prepend
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Prepend a new row to Tree_Store.
If Parent is non-null, then it will prepend the new row before the first child of Parent, otherwise it will prepend a row to the top level. Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value. The efficiency of this procedure is O(N).

```
procedure Append
  (Tree_Store         : access Gtk_Tree_Store_Record;
```

```
   Iter               : in out Gtk.Tree_Model.Gtk_Tree_Iter;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Append a new row to Tree_Store.
If Parent is non-null, then it will append the new row after the last child of Parent, otherwise it will append a row to the top level. Iter will be changed to point to this new row. The row will be empty after this function is called. To fill in values, you need to call Set_Value. The efficiency of this procedure is O(N^2).

```
function Is_Ancestor
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Descendant         :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Boolean;
```

Return True if Iter is an ancestor of Descendant.
That is, Iter is the parent (or grandparent or great-grandparent) of Descendant.

```
function Iter_Depth
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Gint;
```

Returns the depth of Iter.
This will be 0 for anything on the root level, 1 for anything down a level, etc.

```
function Iter_Is_Valid
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter)
   return Boolean;
```

WARNING: This function is slow. Only use it for debugging and/or testing purposes. Checks if the given iter is a valid iter for Tree_Store.

```
procedure Move_After
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Position           :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Moves the row pointed to by Iter to the position after Position. Iter and Position should be in the same level. Note that this function only works with unsorted stores. If Position is Null_Iter, Iter will be moved to the start of the level.

```
procedure Move_Before
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Position           :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Moves the row pointed to by Iter to the position before Position. Iter and Position should be in the same level. Note that this function only works with unsorted stores. If Position is Null_Iter, Iter will be

```
procedure Clear
  (Tree_Store         : access Gtk_Tree_Store_Record);
```

Removes all rows from Tree_Store

```
procedure Reorder
  (Tree_Store         : access Gtk_Tree_Store_Record;
   Parent             :      Gtk.Tree_Model.Gtk_Tree_Iter;
   New_Order          :      Glib.Gint_Array);
```

Reorders the children of Parent to follow the order indicated by New_order. Note that this function only works with unsorted stores. New order is an array

of integers mapping the new position of each child to its old position before the re-ordering, i.e. New_order[newpos] = oldpos If Parent is Null_Iter, it also reorders the root nodes

```
procedure Swap
  (Tree_Store          : access Gtk_Tree_Store_Record;
   A                   :      Gtk.Tree_Model.Gtk_Tree_Iter;
   B                   :      Gtk.Tree_Model.Gtk_Tree_Iter);
```

Swaps the rows pointed to by A and B (in the same level). Note that this function only works with unsorted stores.

201.2.1 Sorting Freeze / Thaw

```
function Freeze_Sort
  (Tree                : access Gtk.Tree_Store.Gtk_Tree_Store_Record'Class)
   return Gint;
```

Freeze the sorting in the tree view, and returns the current sort_column_id, which should be used when thawing. (See Thaw_Sort)

```
procedure Thaw_Sort
  (Tree                : access Gtk.Tree_Store.Gtk_Tree_Store_Record'Class;
   Column_Id           :      Gint);
```

Thaw a frozen tree_view. Column_Id should be the value returned by the corresponding call to Freeze_Sort.

201.2.2 Interfaces

This class implements several interfaces. See Glib.Types@*

@itemize @bullet @item "Gtk_Tree_Sortable" This interface allows you to specify your own sort function

@item "Gtk_Tree_Drag_Source" This interface allows this widget to act as a dnd source

@item "Gtk_Tree_Drag_Dest" This interface allows this widget to act as a dnd destination @end itemize

```
function "+"
  (Model               : access Gtk_Tree_Store_Record'Class)
   return Gtk.Tree_Sortable.Gtk_Tree_Sortable;
function "-"
  (Sortable            :      Gtk.Tree_Sortable.Gtk_Tree_Sortable)
   return Gtk_Tree_Store;
```

Converts to and from the Gtk_Tree_Sortable interface

```
function "+"
  (Model               : access Gtk_Tree_Store_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Source;
function "-"
  (Drag_Source         :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Source)
   return Gtk_Tree_Store;
```

Converts to and from the Gtk_Tree_Drag_Source interface

```
function "+"
  (Model               : access Gtk_Tree_Store_Record'Class)
   return Gtk.Tree_Dnd.Gtk_Tree_Drag_Dest;
function "-"
  (Drag_Dest           :      Gtk.Tree_Dnd.Gtk_Tree_Drag_Dest)
   return Gtk_Tree_Store;
```

Converts to and from the Gtk_Tree_Drag_Source interface

201.3 Example

Adding a **new** line **in** the model:

```
declare
   Iter  : Gtk_Text_Iter;
   Value : Glib.Values.GValue;
begin
   Append (Model, Iter, Null_Iter);

   --  First method:

   Init (Value, GType_String);
   Set_String (Value, "foo");
   Set_Value (Model, Iter, 0, Value);
   Unref (Value);

   --  Second method:

   Set (Model, Iter, 0, "foo");
end;
```

Defining your own Set **function for** your model: This can be done by directly importing the C **function**, **with** the appropriate number **of** parameters. Remember that you are passing data directly to C, thus you need to **end** strings **with** ASCII.NUL

```
procedure My_Set
   (Tree_Store : access Gtk_Tree_Store_Record'Class;
    Iter       : Gtk.Tree_Model.Gtk_Tree_Iter;
    Column1 : Gint; Value1 : UTF8_String;
    Column2 : Gint; Value2 : Boolean)
is
    procedure Set_String
      (Tree : System.Address;
       Iter : Gtk.Tree_Model.Gtk_Tree_Iter;
       Column : Gint; Value : UTF8_String);
    pragma Import (C, Set_String, "ada_gtk_tree_store_set_ptr");

    procedure Set_Int
      (Tree : System.Address;
       Iter : Gtk.Tree_Model.Gtk_Tree_Iter;
       Column : Gint; Value : Gint);
    pragma Import (C, Internal, "ada_gtk_tree_store_set_int");
 begin
```

```
      Internal
        (Get_Object (Tree_Store), Iter, Column1, Value1 & ASCII.NUL);
      Internal
        (Get_Object (Tree_Store), Iter, Column2, Boolean'Pos (Value2));
   end Set;
```

202 Package Gtk.Tree_View

See extended documentation in Gtk.Tree_View_Column and Gtk.Tree_Store.

202.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Tree_View   (Package Gtk.Tree_View)
```

202.2 Signals

- **"columns_changed"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class);
  ```

- **"expand_collapse_cursor_row"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Logical : Boolean;
  Expand : Boolean;
  Open_All : Boolean);
  ```

- **"move_cursor"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Step : Gtk_Movement_Step;
  Count : Gint);
  ```

- **"row_activated"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Path : Gtk.Tree_Model.Gtk_Tree_Path;
  Column : Gtk.Tree_View_Column.Gtk_Tree_View_Column);
  ```

- **"row_collapsed"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Iter : access Gtk.Tree_Iter.Gtk_Tree_Iter_Record'Class;
  Path : Gtk.Tree_Model.Gtk_Tree_Path);
  ```

- **"row_expanded"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Iter : access Gtk.Tree_Iter.Gtk_Tree_Iter_Record'Class;
  Path : Gtk.Tree_Model.Gtk_Tree_Path);
  ```

- **"select_all"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class);
  ```

- **"select_cursor_parent"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class);
  ```

- **"select_cursor_row"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Start_Editing : Boolean);
  ```

- **"set_scroll_adjustments"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Record'Class;
  Hadjustment : access Gtk.Adjustment.Gtk_Adjustment_Record'Class;
  Vadjustment : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
  ```

- **"start_interactive_search"**

```
procedure Handler (Widget : access Gtk_Tree_View_Record'Class);
```

- **"test_collapse_row"**

```
function Handler (Widget : access Gtk_Tree_View_Record'Class;
Iter : access Gtk.Tree_Iter.Gtk_Tree_Iter_Record'Class;
Path : Gtk.Tree_Model.Gtk_Tree_Path)
return Gboolean;
```

- **"test_expand_row"**

```
function Handler (Widget : access Gtk_Tree_View_Record'Class;
Iter : access Gtk.Tree_Iter.Gtk_Tree_Iter_Record'Class;
Path : Gtk.Tree_Model.Gtk_Tree_Path)
return Gboolean;
```

- **"toggle_cursor_row"**

```
procedure Handler (Widget : access Gtk_Tree_View_Record'Class);
```

202.3 Types

```
type Gtk_Tree_View_Column_Drop_Func is access function
          (Tree_View   : System.Address; --  Gtk_Tree_View
           Column      : System.Address; --  Gtk_Tree_View_Column
           Prev_Column : System.Address; --  Gtk_Tree_View_Column
           Next_Column : System.Address; --  Gtk_Tree_View_Column
           User_Data   : System.Address) return Gboolean;
```

```
type Gtk_Tree_View_Drop_Position is
          (Tree_View_Drop_Before,
           Tree_View_Drop_After,
           Tree_View_Drop_Into_Or_Before,
           Tree_View_Drop_Into_Or_After);
```

```
type Gtk_Tree_View_Mapping_Func is access procedure
          (Tree_View : System.Address;  --  Gtk_Tree_View
           Path      : Gtk.Tree_Model.Gtk_Tree_Path;
           User_Data : System.Address);
```

```
type Gtk_Tree_View_Row_Separator_Func is access function
          (Model     : System.Address;
           Iter      : Gtk.Tree_Model.Gtk_Tree_Iter;
           User_Data : System.Address) return Gboolean;
```

```
type Gtk_Tree_View_Search_Equal_Func is access function
          (Model  : System.Address;
           Column : Gint;
           Key    : Interfaces.C.Strings.chars_ptr;
           Iter   : Gtk.Tree_Model.Gtk_Tree_Iter;
```

```
   User_Data : System.Address) return Gboolean;
```

202.4 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Tree_View);
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with this widget.

```
procedure Gtk_New
  (Widget             : out   Gtk_Tree_View;
   Model              : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class);
procedure Set_Model
  (Tree_View          : access Gtk_Tree_View_Record;
   Model              :      Gtk.Tree_Model.Gtk_Tree_Model);
function Get_Model
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Tree_Model.Gtk_Tree_Model;
```

Sets the model for a Gtk_Tree_View. If the Tree_View already has a model set, it will remove it before setting the new model. If Model is Null, then it will unset the old model.

```
function Get_Selection
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Tree_Selection.Gtk_Tree_Selection;
```

Gets the Gtk_Tree_Selection associated with Tree_View.

```
procedure Set_Hadjustment
  (Tree_View          : access Gtk_Tree_View_Record;
   Adjustment         : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
function Get_Hadjustment
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Sets or gets the Gtk_Adjustment for the current horizontal aspect.

```
procedure Set_Vadjustment
  (Tree_View          : access Gtk_Tree_View_Record;
   Adjustment         : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
function Get_Vadjustment
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Adjustment.Gtk_Adjustment;
```

Sets or Gets the Gtk_Adjustment currently being used for the vertical aspect.

202.4.1 Column and header operations

```
procedure Set_Headers_Visible
  (Tree_View          : access Gtk_Tree_View_Record;
   Headers_Visible    :      Boolean);
function Get_Headers_Visible
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

Returns True if the headers on the Tree_View are visible.

```
procedure Columns_Autosize
  (Tree_View          : access Gtk_Tree_View_Record);
```

Resizes all columns to their optimal width.

```
procedure Set_Headers_Clickable
  (Tree_View          : access Gtk_Tree_View_Record;
   Setting            :      Boolean);
```

Allow the column title buttons to be clicked.

```
procedure Set_Rules_Hint
  (Tree_View          : access Gtk_Tree_View_Record;
   Setting            :      Boolean);

function Get_Rules_Hint
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

This function tells GtkAda that the user interface for your
application requires users to read across tree rows and associate cells with one another. By default, GtkAda will then render the tree with alternating row colors. Do *not* use it just because you prefer the appearance of the ruled tree; that's a question for the theme. Some themes will draw tree rows in alternating colors even when rules are turned off, and users who prefer that appearance all the time can choose those themes. You should call this function only as a *semantic* hint to the theme engine that your tree makes alternating colors useful from a functional standpoint (since it has lots of columns, generally).

202.4.2 Public Column functions

```
function Append_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column)
   return Gint;
```

Append Column to the list of columns.

```
function Remove_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column)
   return Gint;
```

Remove Column from Tree_View.
Return value: The number of columns in Tree_View after removing.

```
function Insert_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Position           :      Gint := -1)
   return Gint;
```

Insert the Column into the Tree_View at Position.
If Position is -1, then the column is inserted at the end. Return the number of columns in Tree_View after insertion.

```
function Insert_Column_With_Data_Func
  (Tree_View          : access Gtk_Tree_View_Record;
   Position           :      Gint;
   Title              :      String;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Func               :      Gtk.Tree_View_Column.Cell_Data_Func)
   return Gint;
```

Convenience function that inserts a new column into the tree view
with the given cell renderer and a function to set cell renderer attributes (normally using data from the model). See also Gtk.Tree_View_Column.Set_Cell_Data_Func and Gtk.Tree_View_Column.Pack_Start. If Tree_View has "fixed_height" mode enabled, then Column must have its "sizing" property set to be TREE_VIEW_COLUMN_FIXED.

Return value: number of columns in the tree view post-insert

```
function Get_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   N                  :      Gint)
   return Gtk.Tree_View_Column.Gtk_Tree_View_Column;
```

Gets the Gtk_Tree_ViewColumn at the given position in the Tree_View.

```
function Get_Columns
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Tree_View_Column.Column_List.Glist;
```

Return a list of all the Gtk_Tree_View_Column s currently in Tree_View.
The returned list must be freed with g_list_free ().

```
procedure Move_Column_After
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Base_Column        :      Gtk.Tree_View_Column.Gtk_Tree_View_Column);
```

Move Column to be after to Base_Column. If Base_Column is Null, then
Column is placed in the first position.

```
procedure Set_Expander_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column);
```

Set the column to draw the expander arrow at. It must be in Tree_View.
If Column is Null, then the expander arrow is fixed at the first column.

```
function Get_Expander_Column
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk.Tree_View_Column.Gtk_Tree_View_Column;
```

Return the column that is the current expander column.
This column has the expander arrow drawn next to it.

```
procedure Scroll_To_Point
  (Tree_View          : access Gtk_Tree_View_Record;
   Tree_X             :      Gint;
   Tree_Y             :      Gint);
```

Scroll the tree view such that the top-left corner of the visible
area is Tree_X, Tree_Y, where Tree_X and Tree_Y are specified in tree window coordinates. The Tree_View must be realized before this function is called. If it isn't, you probably want to be using Scroll_To_Cell.

```
procedure Scroll_To_Cell
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Use_Align          :      Boolean;
   Row_Align          :      Gfloat;
   Col_Align          :      Gfloat);
```

Move the alignments of Tree_View to the position specified by Column and
Path. If Column is Null, then no horizontal scrolling occurs. Likewise, if Path is Null

no vertical scrolling occurs. Row_Align determines where the row is placed, and Col_align determines where Column is placed. Both are expected to be between 0.0 and 1.0. 0.0 means left/top alignment, 1.0 means right/bottom alignment, 0.5 means center. If Use_Align is False, then the alignment arguments are ignored, and the tree does the minimum amount of work to scroll the cell onto the screen.

```
procedure Get_Visible_Range
  (Tree_View          : access Gtk_Tree_View_Record;
   Start_Path         : out   Gtk.Tree_Model.Gtk_Tree_Path;
   End_Path           : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Success            : out   Boolean);
```

Sets Start_path and End_path to be the first and last visible path.
Note that there may be invisible paths in between. The paths should be freed with Free after use.

```
procedure Row_Activated
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column);
```

Activate the cell determined by Path and Column.

```
procedure Expand_All
  (Tree_View          : access Gtk_Tree_View_Record);
```

Recursively expand all nodes in the Tree_View.

```
procedure Collapse_All
  (Tree_View          : access Gtk_Tree_View_Record);
```

Recursively collapse all visible, expanded nodes in Tree_View.

```
function Expand_Row
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Open_All           :      Boolean)
   return Boolean;
```

Open the row so its children are visible
Return True if the row existed and had children

```
procedure Expand_To_Path
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Expands the row at Path. This will also expand all parent rows of
Path as necessary.

```
procedure Map_Expanded_Rows
  (Tree_View          : access Gtk_Tree_View_Record;
   Func               :      Gtk_Tree_View_Mapping_Func;
   Data               :      System.Address);
```

Calls Func on all expanded rows.

```
function Collapse_Row
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Collapse a row (hides its child rows, if they exist.)

```
function Row_Expanded
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Boolean;
```

Return True if the node pointed to by Path is expanded in Tree_View.

```
procedure Set_Fixed_Height_Mode
  (Tree_View          : access Gtk_Tree_View_Record;
   Enable             :      Boolean);
function Get_Fixed_Height_Mode
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

Enables or disables the fixed height mode of tree_view.
Fixed height mode speeds up the rendering by assuming that all rows have the same height. Only enable this option if all rows are the same height and all columns are of type TREE_VIEW_COLUMN_FIXED.

```
procedure Set_Hover_Expand
  (Tree_View          : access Gtk_Tree_View_Record;
   Expand             :      Boolean);
function Get_Hover_Expand
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

Enables of disables the hover expansion mode of Tree_view.
Hover expansion makes rows expand or collaps if the pointer moves over them.

```
procedure Set_Hover_Selection
  (Tree_View          : access Gtk_Tree_View_Record;
   Hover              :      Boolean);
function Get_Hover_Selection
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

Enables of disables the hover selection mode of Tree_View.
Hover selection makes the selected row follow the pointer. Currently, this works only for the selection modes SELECTION_SINGLE and SELECTION_BROWSE.

```
procedure Set_Cursor
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Focus_Column       :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Start_Editing      :      Boolean);
```

Sets the current keyboard focus to be at Path, and selects it. This is
useful when you want to focus the user's attention on a particular row. If Column is not Null, then focus is given to that column. Additionally, if Column is specified, and Start_Editing is True, then editing should be started in the specified cell. Keyboard focus is given to the widget after this is called. Please note that editing can only happen when the widget is realized.

```
procedure Get_Cursor
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               : out    Gtk.Tree_Model.Gtk_Tree_Path;
   Focus_Column       : out    Gtk.Tree_View_Column.Gtk_Tree_View_Column);
```

Fills in Path and Focus_Column with the current path and focus column.
If the cursor isn't currently set, then *path will be Null. If no column currently has focus, then *focus_column will be Null.

```
procedure Set_Cursor_On_Cell
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Focus_Column       :      Gtk.Tree_View_Column.Gtk_Tree_View_Column
```

```
                    := null;
   Focus_Cell       :      Gtk.Cell_Renderer.Gtk_Cell_Renderer
                    := null;
   Start_Editing    :      Boolean);
```

Sets the current keyboard focus to be atPath, and selects it. This is
useful when you want to focus the user's attention on a particular row. If Focus_Column is not null, then focus is given to the column specified by it. If Focus_Column and Focus_Cell are not null, and Focus_Column contains 2 or more editable or activatable cells, then focus is given to the cell specified by Focus_Cell. Additionally, if Focus_Column is specified, and Start_Editing is true, then editing should be started in the specified cell. This function is often followed by gtk.widget.grab_focus (Tree_View) in order to give keyboard focus to the widget. Please note that editing can only happen when the widget is realized.

```
function Get_Bin_Window
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gdk.Window.Gdk_Window;
```

Return the window that Tree_View renders to.
This is used primarily to compare to Get_Window (Event) to confirm that the event on Tree_View is on the right window.

```
procedure Set_Row_Separator_Func
  (Tree_View          : access Gtk_Tree_View_Record;
   Func               :      Gtk_Tree_View_Row_Separator_Func;
   Data               :      System.Address;
   Destroy            :      Glib.G_Destroy_Notify_Address
                      := null);
function Get_Row_Separator_Func
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk_Tree_View_Row_Separator_Func;
```

Sets the row separator function, which is used to determine
whether a row should be drawn as a separator. If the row separator function is NULL, no separators are drawn. This is the default value.

```
procedure Get_Path_At_Pos
  (Tree_View          : access Gtk_Tree_View_Record;
   X                  :      Gint;
   Y                  :      Gint;
   Path               : out  Gtk.Tree_Model.Gtk_Tree_Path;
   Column             : out  Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Cell_X             : out  Gint;
   Cell_Y             : out  Gint;
   Row_Found          : out  Boolean);
```

Find the path at the point (X, Y) relative to Window.
If Window is null, then the point is found relative to the widget coordinates. This function is expected to be called after an event. It is primarily for things like popup menus. Path will be filled with the Gtk_Tree_Path at that point. It should be freed with Tree_Path_Free. Column will be filled with the column at that point. Cell_X and Cell_Y return the coordinates relative to the cell background (i.e. the background_area passed to gtk_cell_renderer_render()). This function only works if Tree_View is realized. Row_Found is set to True if a row exists at that coordinate.

```
procedure Get_Cell_Area
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
```

```
   Column              :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Rect                : out  Gdk.Rectangle.Gdk_Rectangle);
```

Fills the bounding rectangle in tree window coordinates for the cell at the row specified by Path and the column specified by Column. If Path is Null, or points to a path not currently displayed, the Y and Height fields of the rectangle will be filled with 0. If Column is Null, the X and Width fields will be filled with 0. The sum of all cell rects does not cover the entire tree; there are extra pixels in between rows, for example. The returned rectangle is equivalent to the Cell_Area passed to gtk_cell_renderer_render(). This function is only valid if Tree_View is realized.

```
procedure Get_Background_Area
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Column             :      Gtk.Tree_View_Column.Gtk_Tree_View_Column;
   Rect               : out  Gdk.Rectangle.Gdk_Rectangle);
```

Fills the bounding rectangle in tree window coordinates for the cell at the row specified by Path and the column specified by Column. If Path is Null, or points to a node not found in the tree, the Y and Height fields of the rectangle will be filled with 0. If Column is Null, the X and Width fields will be filled with 0. The returned rectangle is equivalent to the Background_Area passed to Gtk.Cell_Renderer.Render. These background areas tile to cover the entire tree window (except for the area used for header buttons). Contrast with the cell_area, returned by gtk_tree_view_get_cell_area(), which returns only the cell itself, excluding surrounding borders and the tree expander area.

```
procedure Get_Visible_Rect
  (Tree_View          : access Gtk_Tree_View_Record;
   Visible_Rect       : out  Gdk.Rectangle.Gdk_Rectangle);
```

Fills Visible_Rect with the currently-visible region of the buffer, in tree coordinates. Convert to widget coordinates with gtk_tree_view_tree_to_widget_coords(). Tree coordinates start at 0,0 for row 0 of the tree, and cover the entire scrollable area of the tree.

```
procedure Widget_To_Tree_Coords
  (Tree_View          : access Gtk_Tree_View_Record;
   Wx                 :      Gint;
   Wy                 :      Gint;
   Tx                 : out  Gint;
   Ty                 : out  Gint);
```

Converts widget coordinates to coordinates for the tree window (the full scrollable area of the tree).

```
procedure Tree_To_Widget_Coords
  (Tree_View          : access Gtk_Tree_View_Record;
   Tx                 :      Gint;
   Ty                 :      Gint;
   Wx                 : out  Gint;
   Wy                 : out  Gint);
```

Converts tree coordinates (coordinates in full scrollable area of the tree) to widget coordinates.

202.4.3 Searching

```
procedure Set_Enable_Search
  (Tree_View          : access Gtk_Tree_View_Record;
   Enable_Search      :      Boolean);
```

```
function Get_Enable_Search
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

If enable_search is set, then the user can type in text to search through the tree interactively (this is sometimes called "typeahead find"). Note that even if this is FALSE, the user can still initiate a search using the "start-interactive-search" key binding.

```
procedure Set_Search_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gint);
function Get_Search_Column
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gint;
```

Sets column as the column where the interactive search code should search in. If the sort column is set, users can use the "start-interactive-search" key binding to bring up search popup. The enable-search property controls whether simply typing text will also start an interactive search. Note that column refers to a column of the model.

```
procedure Set_Search_Equal_Func
  (Tree_View          : access Gtk_Tree_View_Record;
   Search_Equal_Func  :      Gtk_Tree_View_Search_Equal_Func;
   Search_User_Data   :      System.Address;
   Search_Destroy     :      G_Destroy_Notify_Address
                      := null);
function Get_Search_Equal_Func
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gtk_Tree_View_Search_Equal_Func;
```

Sets the compare function for the interactive search capabilities

202.4.4 Tooltips

```
procedure Set_Tooltip_Column
  (Tree_View          : access Gtk_Tree_View_Record;
   Column             :      Gint);
function Get_Tooltip_Column
  (Tree_View          : access Gtk_Tree_View_Record)
   return Gint;
```

If you only plan to have simple (text-only) tooltips on full rows, you can use this function to have GtkTreeView handle these automatically for you. column should be set to the column in tree_view's model containing the tooltip texts, or -1 to disable this feature.

When enabled, "has-tooltip" will be set to TRUE and tree_view will connect a "query-tooltip" signal handler.

Note that the signal handler sets the text with gtk_tooltip_set_markup(), so &, <, etc have to be escaped in the text.

```
procedure Get_Tooltip_Context
  (Tree_View          : access Gtk_Tree_View_Record;
   X                  : in out Glib.Gint;
   Y                  : in out Glib.Gint;
   Keyboard_Mode      :      Boolean;
   Model              : out   Gtk.Tree_Model.Gtk_Tree_Model;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
```

```
   Iter               : out   Gtk.Tree_Model.Gtk_Tree_Iter;
   Success            : out   Boolean);
```

This function is supposed to be used in a "query-tooltip" signal handler for GtkTreeView. The x, y and keyboard_tip values which are received in the signal handler, should be passed to this function without modification.

The Success indicates whether there is a tree view row at the given coordinates (True) or not (False) for mouse tooltips. For keyboard tooltips the row returned will be the cursor row. When True, then any of model, path and iter which have been provided will be set to point to that row and the corresponding model. x and y will always be converted to be relative to tree_view's bin_window if keyboard_tooltip is False.

```
procedure Set_Tooltip_Row
  (Tree_View          : access Gtk_Tree_View_Record;
   Tooltip            : access Gtk.Tooltips.Gtk_Tooltips_Record'Class;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path);
```

Sets the tip area of tooltip to be the area covered by the row at path. See also Set_Tooltip_Column for a simpler alternative. See also Gtk.Tooltips.Set_Tip_Area.

202.4.5 Columns reordering

```
procedure Set_Reorderable
  (Tree_View          : access Gtk_Tree_View_Record;
   Reorderable        :      Boolean);
function Get_Reorderable
  (Tree_View          : access Gtk_Tree_View_Record)
   return Boolean;
```

This function is a convenience function to allow you to reorder models that support the Gtk_Drag_Source_Iface and the Gtk_Drag_Dest_Iface. Both Gtk_Tree_Store and Gtk_List_Store support these. If Reorderable is True, then the user can reorder the model by dragging and dropping columns. The developer can listen to these changes by connecting to the model's signals. This function does not give you any degree of control over the order

- any reorderering is allowed. If more control is needed, you should probably handle drag and drop manually.

```
procedure Set_Column_Drag_Function
  (Tree_View          : access Gtk_Tree_View_Record;
   Func               :      Gtk_Tree_View_Column_Drop_Func;
   User_Data          :      System.Address;
   Destroy            :      Glib.G_Destroy_Notify_Address);
```

Sets a user function for determining where a column may be dropped. If Func is set to be %NULL, then Tree_View reverts to the default behavior of allowing all columns to be dropped everywhere.

202.4.6 Drag-and-drop

```
procedure Enable_Model_Drag_Dest
  (Tree_View          : access Gtk_Tree_View_Record;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Actions            :      Gdk.Dnd.Drag_Action);
```

Turns Tree_View into a drop destination for automatic drag-and-drop. Targets is the table of targets that the drag will support. Actions is a bitmask of possible actions for a drag to this widget.

```
procedure Enable_Model_Drag_Source
  (Tree_View          : access Gtk_Tree_View_Record;
   Start_Button_Mask  :      Gdk.Types.Gdk_Modifier_Type;
   Targets            :      Gtk.Selection.Target_Entry_Array;
   Actions            :      Gdk.Dnd.Drag_Action);
```

Turns Tree_View into a drag source for automatic DND.
Targets is the list of targets that the drag will support. Actions is the bitmask of possible actions for a drag from this widget. Start_Button_Mask is the mask of allowed buttons to start the drag. You need to connect to the usual dnd signals (see gtk-dnd.ads) to provide the actual data upon request.

```
procedure Unset_Rows_Drag_Source
  (Tree_View          : access Gtk_Tree_View_Record);
```

Undoes the effect of Enable_Model_Drag_Source.

```
procedure Unset_Rows_Drag_Dest
  (Tree_View          : access Gtk_Tree_View_Record);
```

Undoes the effect of Enable_Model_Drag_Dest.

```
function Create_Row_Drag_Icon
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path)
   return Gdk.Pixmap.Gdk_Pixmap;
```

Creates a Gdk_Pixmap representation of the row at path. This image is used for a drag icon. The returned pixmap must be freed by the user

```
procedure Get_Dest_Row_At_Pos
  (Tree_View          : access Gtk_Tree_View_Record;
   Drag_X             :      Gint;
   Drag_Y             :      Gint;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                : out   Gtk_Tree_View_Drop_Position;
   Success            : out   Boolean);
```

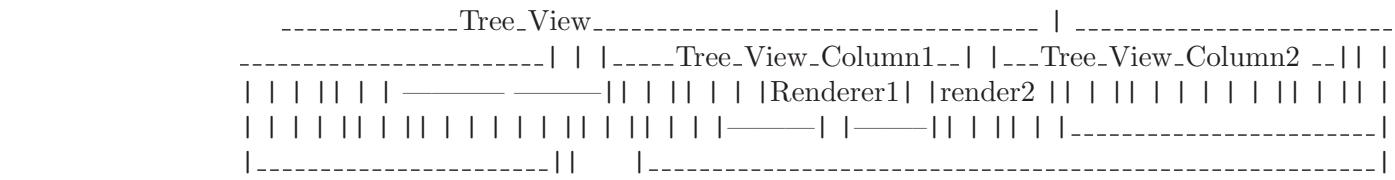
Determines the destination row for a given position.
(Drag_X, Drag_Y) is the position to determine the destination row for.

```
procedure Set_Drag_Dest_Row
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               :      Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                :      Gtk_Tree_View_Drop_Position);
procedure Get_Drag_Dest_Row
  (Tree_View          : access Gtk_Tree_View_Record;
   Path               : out   Gtk.Tree_Model.Gtk_Tree_Path;
   Pos                : out   Gtk_Tree_View_Drop_Position);
```

Sets or gets information about the row that is highlighted for feedback.

203 Package Gtk.Tree_View_Column

General organization of the tree_view widgets: @example



@end example

A tree view can contain multiple physical columns on the screen. These columns can have a button at the top, typically to force an ordering of the tree). They can also be reorganized interactively by the user.

Each physical column can display several information, like buttons, strings, ... Each of this display comes from a cell_renderer, that displays some data it reads from the model associated with the tree_view.

The renderers are then divided into lines, which are typically pointed to by iterators (Gtk_Tree_Iter).

203.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Tree_View_Column  (Package Gtk.Tree_View_Column)
```

203.2 Signals

- **"clicked"**

  ```
  procedure Handler (Widget : access Gtk_Tree_View_Column_Record'Class);
  ```

203.3 Types

type Cell_Data_Func **is access procedure**

type Data_Type **is private**;

```
type Gtk_Tree_View_Column_Sizing is
          (Tree_View_Column_Grow_Only,
           Tree_View_Column_Autosize,
           Tree_View_Column_Fixed);
```

203.4 Subprograms

```
procedure Convert;
procedure Convert;
procedure Gtk_New
  (Widget             : out   Gtk_Tree_View_Column);
function Get_Type             return Glib.GType;
```

Return the internal value associated with this widget.

203.4.1 Visual representation of the data

All the cells in a column have a similar graphical representation. This@* could be either a simple text, an editable text, a toggle button, ... This visual representation is independent from the actual data to represent. For instance, the same data from the model could be used for two different columns, once for a text and once for a button.

The visual representation is specified through a "renderer". See the various Gtk.Cell_Renderer* packages for more information on the available renderers.

Note that the same renderer can be used for multiple columns, even though its properties can be different each time. This means that for instance you can instantiate only one Gtk_Cell_Renderer_Text, and use it for all the columns that need to display text.

```
procedure Pack_Start
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Expand             :      Boolean);
```

Add a renderer to the Tree_Column.
Multiple renderers can be put in a specific column, and each of them can be associated with different data from the model. This provides a very powerful way to display different data in the same column.

```
procedure Pack_End
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Expand             :      Boolean);
```

Same as the above. See the description of Pack_Start and Pack_End in Gtk.Box for the precise difference between the two

```
procedure Clear
  (Tree_Column        : access Gtk_Tree_View_Column_Record);
```

Remove all the renderers set in the column.
The column will always be empty until you put some new renderers.

```
function Get_Cell_Renderers
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gtk.Cell_Renderer.Cell_Renderer_List.Glist;
```

Return the list of cell renderers set in the column. The returned list must be freed by the caller.

203.4.2 Specifying the data to display

The data to display in a column is always read from the model associated@* with the tree. In some cases (like if you are using the Gtk_Tree_Store model), this means that is has to be physically stored in a data structure. However, if you define your own models, you could also compute it on the fly.

For instance, if you have a database that contains some distance and time information, and you want to display the speed in a tree view: if you are using a Gtk_Tree_Store model, you have to create a third column in the model to store the string, and have a renderer point to that third column.

However, if you are using your own model, it is conceivable that the speed is computed on the fly from the distance and time.

The subprograms below use two or three parameters to precisely identify the part of the tree they impact: the column, the renderer in the column, and in some cases the specific line.

A renderer is always associated with a column in the model (even if that is a virtual column not associated with physical data). This is done through the Add_Attribute subprogram. This will read the data from the model. The type of the data read depends on the type of the column in the model. The type of data that Add_Attribute excepts to find in the column is documented in the packages for each of the renderer.

```
procedure Add_Attribute
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell_Renderer      : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Attribute          :      String;
   Column             :      Gint);
```

Add an attribute mapping to the list in Tree_Column.
The Column is the column of the model to get a value from, and the Attribute is the parameter on Cell_Renderer to be set from the value. So for example if column 2 of the model contains strings, you could have the "text" attribute of a Gtk_Cell_Renderer_Text get its values from column 2.

For a list of properties available for each Cell_Renderer, please refer to the corresponding package specifications.

See also the function Set_Cell_Data_Func for another way to query the data to display in the tree.

```
procedure Set_Cell_Data_Func
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Func               :      Cell_Data_Func);
```

Set the function to use for the column.
This function is used instead of the standard attributes mapping for setting the column value, and should set the value of Tree_Column's cell renderer as appropriate. Func may be null to remove an older one. It should be used when values from the model should be computed from application-specific data structures rather than stored in the model.

```
procedure Set_Cell_Data_Func
  (Tree_Column        : access Gtk_Tree_View_Column_Record'Class;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Func               :      Cell_Data_Func;
   Data               :      Data_Type);

procedure Clear_Attributes
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell_Renderer      : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class);
```

Clear all existing attributes previously set with
Gtk.Tree_View_Column.Set_Attributes.

203.4.3 Options for manipulating the columns

```
procedure Set_Spacing
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Spacing            :      Gint);
function Get_Spacing
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Set the spacing field of Tree_Column.
The spacing field is the number of pixels to place between cell renderers packed into it.

```
procedure Set_Visible
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Visible            :      Boolean);
function Get_Visible
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Set the visibility of Tree_Column.

```
procedure Set_Resizable
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Resizable          :      Boolean);
function Get_Resizable
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Set whether the Tree_Column is resizable.

```
procedure Set_Sizing
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   The_Type           :      Gtk_Tree_View_Column_Sizing);
function Get_Sizing
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gtk_Tree_View_Column_Sizing;
```

Set the growth behavior of Tree_Column to The_Type.

```
function Get_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Return the current size of the Tree_Column in pixels.

```
procedure Queue_Resize
  (Tree_Column        : access Gtk_Tree_View_Column_Record);
```

Flags the column, and the cell renderers added to this column, to have
their sizes renegotiated.

```
procedure Set_Fixed_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Fixed_Width        :      Gint);
function Get_Fixed_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Set the size of the column in pixels.
This is meaningful only if the sizing type is Gtk_Tree_View_Column_Fixed. In this case, the value is discarded as the size of the column is based on the calculated width of the column. The width is clamped to the min/max width for the column. The value returned by Get_Fixed_width may not be the actual width of the column on the screen, just what is requested.

```
procedure Set_Min_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Min_Width          :      Gint);

function Get_Min_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Set the minimum width of the Tree_Column.
If Min_Width is -1, then the minimum width is unset.

```
procedure Set_Max_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Max_Width          :      Gint);

function Get_Max_Width
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Set the maximum width of the Tree_Column.
If Max_Width is -1, then the maximum width is unset. Note, the column can actually be wider than max width if it's the last column in a view. In this case, the column expands to fill the view.

```
procedure Clicked
  (Tree_Column        : access Gtk_Tree_View_Column_Record);
```

Emit the "clicked" signal on the column.
This function will only work if the user could have conceivably clicked on the button.

```
procedure Set_Expand
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Expand             :      Boolean);

function Get_Expand
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Sets the column to take available extra space. This space is shared
equally amongst all columns that have the expand set to TRUE. If no column has this option set, then the last column gets all extra space. By default, every column is created with this FALSE.

```
procedure Set_Title
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Title              :      UTF8_String);

function Get_Title
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return UTF8_String;
```

Set the title of the Tree_Column.
If a custom widget has been set, then this value is ignored.

```
procedure Set_Clickable
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Clickable          :      Boolean);

function Get_Clickable
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Set the header to be active if Active is True.
When the header is active, then it can take keyboard focus, and can be clicked.

```
procedure Set_Widget
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Widget             : access Gtk.Widget.Gtk_Widget_Record'Class);
function Get_Widget
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the Gtk_Widget in the button in the column header.
If a custom widget has not been set, then this will be a Gtk_Alignment with a Gtk_Label in it.

```
procedure Set_Alignment
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Xalign             :      Gfloat);
function Get_Alignment
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gfloat;
```

Set the alignment of the title or custom widget inside the column header
The alignment determines its location inside the button 0.0 for left, 0.5 for center, 1.0 for right.

```
procedure Set_Reorderable
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Reorderable        :      Boolean);
function Get_Reorderable
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Whether this column can be drag-and-dropped to some other place in the tree.

```
procedure Set_Sort_Column_Id
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Sort_Column_Id     :      Gint);
function Get_Sort_Column_Id
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gint;
```

Set the logical model columns that this column sorts on when this column is selected for sorting. Doing so makes the column header clickable. Get_Sort_Column_Id returns -1 if this column can't be used for sorting.

```
procedure Set_Sort_Indicator
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Setting            :      Boolean);
function Get_Sort_Indicator
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Call this function with a Setting of True to display an arrow in the header button indicating the column is sorted. Call Set_Sort_Order to change the direction of the arrow.

```
procedure Set_Sort_Order
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Order              :      Gtk_Sort_Type);
function Get_Sort_Order
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Gtk_Sort_Type;
```

Change the appearance of the sort indicator.
This does *not* actually sort the model. Use Gtk.Tree_View_Column.Set_Sort_Column_Id if you want automatic sorting support. This function is primarily for custom sorting behavior, and should be used in conjunction with Gtk.Tree_Sortable.Set_Sort_Column to do that. For custom models, the mechanism will vary. The sort indicator changes direction to indicate normal sort or reverse sort. Note that you must have the sort indicator enabled to see anything when calling this function; see Set_Sort_Indicator.

```
procedure Cell_Set_Cell_Data
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Tree_Model         : access Gtk.Tree_Model.Gtk_Tree_Model_Record'Class;
   Iter               :      Gtk.Tree_Model.Gtk_Tree_Iter;
   Is_Expander        :      Boolean;
   Is_Expanded        :      Boolean);
```

Set the cell renderer based on the Tree_Model and Tree_Node.
That is, for every attribute mapping in Tree_Column, it will get a value from the set column on the Tree_Node, and use that value to set the attribute on the cell renderer. This is used primarily by the Gtk_Tree_View.

```
procedure Cell_Get_Size
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell_Area          :      Gdk.Rectangle.Gdk_Rectangle;
   X_Offset           : out   Gint;
   Y_Offset           : out   Gint;
   Width              : out   Gint;
   Height             : out   Gint);
```

Obtain the width and height needed to render the column.
This is used primarily by the Gtk_Tree_View.

```
function Cell_Is_Visible
  (Tree_Column        : access Gtk_Tree_View_Column_Record)
   return Boolean;
```

Returns true if any of the cells packed in the column is visible

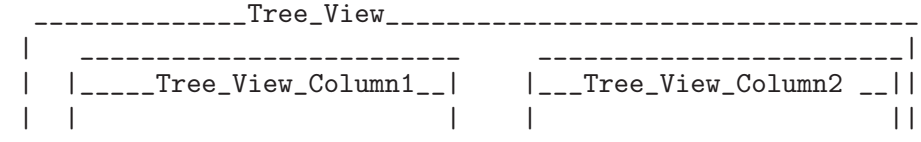
```
procedure Cell_Get_Position
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell_Renderer      : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class;
   Start_Pos          : out   Gint;
   Width              : out   Gint;
   Success            : out   Boolean);
```

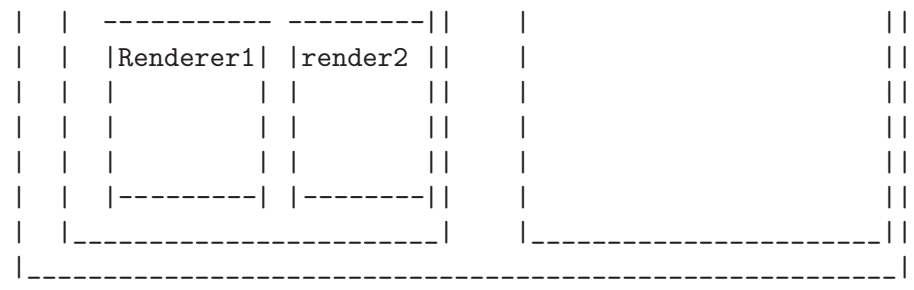
Obtains the horizontal position and size of a cell in a column. If the
cell is not found in the column, start_pos and width are not changed and FALSE is returned.

```
procedure Focus_Cell
  (Tree_Column        : access Gtk_Tree_View_Column_Record;
   Cell               : access Gtk.Cell_Renderer.Gtk_Cell_Renderer_Record'Class);
```

Sets the current keyboard focus to be at Cell, if the column contains
2 or more editable and activatable cells.

203.5 Example




Renderer1
render2

204 Package Gtk.Type_Conversion

Provides full dynamic typing within GtkAda. Note that this package is obsolete, since GtkAda now provides this capability by default.

204.1 Subprograms

procedure `Init;`

No-op, GtkAda provides automatic conversion of C widgets to Ada types automatically.

205 Package Gtk.UI_Manager

A Gtk_UI_Manager constructs a user interface (menus and toolbars) from one or more UI definitions, which reference actions from one or more action groups.

————————- UI Definitions – ————————-

The UI definitions are specified in an XML format which can be roughly described by the following DTD. @itemize @bullet @item <!ELEMENT ui (menubar | toolbar | popup | accelerator)* > @item <!ELEMENT menubar (menuitem | separator | placeholder | menu)* > @item <!ELEMENT menu (menuitem | separator | placeholder | menu)* > @item <!ELEMENT popup (menuitem | separator | placeholder | menu)* > @item <!ELEMENT toolbar (toolitem | separator | placeholder)* > @item <!ELEMENT placeholder (menuitem | toolitem | separator | placeholder | menu)* > @item <!ELEMENT menuitem EMPTY > @item <!ELEMENT toolitem (menu?) > @item <!ELEMENT separator EMPTY > @item <!ELEMENT accelerator EMPTY > @item <!ATTLIST menubar name #IMPLIED @item action #IMPLIED > @item <!ATTLIST toolbar name #IMPLIED @item action #IMPLIED > @item <!ATTLIST popup name #IMPLIED @item action #IMPLIED > @item <!ATTLIST placeholder name #IMPLIED @item action #IMPLIED > @item <!ATTLIST separator name #IMPLIED @item action #IMPLIED @item expand (true | false) #IMPLIED > @item <!ATTLIST menu name #IMPLIED @item action #REQUIRED @item position (top | bot) #IMPLIED > @item <!ATTLIST menuitem name #IMPLIED @item action #REQUIRED @item position (top | bot) #IMPLIED > @item <!ATTLIST toolitem name #IMPLIED @item action #REQUIRED @item position (top | bot) #IMPLIED > @item <!ATTLIST accelerator name #IMPLIED @item action #REQUIRED >

@end itemize There are some additional restrictions beyond those specified in the DTD, e.g. every toolitem must have a toolbar in its ancestry and every menuitem must have a menubar or popup in its ancestry. Since a GMarkup parser is used to parse the UI description, it must not only be valid XML, but valid GMarkup.

If a name is not specified, it defaults to the action. If an action is not specified either, the element name is used. The name and action attributes must not contain '/' characters after parsing (since that would mess up path lookup) and must be usable as XML attributes when enclosed in doublequotes, thus they must not '"' characters or references to the " entity.

Here is an example: @example

<ui> <menubar> <menu name="FileMenu" action="FileMenuAction"> <menuitem name="New" action="New2Action" /> <placeholder name="FileMenuAdditions" /> </menu> <menu name="JustifyMenu" action="JustifyMenuAction"> <menuitem name="Left" action="justify-left"/> <menuitem name="Centre" action="justify-center"/> <menuitem name="Right" action="justify-right"/> <menuitem name="Fill" action="justify-fill"/> </menu> </menubar> <toolbar action="toolbar1"> <placeholder name="JustifyToolItems"> <separator/> <toolitem name="Left" action="justify-left"/> <toolitem name="Centre" action="justify-center"/> <toolitem name="Right" action="justify-right"/> <toolitem name="Fill" action="justify-fill"/> <separator/> </placeholder> </toolbar> </ui> @end example

The constructed widget hierarchy is very similar to the element tree of the XML, with the exception that placeholders are merged into their parents. The correspondence of XML elements to widgets should be almost obvious:

@itemize @bullet @item menubar : a Gtk_Menu_Bar @item toolbar : a Gtk_Toolbar @item popup : a toplevel Gtk_Menu @item menu : a Gtk_Menu attached to a menuitem @item menuitem : a Gtk_Menu_Item subclass, the exact type depends on the action @item toolitem : a Gtk_Tool_Item subclass, exact type depends on the action. This may contain a menu element only if the associated action specifies a Gtk_Menu_Tool_Button as proxy @item separator : a Gtk_Separator_Menu_Item or Gtk_Separator_Tool_Item @item accelerator : a keyboard accelerator

@end itemize The "position" attribute determines where a constructed widget is positioned wrt. to its siblings in the partially constructed tree. If it is "top", the widget is prepended, otherwise it is appended.

———— UI Merging – ————

The most remarkable feature of Gtk_UI_Manager is that it can overlay a set of menuitems and toolitems over another one, and demerge them later.

Merging is done based on the names of the XML elements. Each element is identified by a path which consists of the names of its anchestors, separated by slashes. For example, the menuitem named "Left" in the example above has the path /ui/menubar/JustifyMenu/Left and the toolitem with the same name has path /ui/toolbar1/JustifyToolItems/Left.

———— Accelerators – ————

Every action has an accelerator path. Accelerators are installed together with menuitem proxies, but they can also be explicitly added with <accelerator> elements in the UI definition. This makes it possible to have accelerators for actions even if they have no visible proxies.

———— Smart Separators – ————

The separators created by Gtk_UI_Manager are "smart", i.e. they do not show up in the UI unless they end up between two visible menu or tool items. Separators which are located at the very beginning or end of the menu or toolbar containing them, or multiple separators next to each other, are hidden. This is a useful feature, since the merging of UI elements from multiple sources can make it hard or impossible to determine in advance whether a separator will end up in such an unfortunate position.

For separators in toolbars, you can set expand="true" to turn them from a small, visible separator to an expanding, invisible one. Toolitems following an expanding separator are effectively right-aligned.

————- Empty Menus – ————-

Submenus pose similar problems to separators inconnection with merging. It is impossible to know in advance whether they will end up empty after merging. Gtk_UI_Manager offers two ways to treat empty submenus:

@itemize @bullet @item make them disappear by hiding the menu item they're attached to @item add an insensitive "Empty" item

@end itemize The behaviour is chosen based on the "hide_if_empty" property of the action to which the submenu is associated.

205.1 Signals

- **"actions_changed"**

  ```
  procedure Handler (UI : access Gtk_UI_Manager_Record'Class);
  ```

  This signal is emitted whenever the set of actions changes.

- **"add_widget"**

  ```
  procedure Handler
  (UI     : access Gtk_UI_Manager_Record'Class;
  Widget : access Gtk_Widget_Record'Class);
  ```

  The add_widget signal is emitted for each generated menubar and toolbar. It is not emitted for generated popup menus, which can be obtained by Get_Widget.

- **"connect_proxy"**

  ```
  procedure Handler
  (UI     : access Gtk_UI_Manager_Record'Class;
  Action : access Gtk_Action_Record'Class;
  Proxy  : access Gtk_Widget_Record'Class);
  ```

  This signal is emitted after connecting a proxy to an action in the group. This is intended for simple customizations for which a custom action class would be too clumsy, e.g. showing tooltips for menuitems in the statusbar.

- **"disconnect_proxy"**

  ```
  procedure Handler
  (UI     : access Gtk_UI_Manager_Record'Class;
  Action : access Gtk_Action_Record'Class;
  Proxy  : access Gtk_Widget_Record'Class);
  ```

  The disconnect_proxy signal is emitted after disconnecting a proxy from an action in the group.

- **"post_activate"**

  ```
  procedure Handler
  (UI     : access Gtk_UI_Manager_Record'Class;
  Action : access Gtk_Action_Record'Class);
  ```

  The post_activate signal is emitted just after the action is activated. This is intended for applications to get notification just after any action is activated.

- **"pre_activate"**

  ```
  procedure Handler
  (UI     : access Gtk_UI_Manager_Record'Class;
  Action : access Gtk_Action_Record'Class);
  ```

  The pre_activate signal is emitted just before the action is activated. This is intended for applications to get notification just before any action is activated.

205.2 Types

type Manager_Item_Type **is mod** 2 ** 16;

205.3 Subprograms

205.3.1 Creation

```
procedure Gtk_New
  (UI                : out   Gtk_UI_Manager);
function Get_Type             return GType;
```

Return the internal value associated with a Gtk_UI_Manager.

```
function New_Merge_Id
  (Self              : access Gtk_UI_Manager_Record)
   return Guint;
```

Returns an unused merge id, suitable for use with Add_UI.

```
procedure Ensure_Update
  (Self              : access Gtk_UI_Manager_Record);
```

Makes sure that all pending updates to the UI have been completed.
This may occasionally be necessary, since Gtk_UI_Manager updates the UI in an idle function. A typical example where this function is useful is to enforce that the menubar and toolbar have been added to the main window before showing it: Add (Window, Vbox); Connect (Merge, "add_widget", Add_Widget'Access, Vbox); Add_UI_From_File (Merge, "my-menus"); Add_UI_From_File (Merge, "my-toolbars"); Ensure_Update (Merge); Show (Window);

205.3.2 Merging contents

```
procedure Add_UI
  (Self              : access Gtk_UI_Manager_Record;
   Merge_Id          :      Guint;
   Path              :      String;
   Name              :      String;
   Action            :      String := "";
   Typ               :      Manager_Item_Type
                     := Manager_Auto;
   Top               :      Boolean := False);
procedure Remove_UI
  (Self              : access Gtk_UI_Manager_Record;
   Merge_Id          :      Guint);
```

Adds a UI element to the current contents of Self.

If Typ is Manager_Auto, GTK+ inserts a menuitem, toolitem or separator if such an element can be inserted at the place determined by Path. Otherwise Typ must indicate an element that can be inserted at the place determined by Path.

If Path points to a menuitem or toolitem, the new element will be inserted before or after this item, depending on Top.

Merge_Id: see New_Merge_Id. Action should be the empty string for a separator.

```
function Add_UI_From_File
  (Self              : access Gtk_UI_Manager_Record;
   Filename          :      String;
   Error             :      Glib.Error.GError_Access
                     := null)
   return Guint;
```

Parses a file containing a UI definition and merges it with the current
contents of Self. Return value: The merge id for the merged UI. The merge id can be used

to unmerge the UI with Remove_UI. If an error occurred, the return value is 0, and if Error was specified it is set to the error message.

```
function Add_UI_From_String
  (Self               : access Gtk_UI_Manager_Record;
   Buffer             :      String;
   Error              :      Glib.Error.GError_Access
                      := null)
   return Guint;
```

Parses a string containing a UI definition and merges it with the current contents of Self. An enclosing <ui> element is added if it is missing. Return value: The merge id for the merged UI. The merge id can be used to unmerge the UI with Remove_UI. If an error occurred, the return value is 0, and if Error was specified it is set to the error message.

```
procedure Insert_Action_Group
  (Self               : access Gtk_UI_Manager_Record;
   Action_Group       : access Gtk.Action_Group.Gtk_Action_Group_Record'Class;
   Pos                :      Gint);
procedure Remove_Action_Group
  (Self               : access Gtk_UI_Manager_Record;
   Action_Group       : access Gtk.Action_Group.Gtk_Action_Group_Record'Class);
```

Inserts an action group into the list of action groups associated with Self. Actions in earlier groups hide actions with the same name in later groups. with Self.

205.3.3 Querying contents

```
function Get_Accel_Group
  (Self               : access Gtk_UI_Manager_Record)
   return Gtk.Accel_Group.Gtk_Accel_Group;
```

Returns the Gtk_Accel_Group associated with Self.

```
function Get_Action
  (Self               : access Gtk_UI_Manager_Record;
   Path               :      String)
   return Gtk.Action.Gtk_Action;
```

Looks up an action by following a path. See Get_Widget for more information about paths. Returns the action whose proxy widget is found by following the path, or null if no widget was found.

```
function Get_Action_Groups
  (Self               : access Gtk_UI_Manager_Record)
   return Glib.Object.Object_Simple_List.Glist;
```

Returns the list of action groups associated with Self. The returned list should not be modified.

```
procedure Set_Add_Tearoffs
  (Self               : access Gtk_UI_Manager_Record;
   Add_Tearoffs       :      Boolean);
function Get_Add_Tearoffs
  (Self               : access Gtk_UI_Manager_Record)
   return Boolean;
```

Sets the "add_tearoffs" property, which controls whether menus generated by this Gtk_UI_Manager will have tearoff menu items. Note that this only affects regular menus. Generated popup menus never have tearoff menu items.

```
function Get_Toplevels
  (Self               : access Gtk_UI_Manager_Record;
   Types              :      Manager_Item_Type)
   return Gtk.Widget.Widget_SList.GSlist;
```

Obtains a list of all toplevel widgets of the requested types.
Types may contain Manager_Menubar, Manager_Toolbar or Manager_Popup. The returned list must be freed by the caller.

```
function Get_UI
  (Self               : access Gtk_UI_Manager_Record)
   return String;
```

Create a UI definition of the merged UI

```
function Get_Widget
  (Self               : access Gtk_UI_Manager_Record;
   Path               :      String)
   return Gtk.Widget.Gtk_Widget;
```

Looks up a widget by following a path.
The path consists of the names specified in the XML description of the UI. separated by '/'. Elements which don't have a name or action attribute in the XML (e.g. <popup>) can be addressed by their XML element name (e.g. "popup"). The root element ("/ui") can be omitted in the path.

Note that the widget found by following a path that ends in a <menu>; element is the menuitem to which the menu is attached, not the menu itself.

Also note that the widgets constructed by a ui manager are not tied to the lifecycle of the ui manager. If you add the widgets returned by this function to some container or explicitly ref them, they will survive the destruction of the ui manager.

205.4 Example

```
<ui>
   <menubar>
     <menu name="FileMenu" action="FileMenuAction">
       <menuitem name="New" action="New2Action" />
       <placeholder name="FileMenuAdditions" />
     </menu>
     <menu name="JustifyMenu" action="JustifyMenuAction">
       <menuitem name="Left" action="justify-left"/>
       <menuitem name="Centre" action="justify-center"/>
       <menuitem name="Right" action="justify-right"/>
       <menuitem name="Fill" action="justify-fill"/>
     </menu>
   </menubar>
   <toolbar action="toolbar1">
     <placeholder name="JustifyToolItems">
     <separator/>
     <toolitem name="Left" action="justify-left"/>
     <toolitem name="Centre" action="justify-center"/>
     <toolitem name="Right" action="justify-right"/>
     <toolitem name="Fill" action="justify-fill"/>
```

```
      <separator/>
    </placeholder>
  </toolbar>
</ui>
```

206 Package Gtk.Vbutton_Box

A Gtk_Vbutton_Box is a specific Gtk_Button_Box that organizes its children vertically. The beginning of the box (when you add children with Gtk.Box.Pack_Start) is on the top of the box. Its end (for Gtk.Box.Pack_End) is on the bottom.

206.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Box          (Package Gtk.Box)
            \___ Gtk_Button_Box (Package Gtk.Button_Box)
               \___ Gtk_Vbutton_Box (Package Gtk.Vbutton_Box)
```

206.2 Subprograms

```
procedure Gtk_New
  (Widget             : out   Gtk_Vbutton_Box);
```

Create a new vertical button box.

```
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Vbutton_Box.

207 Package Gtk.Viewport

This widget is an adapter: it can contain any child, and will make it scrollable. Its use is not necessary inside a Gtk_Scrolled_Window, which automatically uses a Gtk_Viewport when necessary.

207.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Viewport (Package Gtk.Viewport)
```

207.2 Signals

- **"set_scroll_adjustments"**

```
procedure Handler
(Viewport   : access Gtk_Viewport_Record'Class;
Hadj, Vadj : access Gtk_Adjustment_Record'Class);
```

  You should emit this signal to request a change of adjustments for the viewport. Seldom used, it is simpler to use Set_Vadjusment and Set_Hadjustment.

207.3 Subprograms

```
procedure Gtk_New
  (Viewport           : out   Gtk_Viewport;
   Hadjustment        :       Adjustment.Gtk_Adjustment
                       := null;
   Vadjustment        :       Adjustment.Gtk_Adjustment
                       := null);
function Get_Type               return Glib.GType;
```

Return the internal value associated with a Gtk_Viewport.

```
function Get_Bin_Window
  (Widget             : access Gtk_Viewport_Record)
   return Gdk.Gdk_Window;
```

Return the window associated with the viewport.
You should use this one rather than Gtk.Widget.Get_Window.

```
procedure Set_Hadjustment
  (Viewport           : access Gtk_Viewport_Record;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment);
function Get_Hadjustment
  (Viewport           : access Gtk_Viewport_Record)
   return Adjustment.Gtk_Adjustment;
```

Sets or gets the Gtk_Adjustment used for horizontal scrolling

```
procedure Set_Vadjustment
  (Viewport           : access Gtk_Viewport_Record;
   Adjustment         :       Gtk.Adjustment.Gtk_Adjustment);
function Get_Vadjustment
  (Viewport           : access Gtk_Viewport_Record)
   return Adjustment.Gtk_Adjustment;
```

Sets or gets the Gtk_Adjustment used for vertical scrolling

```
procedure Set_Shadow_Type
  (Viewport           : access Gtk_Viewport_Record;
   The_Type           :      Gtk_Shadow_Type);
function Get_Shadow_Type
  (Viewport           : access Gtk_Viewport_Record)
   return Gtk_Shadow_Type;
```

Sets or gets the visual rendering of the viewport

208 Package Gtk.Widget

This widget is the base of the tree for displayable objects. (A displayable object is one which takes up some amount of screen real estate). It provides a common base and interface which actual widgets must adhere to.

This package provides some services which might have been more appropriate in some other packages, but could not because of dependency circularities (there are for instance some functions relating to colors and colormaps). We have tried to reference these functions in the other packages as well.

208.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
```

208.2 Signals

- **"add_accelerator"**

- **"button_press_event"**

  ```
  function Handler (Widget : access Gtk_Widget_Record'Class;
  Event  : Gdk.Event.Gdk_Event_Button)
  return Boolean;
  ```

  A button was pressed while the pointer was inside the widget. To get this signal, some widgets by have to use the Set_Events subprogram first to get this event. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"button_release_event"**

  ```
  function Handler (Widget : access Gtk_Widget_Record'Class;
  Event  : Gdk.Event.Gdk_Event_Button)
  return Boolean;
  ```

  A button was released while the pointer was inside the widget. Note that in some cases (Gtk_Buttons for instance), another "clicked" signal could be emitted). This "button_release_event" should mainly be used for widgets that don't already have specific signals to cover that case (Gtk_Drawing_Area for instance).

  To get this signal, some widgets may have to use the Set_Events subprogram first to get this event.

  If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"child_notify"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class);
  ```

  This signal is emitted when the value of one of the child properties for the widget has been changed. If you are only interested in the changes for a specific property, you can also connect directly to "child_notify::<property>", for instance "child_notify:right_attach" for a child of Gtk.Menu.Gtk_Menu.

- **"client_event"**

- **"configure_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Configure)
return Boolean;
```

  Some configuration of the window has changed (it has been moved or resized). If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"delete_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event)
return Boolean;
```

  The user has clicked on the "close" button in the window's frame (the button that is automatically set by the window manager). If the handler returns False, the widget will be destroyed (and the window closed), but if the handler returns True, nothing will be done. This is a good way to prevent the user from closing your application's window if there should be some clean ups first (like saving the document).

- **"destroy_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event)
return Boolean;
```

  This signal is apparently never emitted by Gtk+. You might want to use "destroy" instead, which is documented in Gtk.Object.

- **"drag_begin"**

- **"drag_data_delete"**

- **"drag_data_get"**

- **"drag_data_received"**

- **"drag_drop"**

- **"drag_end"**

- **"drag_leave"**

- **"drag_motion"**

- **"draw"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class;
Area   : Gdk.Rectangle.Gdk_Rectangle);
```

  Emitted when a widget needs to be drawn. The default handler emits the "expose" event.

- **"draw_default"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

  Emitted when a widget needs to be drawn and it does not have the focus. This is never called if the widget can not have the focus (ie the "Can_Focus" flag is unset).

- **"draw_focus"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

  Emitted when a widget needs to be drawn and it has the focus. Some widgets might want to provide visual clues that they have the focus, like a black border. This is never called if the widget can not have the focus (ie the "Can_Focus" flag is unset).

- **"enter_notify_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Crossing)
return Boolean;
```

  The pointer has just entered the widget. If the "Can_Focus" flag is set, Widget will gain the focus, and the widget might be drawn differently. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event)
return Boolean;
```

  Some event was sent to the widget. This covers all the cases below, and acts as a general handler. This is called in addition to the relevant specific handler below. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"expose_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Expose)
return Boolean;
```

  The widget needs to be partly redrawn. The exact area to redraw is found in Event. For some widgets, you should rather connect to the "draw" signal. However, for instance for Gtk_Drawing_Area widgets, you have to use this, after setting the correct event mask with Set_Events. If the handler returns False, the event might be passed to the parent of widget (if no other handler of widget has returned True).

- **"focus_in_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Focus)
return Boolean;
```

  The widget has just gained the focus. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True). This event is only emitted if you called Add_Events with a Enter_Notify_Mask parameter

- **"focus_out_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Focus)
return Boolean;
```

The widget has just lost the focus. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True). This event is only emitted if you called Add_Events with a Leave_Notify_Mask parameter

- **"grab_focus"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

The widget has got the focus, ie will now get the keyboard events sent to a window. This is only called if the "Can_Focus" flag is set. The "Has_Focus" flag might not be set when this signal is emitted.

- **"hide"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

Emitted when a widget is to be hidden (see explanation for the Hide subprogram). Hides the widget from the screen, and if its parent is shown, the widget will not appear on the screen again.

- **"key_press_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Key)
return Boolean;
```

A key has been pressed while Widget had the focus. Note that some widgets like Gtk_Editable provide some higher-level signals to handle this. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"key_release_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Key)
return Boolean;
```

A key has been released while Widget had the focus. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"leave_notify_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Crossing)
return Boolean;
```

The pointer has just leaved the widget. If the "Can_Focus" flag is set, Widget will gain the focus, and the widget might be drawn differently. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"map"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

Emitted when a widget is mapped on the screen (the default handler simply emits the "show" signal).

- **"map_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event)
return Boolean;
```

The widget has just been mapped. This is different from the "map" signal, which is called *before* the widget is actually mapped. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"motion_notify_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Motion)
return Boolean;
```

The pointer has moved while remaining inside the widget. The Set_Events subprogram has to be called first to get this event.

If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"no_expose_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event)
return Boolean;
```

???

- **"parent_set"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class;
Previous_Parent : access Gtk_Widget_Record'Class);
```

A new parent has been set for the widget. The previous parent is given in arguments (if there was none, Gdk.Is_Created (Previous_Parent) returns False).

- **"property_notify_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Property)
return Boolean;
```

???

- **"proximity_in_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Proximity)
return Boolean;
```

Used for special input devices. See the description of Gdk.Event.Gdk_Event_Proximity. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"proximity_out_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Proximity)
return Boolean;
```

Used for special input devices. See the description of Gdk.Event.Gdk_Event_Proximity. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"query-tooltip"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Params : Glib.Values.GValues)
return Boolean;
```

Emitted when "has-tooltip" is TRUE and the "gtk-tooltip-timeout" has expired with the cursor hovering "above" widget; or emitted when widget got focus in keyboard mode.

Using the given coordinates, the signal handler should determine whether a tooltip should be shown for widget. If this is the case TRUE should be returned, FALSE otherwise. Note that if keyboard_mode is TRUE, the values of x and y are undefined and should not be used.

The signal handler is free to manipulate tooltip with the therefore destined function calls.

- **"realize"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

Emitted when a widget is realized. The default handler creates the Gdk window associated with the widget, and its ancestors.

- **"remove_accelerator"**

- **"selection_clear_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Selection)
return Boolean;
```

???

- **"selection_get"**

- **"selection_notify_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Selection)
return Boolean;
```

???

- **"selection_received"**

- **"selection_request_event"**

```
function Handler (Widget : access Gtk_Widget_Record'Class;
Event  : Gdk.Event.Gdk_Event_Selection)
return Boolean;
```

???

- **"show"**

```
procedure Handler (Widget : access Gtk_Widget_Record'Class);
```

Emitted when a widget is to be shown (see explanation for the Show subprogam). This schedules the widget to be displayed on the screen, and if this is a toplevel widget it actually appears on the screen and all its children that have been shown.

- **"size_allocate"**

```
procedure Handler (Widget     : access Gtk_Widget_Record'Class;
Allocation : Gtk_Allocation);
```

A size and position were assigned to the widget. This is called every time the size of the widget changes. The default handler takes care of resizing and moving the widget.

- **"size_request"**

```
procedure Handler (Widget      : access Gtk_Widget_Record'Class;
Requisition : access Gtk_Requisition);
```

Should return (in Requisition) the ideal size the widget would like to have. It is not sure this is the size that will be assigned to it, since it depends on the size of its parent).

- **"state_changed"**

  ```
  procedure Handler (Widget         : access Gtk_Widget_Record'Class;
  Previous_State : Gtk.Enums.Gtk_State_Type);
  ```

  The state of the widget has changed.

- **"style_set"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class);
  ```

  Previous_Style : Gtk.Style.Gtk_Style); The widget's style has been changed (this is not call when some settings in the style are changed, only when the style itself is completely changed with a call to Set_Style or Set_Default_Style).

- **"unmap"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class);
  ```

  Emitted when a widget needs to be unmapped on the screen (the default handler simply emits the "hide" signal).

- **"unmap_event"**

  ```
  function Handler (Widget : access Gtk_Widget_Record'Class;
  Event  : Gdk.Event.Gdk_Event)
  return Boolean;
  ```

  The widget has just been unmapped. This is different from the "unmap" signal, which is called *before* the widget is actually unmapped. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

- **"unrealize"**

  ```
  procedure Handler (Widget : access Gtk_Widget_Record'Class);
  ```

  Emitted when a widget is unrealized. The default handler destroys the Gdk windows of the widget and all its children.

- **"visibility_notify_event"**

  ```
  function Handler (Widget : access Gtk_Widget_Record'Class;
  Event  : Gdk.Event.Gdk_Event_Visibility)
  return Boolean;
  ```

  The visibility state of the widget has changed (partially visible, fully visible, ...). You might want to use the "expose" signal instead. If the handler returns False, the event might be pass to the parent of widget (if no other handler of widget has returned True).

208.3 Types

```
type Gtk_Allocation is record
        X      : Gint;
        Y      : Gint;
        Width  : Allocation_Int;
        Height : Allocation_Int;
        end record;
```

Gtk_Allocation indicates a size and position a widget was allocated. See the section in the user guide on how to create new widgets for more information. pragma Convention (C, Gtk_Allocation);

```
type Gtk_Allocation_Access is access all Gtk_Allocation;
```

```
type Gtk_Requisition is record
          Width  : Gint;
          Height : Gint;
          end record;
```

Gtk_Requisition is the desired amount of screen real-estate a widget requests to the server. Its real allocated size might be different. See the section in the GtkAda user guide on how to create new widgets in Ada, and the examples/base_widget directory for an example on how to use this. pragma Convention (C, Gtk_Requisition);

```
type Gtk_Requisition_Access is access all Gtk_Requisition;
```

```
type Widget_Type is new Gtk_Widget_Record with private;
```

208.4 Subprograms

```
function Get_Type              return Glib.GType;
```

Return the internal type associated with a Gtk_Widget.

```
function Requisition_Get_Type  return Glib.GType;
```

Return the internal type for a Gtk_Requisition

208.4.1 Widgets' life cycle

```
procedure Destroy_Cb
  (Widget             : access Gtk_Widget_Record'Class);
```

This function should be used as a callback to destroy a widget.
All it does is call Destroy on its argument, but its profile is compatible with the handlers found in Gtk.Handlers.

```
procedure Unparent
  (Widget             : access Gtk_Widget_Record'Class);
```

This function is only for use in widget implementations.
Should be called by implementations of the remove method on Gtk_Container, to dissociate a child from the container. Users should call Remove instead. This function might be dangereous: it correctly updates widget to reflect that it no longer belongs to its parent, however the parent keeps an internal pointer to the widget, which will result in a storage_error if you try to further access it.

```
procedure Show
  (Widget             : access Gtk_Widget_Record);
```

Schedule the widget to be displayed on the screen when its parent is
also shown (emits the "show" signal). If its ancestors are already mapped to the screen, then the widget is immediately displayed through a call to Map below.

```
procedure Show_Now
  (Widget             : access Gtk_Widget_Record);
```

Show the widget.
If it is an unmapped toplevel widget, wait for it to be mapped. This creates a recursive main_loop.

```
procedure Hide
  (Widget             : access Gtk_Widget_Record);
```

Hide the widget from the screen (emits the "hide" signal).
If Widget was visible, it is immediately hidden. If one of its ancestor is later shown on the screen, Widget won't appear. Note that on some window managers, including CDE, hiding an iconified window will not do anything. You should in addition call Gdk.Window.Withdraw to make sure the window is properly hidden.

```
procedure Show_All
  (Widget             : access Gtk_Widget_Record);
```

Show Widget and all its children recursively.
See also Set_Child_Visible below

```
procedure Hide_All
  (Widget             : access Gtk_Widget_Record);
```

Hide Widget and all its children.
Note that if you simply want to delete Widget from the screen, you can simply call the Hide subprogram on it. This procedure Hide_All should only be used if you want to unschedule a widget to be displayed later, not to remove an actual widget from the screen. See also Set_Child_Visible below.

```
procedure Set_No_Show_All
  (Widget             : access Gtk_Widget_Record;
   No_Show_All        :      Boolean);

function Get_No_Show_All
  (Widget             : access Gtk_Widget_Record)
   return Boolean;
```

Sets the "no_show_all" property, which determines whether calls to
Show_All() and Hide_All() will affect this widget. This is mostly for use in constructing widget hierarchies with externally controlled visibility.

```
procedure Map
  (Widget             : access Gtk_Widget_Record);
```

Map a widget to the screen.
A window is created for it on the screen (through a call to Realize) and Widget is then drawn on the screen (if its ancestors are also mapped). This function is recursive and will also map all the children of Widget.

It is recommended to use the higher-level Show instead.

```
procedure Unmap
  (Widget             : access Gtk_Widget_Record);
```

Unmap a widget from the screen.
This results in the widget being hidden, but not destroyed. It can be shown again any time through a call to Map (provided its ancestors are also mapped).

It is recommended to use the higher-level Hide instead.

```
procedure Realize
  (Widget             : access Gtk_Widget_Record);
```

Create a window for Widget and its ancestors (emit the "realize" signal)
This does not mean that the widget will appear on the screen, but resources such as colormaps, etc. become available. Some routines require that the widget is realized before any call. You must set the Event_Mask before calling this routine if you want to change it from its default value.

```
procedure Unrealize
  (Widget             : access Gtk_Widget_Record);
```

Hide the widget from the screen and deletes the associated window.
This does not destroy the widget itself, only its server-side resources.

```
procedure Set_Realize
  (Widget             : access Gtk_Widget_Record'Class);
```

Set the realize handler at the low level.
This is needed to replace the default realize in new widgets.

```
function Hide_On_Delete
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Hide widget and return True.
This function is intended to be used as a callback.

```
procedure Set_Child_Visible
  (Widget             : access Gtk_Widget_Record;
   Is_Visible         :      Boolean);
function Get_Child_Visible
  (Widget             : access Gtk_Widget_Record)
   return Boolean;
```

Sets whether Widget should be mapped along with its parent when its
parent is mapped and Widget has been shown with Show.

"mapped" indicates the moment the window is actually shown on the screen. Show and Hide indicate your intention to show Widget on the scree or not, but if the parent of Widget is itself not shown at that time, the two commands Show and Hide have no immediate effect, and just set a flag to save your intent. Set_Child_Visible indicates that the widget shouldn't be part of the recursive processing done by Show_All and Hide_All on the parent. You have decided once and for all what the behavior should be, and you don't want it to be changed by future calls to Show_All and Hide_All.

The child visibility can be set for widget before it is added to a container with Set_Parent, to avoid mapping children unnecessary before immediately unmapping them. However it will be reset to its default state of True when the widget is removed from a container.

Note that changing the child visibility of a widget does not queue a resize on the widget. Most of the time, the size of a widget is computed from all visible children, whether or not they are mapped. If this is not the case, the container can queue a resize itself.

This function is only useful for container implementations and should generally not be called by an application.

```
function Has_Screen
  (Widget             : access Gtk_Widget_Record)
   return Boolean;
```

Checks whether there is a Gdk_Screen is associated with
this widget. All toplevel widgets have an associated screen, and all widgets added into a hierarchy with a toplevel window at the top.

208.4.2 Drawing a widget

```
procedure Queue_Draw
  (Widget             : access Gtk_Widget_Record);
```

Add a drawing request to the event queue for the whole widget.
This is more efficient than calling Draw directly, since GtkAda groups drawing requests as much as possible to speed up the drawing process. The actual drawing will take place as soon as GtkAda is not busy processing other events, but before idle events.

```
procedure Queue_Draw_Area
  (Widget             : access Gtk_Widget_Record;
   X                  :      Gint;
   Y                  :      Gint;
   Width              :      Gint;
   Height             :      Gint);
```

Add a drawing request to the event queue for part of the widget.
This is more efficient that calling Draw directly (see Queue_Draw).

```
procedure Queue_Resize
  (Widget             : access Gtk_Widget_Record);
```

Queue drawing requests after a resizing of the widget.
This clears the widget, and its parent if any, so that everything is correctly redrawn. You should not have to call this function directly. For a Gtk_Window, check the procedure Gtk.Window.Resize instead.

```
procedure Queue_Resize_No_Redraw
  (Widget             : access Gtk_Widget_Record);
```

This function works like Queue_Resize(), except that the
widget is not invalidated (ie will not be redrawn)

```
function Create_Pango_Context
  (Widget             : access Gtk_Widget_Record)
   return Pango.Context.Pango_Context;
```

Create a new Pango_Context with the appropriate colormap, font
description, and base direction for drawing text for this widget. See also Get_Pango_Context. The returned context must be freed by the caller.

```
function Create_Pango_Layout
  (Widget             : access Gtk_Widget_Record;
   Text               :      UTF8_String := "")
   return Pango.Layout.Pango_Layout;
```

Return a new pango_layout that displays Text. This fully handles
internationalization, and should be the preferred way to display text, rather than Gdk.Drawable.Draw_Text Text must be a valid Utf8 text, see Glib.Convert.

208.4.3 Size and position

```
procedure Size_Request
  (Widget             : access Gtk_Widget_Record;
   Requisition        : in out Gtk_Requisition);
```

Emit a "size_request" event for the widget

```
procedure Set_Size_Request
  (Widget             : access Gtk_Widget_Record;
   Width, Height      :      Gint := -1);
procedure Get_Size_Request
  (Widget             : access Gtk_Widget_Record;
   Width, Height      : out   Gint);
```

Sets the minimum size of a widget; that is, the widget's size request
will be Width by Height. You can use this function to force a widget to be either larger or smaller than it normally would be.

In most cases, Set_Default_Size is a better choice for toplevel windows than this function; setting the default size will still allow users to shrink the window. Setting the size request will force them to leave the window at least as large as the size request. When dealing with window sizes, Gtk.Windo.Set_Geometry_Hints can be a useful function as well.

Note the inherent danger of setting any fixed size - themes, translations into other languages, different fonts, and user action can all change the appropriate size for a given widget. So, it's basically impossible to hardcode a size that will always be correct.

The size request of a widget is the smallest size a widget can accept while still functioning well and drawing itself correctly. However in some strange cases a widget may be allocated less than its requested size, and in many cases a widget may be allocated more space than it requested.

If the size request in a given direction is -1 (unset), then the "natural" size request of the widget will be used instead.

Widgets can't actually be allocated a size less than 1 by 1, but you can pass 0,0 to this function to mean "as small as possible."

```
procedure Size_Allocate
  (Widget             : access Gtk_Widget_Record;
   Allocation         :      Gtk_Allocation);
```

Emit a "size_allocate" event for the widget.
Allocation'size is first constrained to a range between 1x1 and 32767x32767. A clear and draw request is also queued if required.

```
function Get_Child_Requisition
  (Widget             : access Gtk_Widget_Record)
   return Gtk_Requisition;
```

Return the size requests by the widget.
This is the ideal size for the widget, not necessarily its actual size. See the user guide's section on how to create new widgets for more information on the size requisition and allocation.

```
function Get_Allocation_Width
  (Widget             : access Gtk_Widget_Record)
   return Allocation_Int;
```

Return the current width of the widget.

```
function Get_Allocation_Height
  (Widget             : access Gtk_Widget_Record)
   return Allocation_Int;
```

Return the current height of the widget.

```
function Get_Allocation_X
  (Widget             : access Gtk_Widget_Record)
```

```
   return Gint;
```

Return the current position of the widget, relative to its parent.

```
function Get_Allocation_Y
  (Widget             : access Gtk_Widget_Record)
   return Gint;
```

Return the current position of the widget, relative to its parent.

```
procedure Set_Redraw_On_Allocate
  (Widget             : access Gtk_Widget_Record;
   Redraw_On_Allocate  :      Boolean);
```

Sets whether the entire widget is queued for drawing when its size allocation changes. By default, this setting is %TRUE and the entire widget is redrawn on every size change. If your widget leaves the upper left unchanged when made bigger, turning this setting on will improve performance. Note that for %NO_WINDOW widgets setting this flag to %FALSE turns off all allocation on resizing: the widget will not even redraw if its position changes; this is to allow containers that don't draw anything to avoid excess invalidations. If you set this flag on %NO_WINDOW widget that *does* draw on Get_Window (Widget), you are responsible for invalidating both the old and new allocation of the widget when the widget is moved and responsible for invalidating regions newly when the widget increases size.

208.4.4 Accelerators

```
procedure Add_Accelerator
  (Widget             : access Gtk_Widget_Record;
   Accel_Signal       :      Glib.Signal_Name;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group;
   Accel_Key          :      Gdk.Types.Gdk_Key_Type;
   Accel_Mods         :      Gdk.Types.Gdk_Modifier_Type;
   Accel_Flags        :      Gtk.Accel_Group.Gtk_Accel_Flags);
```

Add a new accelerator for the widget.

The signal Accel_Signal will be sent to Widget when the matching key is pressed and the widget has the focus. Consider using Gtk.Accel_Map.Add_Entry instead, which is compatible with interactive change of accelerators by the user.

```
procedure Remove_Accelerator
  (Widget             : access Gtk_Widget_Record;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group;
   Accel_Key          :      Gdk.Types.Gdk_Key_Type;
   Accel_Mods         :      Gdk.Types.Gdk_Modifier_Type);
```

Remove an accelerator for the widget.

```
function Can_Activate_Accel
  (Widget             : access Gtk_Widget_Record;
   Signal_Id          :      Gulong)
   return Boolean;
```

Determines whether an accelerator that activates the signal identified by Signal_Id can currently be activated. This is done by emitting the GtkWidget::can-activate-accel signal on Widget; if the signal isn't overridden by handler or in a derived widget, then the default check is that the widget must be sensitive, and the widget and all its ancestors mapped. Signal_Id comes from the value returned by Gtk.Handlers.Connect

```
procedure Set_Accel_Path
  (Widget             : access Gtk_Widget_Record;
```

```
   Accel_Path          :      UTF8_String;
   Group               :      Gtk.Accel_Group.Gtk_Accel_Group);
```

Set the path that will be used to reference the widget in calls to the subprograms in Gtk.Accel_Map. This means, for instance, that the widget is fully setup for interactive modification of the shortcuts by the user, should he choose to activate this possibility in his themes (see gtk-accel_map.ads for more information).

```
function List_Mnemonic_Labels
  (Widget             : access Gtk_Widget_Record)
   return Widget_List.Glist;
```

Returns a newly allocated list of the widgets, normally labels, for which this widget is a the target of a mnemonic (see for example, gtk.label.set_mnemonic_widget). The widgets in the list are not individually referenced. If you want to iterate through the list and perform actions involving callbacks that might destroy the widgets, you must call Ref first, and then unref all the widgets afterwards. The caller must free the returned list.

```
procedure Add_Mnemonic_Label
  (Widget             : access Gtk_Widget_Record;
   Label              : access Gtk_Widget_Record'Class);
```

Adds a widget to the list of mnemonic labels for this widget. (See List_Mnemonic_Labels). Note the list of mnemonic labels for the widget is cleared when the widget is destroyed, so the caller must make sure to update its internal state at this point as well, by using a connection to the ::destroy signal or a weak notifier.

```
procedure Remove_Mnemonic_Label
  (Widget             : access Gtk_Widget_Record;
   Label              : access Gtk_Widget_Record'Class);
```

Removes a widget from the list of mnemonic labels for this widget. The widget must have previously been added to the list with Add_Mnemonic_Label.

```
function Mnemonic_Activate
  (Widget             : access Gtk_Widget_Record;
   Group_Cycling      :      Boolean)
   return Boolean;
```

Emits the signal "mnemonic_activate".
In general (depending on what is connected to this signal), this results in calling the "activate" signal on the widget, as if a mnemonic had been used (when Group_Cycling if False), or to grab the focus on the widget when Group_Cycling is True)

208.4.5 Events and signals

```
function Event
  (Widget             : access Gtk_Widget_Record'Class;
   Event              :      Gdk.Event.Gdk_Event)
   return Boolean;
```

Emit a signal on the widget.
The exact signal depends on the event type (i.e. if the type is Gdk_Button_Press, then a "button_press" signal is emitted).

```
procedure Send_Expose
  (Widget             : access Gtk_Widget_Record;
   Event              :      Gdk.Event.Gdk_Event_Expose);
```

Emit an expose event signals on a widget.
This function is not normally used directly. The only time it is used is when propa-

gating an expose event to a child No_Window widget, and that is normally done using Gtk.Container.Propagate_Expose.

If you want to force an area of a window to be redrawn, use Gdk.Window.Invalidate_Rect or Gdk.Window.Invalidate_Region. To cause the redraw to be done immediately, follow that call with a call to Gdk.Window.Process_Updates.

```
procedure Activate
  (Widget             : access Gtk_Widget_Record);
```

Emit an activate signal on the widget.
The exact signal emitted depends on the widget type (i.e. for a Gtk_Button this emits a "clicked" signal, for a Gtk_Editable this emits the "activate" signal, ...).

```
procedure Grab_Focus
  (Widget             : access Gtk_Widget_Record);
```

Emit the "grab_focus" signal for the widget.
This is sent when the widget gets the focus. Its visual aspect might change. The "Can_Focus" flag must have been set first. See also Gtk.Widget.Child_Focus, which should be used instead when writting new widgets in Ada

```
function Is_Focus
  (Widget             : access Gtk_Widget_Record)
   return Boolean;
```

Determines if the widget is the focus widget within its
toplevel. (This does not mean that the HAS_FOCUS flag is necessarily set; HAS_FOCUS will only be set if the toplevel widget additionally has the global input focus)

```
function Child_Focus
  (Child              : access Gtk_Widget_Record'Class;
   Direction          :      Gtk.Enums.Gtk_Direction_Type
                      := Gtk.Enums.Dir_Tab_Forward)
   return Boolean;
```

Used by custom widget implementations to indicate the focus child.
If you're writing an app, you'd use Grab_Focus to move the focus to a particular widget, and Gtk.Container.Set_Focus_Chain to change the focus tab order. So you may want to investigate those functions instead.

Child_Focus is called by containers as the user moves around the window using keyboard shortcuts. Direction indicates what kind of motion is taking place (up, down, left, right, tab forward, tab backward). Child_Focus invokes the "focus" signal on Child; widgets override the default handler for this signal in order to implement appropriate focus behavior.

The "focus" default handler for a widget should return True if moving in Direction left the focus on a focusable location inside that widget, and False if moving in Direction moved the focus outside the widget. If returning True, widgets normally call Grab_Focus to place the focus accordingly; if returning False, they don't modify the current focus location.

This function replaces Gtk.Container.Focus from GTK+ 1.2. It was necessary to check that the child was visible, sensitive, and focusable before calling Gtk.Container.Focus. Child_Focus returns False if the widget is not currently in a focusable state, so there's no need for those checks.

Return value: True if focus ended up inside Child

```
procedure Set_Events
  (Widget             : access Gtk_Widget_Record;
   Events             :      Gdk.Event.Gdk_Event_Mask);
```

```
function Get_Events
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Event.Gdk_Event_Mask;
```

Sets or gets the event mask for the widget.
Widget should not have been realized before, or nothing is done. This is the only way you can explicitly get mouse or keyboards events on widgets that do not automatically get them, as for instance in a Gtk_Drawing_Area.

```
procedure Add_Events
  (Widget             : access Gtk_Widget_Record;
   Events             :      Gdk.Event.Gdk_Event_Mask);
```

Add some events to the current event mask of the widget.

```
procedure Set_Extension_Events
  (Widget             : access Gtk_Widget_Record;
   Mode               :      Gdk.Types.Gdk_Extension_Mode);

function Get_Extension_Events
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Types.Gdk_Extension_Mode;
```

Set the extension event mask for the widget.
This is used to activate some special input modes for other devices than keyboard and mouse.

```
function Default_Motion_Notify_Event
  (Widget             : access Gtk_Widget_Record'Class;
   Event              :      Gdk.Event.Gdk_Event)
   return Gint;
```

Access to the standard default callback for motion events:
This is mainly used for rulers in Gtk.Ruler (See the example in testgtk, with create_rulers.adb)

```
function Has_Default_Motion_Notify_Handler
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Return True if Widget has a default handler for motion_notify events.
Note that the function Default_Motion_Notify_Event should not be called if this one returns False, since it would create a segmentation fault.

208.4.6 Colors and colormaps

```
procedure Set_Colormap
  (Widget             : access Gtk_Widget_Record;
   Cmap               :      Gdk.Color.Gdk_Colormap);

function Get_Colormap
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Color.Gdk_Colormap;
```

Modify the colormap of the widget.
The widget must not have been realized when you set the colormap. The colormap is generally the same one for all widget, but might be different if for instance Gtk_Drawing_Area needs to display some different colors on a screen that only has a limited amount of colors.

```
function Get_Visual
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Visual.Gdk_Visual;
```

Get the visual used for the widget.
I.e. the structure that indicates the depth of the widget (number of bits per pixel), and some information used internally by GtkAda to handle colors and colormaps.

```
procedure Push_Colormap
  (Cmap               :      Gdk.Color.Gdk_Colormap);

procedure Pop_Colormap;
```

Modify temporarily the default colormap set for newly created widgets.
You should use this in pair with Pop_Colormap (Push the new value, create the widget, and pop the value).

```
procedure Set_Default_Colormap
  (Cmap               :      Gdk.Color.Gdk_Colormap);

function Get_Default_Colormap  return Gdk.Color.Gdk_Colormap;
```

Modify permanently the default colormap used when a widget is created.
If you only want to modify this colormap temporarily for a few widgets, you should consider using Push_Colormap and Pop_Colormap instead. See also Gdk.Screen.Get_Default_Colormap for a multihead-aware version

```
function Get_Default_Visual    return Gdk.Visual.Gdk_Visual;
```

Return the default visual used when a new widget is created.

208.4.7 Styles

```
procedure Set_Style
  (Widget             : access Gtk_Widget_Record;
   Style              :      Gtk.Style.Gtk_Style);

function Get_Style
  (Widget             : access Gtk_Widget_Record)
   return Gtk.Style.Gtk_Style;
```

Set or get the style for a given widget.
See also Gtk.Rc.Modify_Style

```
function Get_Default_Style     return Gtk.Style.Gtk_Style;
```

Get the default global style.

```
procedure Ensure_Style
  (Widget             : access Gtk_Widget_Record);
```

Make sure that the widget has a style associated to it.
Either the default one as set by Set_Default_Style above or one set by the user with Set_Style.

```
procedure Restore_Default_Style
  (Widget             : access Gtk_Widget_Record);
```

Restore the default style that was set for the widget.
The default style is the first one that was set either by a call to Set_Style or Set_Default_Style.

```
procedure Reset_Rc_Styles
  (Widget             : access Gtk_Widget_Record);
```

Restore the Rc style recursively for widget and its children.

```
function Get_Pango_Context
  (Widget             : access Gtk_Widget_Record)
   return Pango.Context.Pango_Context;
```

Get a Pango_Context with the appropriate colormap, font description and
base direction for this widget. Unlike the context returned by Create_Pango_Context, this context is owned by the widget (it can be used as long as widget exists), and will be updated to match any changes to the widget's attributes.

If you create and keep a Pango_Layout using this context, you must deal with changes to the context by calling Pango_Layout.Context_Changed on the layout in response to the ::style_set and ::direction_set signals for the widget.

```
procedure Modify_Fg
  (Widget             : access Gtk_Widget_Record;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
```

Sets the foreground color for a widget in a particular state. All
other style values are left untouched.

```
procedure Modify_Bg
  (Widget             : access Gtk_Widget_Record;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
```

Sets the background color for a widget in a particular state. All
other style values are left untouched. This procedure has no effect when Widget has no physical window associated to it (for instance a Gtk_Label). In such cases, you must put widget inside a Gtk_Event_Box, and set the background color of the box itself.

```
procedure Modify_Text
  (Widget             : access Gtk_Widget_Record;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
```

Sets the text color for a widget in a particular state. All other
style values are left untouched. The text color is the foreground color used along with the base color (see Modify_Base) for widgets such as Gtk_Entry and Gtk_Text_View.

Note that this will not work with a Gtk_Button. Modify_Fg should be called on the button's label in order to set the color of its label. For example, assuming a simple button with a label attached to it:

Modify_Fg (Get_Child (My_Button), My_State, My_New_Color);

```
procedure Modify_Base
  (Widget             : access Gtk_Widget_Record;
   State_Type         :      Enums.Gtk_State_Type;
   Color              :      Gdk.Color.Gdk_Color);
```

Sets the base color for a widget in a particular state.
All other style values are left untouched. The base color is the background color used along with the text color (see Modify_Text) for widgets such as Gtk_Entry and Gtk_Text_View.

```
procedure Modify_Font
  (Widget             : access Gtk_Widget_Record;
   Desc               :      Pango.Font.Pango_Font_Description);
```

Modify the font used for the widget.
Desc must be freed by the caller to avoid memory leaks

```
procedure Set_Default_Direction
  (Dir                :      Gtk.Enums.Gtk_Text_Direction);

function Get_Default_Direction return Gtk.Enums.Gtk_Text_Direction;
```

Obtains the current default reading direction. See
Set_Default_Direction().

```
procedure Set_Direction
  (Widget             : access Gtk_Widget_Record;
   Dir                :      Gtk.Enums.Gtk_Text_Direction);
function Get_Direction
  (Widget             : access Gtk_Widget_Record)
   return Gtk.Enums.Gtk_Text_Direction;
```

Sets the reading direction on a particular widget. This direction
controls the primary direction for widgets containing text, and also the direction in which the children of a container are packed. The ability to set the direction is present in order so that correct localization into languages with right-to-left reading directions can be done. Generally, applications will let the default reading direction present, except for containers where the containers are arranged in an order that is explicitely visual rather than logical (such as buttons for text justification).

If the direction is set to TEXT_DIR_NONE, then the value set by Set_Default_Direction will be used.

208.4.8 Widgets' tree

```
procedure Set_Name
  (Widget             : access Gtk_Widget_Record;
   Name               :      UTF8_String);
```

Set the name for the widget.
This name is used purely internally to identify the widget, and does not give any visual clue.

```
function Get_Name
  (Widget             : access Gtk_Widget_Record)
   return UTF8_String;
```

Return the name of the widget if it was set by Set_Name.
Return the name of its class otherwise.

```
function Path
  (Widget             : access Gtk_Widget_Record)
   return String;
function Path_Reversed
  (Widget             : access Gtk_Widget_Record)
   return String;
```

Obtains the full path to Widget. The path is simply the name of a
widget and all its parents in the container hierarchy, separated by periods. The name of a widget comes from Get_Name. Paths are used to apply styles to a widget in gtkrc configuration files. Widget names are the type of the widget by default (e.g. "GtkButton") or can be set to an application-specific value with Set_Name. By setting the name of a widget, you allow users or theme authors to apply styles to that specific widget in their gtkrc file. Path_Reverse fills in the path in reverse order, starting with widget's name instead of starting with the name of the outermost ancestor.

```
function Class_Path
  (Widget             : access Gtk_Widget_Record)
   return String;
function Class_Path_Reversed
  (Widget             : access Gtk_Widget_Record)
```

```
   return String;
```

Same as Path(), but always uses the name of a widget's type,
never uses a custom name set with Set_Name.

```
function Get_Ancestor
  (Widget             : access Gtk_Widget_Record;
   Ancestor_Type      :      Gtk_Type)
   return Gtk_Widget;
```

Return the closest ancestor of Widget which is of type Ancestor_Type.
Return null if there is none.

```
procedure Set_Parent
  (Widget             : access Gtk_Widget_Record;
   Parent             : access Gtk_Widget_Record'Class);
function Get_Parent
  (Widget             : access Gtk_Widget_Record)
   return Gtk_Widget;
```

Modify the parent for the widget.
This is not the recommended way to do this, you should use Gtk.Container.Add or Gtk.Box.Pack_Start instead.

```
procedure Set_Parent_Window
  (Widget             : access Gtk_Widget_Record;
   Window             :      Gdk.Window.Gdk_Window);
function Get_Parent_Window
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Window.Gdk_Window;
```

Set the parent window for the actual Gdk_Window of the widget. This sets
up required internal fields, and should be used only when you implement your own container, as opposed to using one of the standard containers.

```
function Get_Toplevel
  (Widget             : access Gtk_Widget_Record)
   return Gtk_Widget;
```

This function returns the topmost widget in the container hierarchy
Widget is a part of. If Widget has no parent widgets, it will be returned as the topmost widget.

Note the difference in behavior vs. Get_Ancestor: Get_Ancestor (Widget, GTK_TYPE_WINDOW) would return null if Widget wasn't inside a toplevel window, and if the window was inside a Gtk_Window-derived widget which was in turn inside the toplevel Gtk_Window. While the second case may seem unlikely, it actually happens when a Gtk_Plug is embedded inside a Gtk_Socket within the same application.

To reliably find the toplevel Gtk_Window, use Get_Toplevel and check if the "toplevel" flag is set on the result:

Toplevel := Get_Toplevel (Widget); if Top_Level_Is_Set (Toplevel) then [Perform some action on Toplevel.] end if;

```
function Is_Ancestor
  (Widget             : access Gtk_Widget_Record;
   Ancestor           : access Gtk_Widget_Record'Class)
   return Boolean;
```

Return True if Ancestor is in the ancestor tree for Widget.
I.e. if Widget is contained within Ancestor.

```
procedure Reparent
  (Widget             : access Gtk_Widget_Record;
   New_Parent         : access Gtk_Widget_Record'Class);
```

Change the parent of the widget dynamically.
If both the new parent and the widget are shown, then the widget is visually redrawn in its new parent.

```
procedure Translate_Coordinates
  (Src_Widget         :      Gtk_Widget;
   Dest_Widget        :      Gtk_Widget;
   Src_X              :      Gint;
   Src_Y              :      Gint;
   Dest_X             : out  Gint;
   Dest_Y             : out  Gint;
   Result             : out  Boolean);
```

Translate coordinates relative to Src_Widget's allocation to coordinates relative to Dest_Widget's allocations. In order to perform this operation, both widgets must be realized, and must share a common toplevel.

Result is set to False if either widget was not realized, or there was no common ancestor. In this case, nothing is stored in Dest_X and Dest_Y. Otherwise True.

```
function Get_Root_Window
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Window.Gdk_Window;
```

Get the root window where this widget is located. This function can only be called after the widget has been added to a widget hierarchy.

The root window is useful for such purposes as creating a popup Gdk_Window associated with the window. In general, you should only create display specific resources when a widget has been realized, and you should free those resources when the widget is unrealized.

```
procedure Set_Composite_Name
  (Widget             : access Gtk_Widget_Record;
   Name               :      String);
function Get_Composite_Name
  (Widget             : access Gtk_Widget_Record)
   return String;
```

Sets or gets a widgets composite name. The widget must be a composite child of its parent; see Push_Composite_Child.

```
procedure Push_Composite_Child;
procedure Pop_Composite_Child;
```

Makes all newly-created widgets as composite children until the corresponding Pop_Composite_Child call.

A composite child is a child that's an implementation detail of the container it's inside and should not be visible to people using the container. Composite children aren't treated differently by GTK (but see gtk.container.foreach() vs. gtk.container.forall()), but e.g. GUI builders might want to treat them in a different way.

Here is a simple example: Push_Composite_Child; Gtk_New (Scrolled_Window.Hscrollbar, Hadjustment); Set_Composite_Name (Scrolled_Window.Hscrollbar, "hscrollbar"); Pop_Composite_Child; Set_Parent (Scrolled_Window.Hscrollbar, Scrolled_Window); Ref (Scrolled_Window.Hscrollbar);

208.4.9 Misc functions

```
procedure Set_Scroll_Adjustments
  (Widget             : access Gtk_Widget_Record;
   Hadj               :      Gtk.Adjustment.Gtk_Adjustment;
   Vadj               :      Gtk.Adjustment.Gtk_Adjustment);
```

Emit the "set_scroll_adjustments" signal.
The exact signal emitted depends on the widget type (see Gtk.Object.Initialize_Class_Record). The handler creates the adjustments if null is passed as argument, and makes sure both adjustments are in the correct range.

```
function Intersect
  (Widget             : access Gtk_Widget_Record;
   Area               :      Gdk.Rectangle.Gdk_Rectangle;
   Intersection       : access Gdk.Rectangle.Gdk_Rectangle)
   return Boolean;
```

Return True if the widget intersects the screen area Area.
The intersection area is returned in Intersection.

```
function Region_Intersect
  (Widget             : access Gtk_Widget_Record;
   Region             :      Gdk.Region.Gdk_Region)
   return Gdk.Region.Gdk_Region;
```

Region must be in the same coordinate system as the widget's allocation,
ie relative to the widget's window, or to the parent's window for No_Window widgets. Returns a newly allocated region. The coordinats are in the same system as described above. Computes the intersection of a Widget's area and Region, returning the intersection. The result may be empty, use gdk.region.empty to check.

```
procedure Grab_Default
  (Widget             : access Gtk_Widget_Record);
```

The widget becomes the default widget for its parent window or dialog.
All keyboard events will be sent to it if no other widget has the focus. Note that the "Can_Default" flag must have been set first on WIDGET.

```
procedure Set_State
  (Widget             : access Gtk_Widget_Record;
   State              :      Enums.Gtk_State_Type);
function Get_State
  (Widget             : access Gtk_Widget_Record)
   return Enums.Gtk_State_Type;
```

Modify the state of the widget.
This modifies its visual aspect, and thus should be used only if you change its behavior at the same time, so as not to confuse the user.

```
procedure Set_Sensitive
  (Widget             : access Gtk_Widget_Record;
   Sensitive          :      Boolean := True);
```

Modify the sensitivity of the widget.
An insensitive widget is generally grayed out, and can not be activated. For instance, an insensitive menu item is grayed, and can never be selected.

```
procedure Set_App_Paintable
  (Widget             : access Gtk_Widget_Record;
   App_Paintable      :      Boolean);
```

Modify the "App_Paintable" flag for the widget.

```
procedure Set_Double_Buffered
  (Widget             : access Gtk_Widget_Record;
   Double_Buffered    :      Boolean := True);
```

Modify the "Double_Buffered" flag for the widget.

```
procedure Get_Pointer
  (Widget             : access Gtk_Widget_Record;
   X                  : out   Gint;
   Y                  : out   Gint);
```

Return the coordinates of the pointer (i.e. mouse) relative to Widget.

```
procedure Set_Window
  (Widget             : access Gtk_Widget_Record;
   Window             :      Gdk.Window.Gdk_Window);

function Get_Window
  (Widget             : access Gtk_Widget_Record)
   return Gdk.Window.Gdk_Window;
```

Set the Gdk window associated with the widget.
You can use this window if you need to draw directly on the widget using the functions found in the Gdk hierarchy. These functions are rarely used except when you implement your own own widget types. Predefined widgets takes care of that automatically.

```
procedure Shape_Combine_Mask
  (Widget             : access Gtk_Widget_Record;
   Shape_Mask         :      Gdk.Bitmap.Gdk_Bitmap;
   Offset_X           :      Gint;
   Offset_Y           :      Gint);
```

Modify the shape of the window that contains the widget.
This allows for transparent windows, and requires the Xext library to be available on your system. If this library is not available, your program will still work. See the manual page for XShapeCombineMask(3x) for more information.

```
procedure Reset_Shapes
  (Widget             : access Gtk_Widget_Record);
```

Recursively resets the shape on this widget and its descendants.

```
function Render_Icon
  (Widget             : access Gtk_Widget_Record;
   Stock_Id           :      String;
   Size               :      Gtk.Enums.Gtk_Icon_Size;
   Detail             :      UTF8_String := "")
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

A convenience function that uses the theme engine for Widget, to lookup
a Stock_Id (see Gtk.Stock) and render it to a pixbuf (see Gdk.Pixbuf). Detail should be a string that identifies the widget or code doing the rendering, so that the theme engine can special-case rendering for that widget or code. It can be left to the empty stirng to get the default behavior.

Null is returned if Stock_Id wasn't known.

208.4.10 Tooltips

```
procedure Set_Tooltip_Text
  (Widget             : access Gtk_Widget_Record;
   Text               :      UTF8_String);
```

Sets text as the contents of the tooltip. This function will take care of setting GtkWidget::has-tooltip to TRUE and of the default handler for the GtkWidget::query-tooltip signal.

See also the GtkWidget:tooltip-text property and Gtk_Tooltips.Set_Text.

```
procedure Set_Tooltip_Markup
  (Widget             : access Gtk_Widget_Record;
   Text               :      UTF8_String);
```

Sets markup as the contents of the tooltip, which is marked up with the Pango text markup language.

This function will take care of setting GtkWidget:has-tooltip to TRUE and of the default handler for the GtkWidget::query-tooltip signal.

See also the GtkWidget::tooltip-markup property and Gtk_Tooltips.Set_Markup.

```
procedure Set_Tooltip_Window
  (Widget             : access Gtk_Widget_Record;
   Custom_Window      : access Gtk_Widget_Record'Class);
```

Custom_Window : access Gtk.Window.Gtk_Window_Record'Class);

Replaces the default, usually yellow, window used for displaying tooltips with custom_window. GTK+ will take care of showing and hiding Custom_Window at the right moment, to behave likewise as the default tooltip window. If Custom_Window is NULL, the default tooltip window will be used.

```
function Get_Tooltip_Window
  (Widget             : access Gtk_Widget_Record)
   return Gtk_Widget;
```

return Gtk.Window.Gtk_Window;

Returns the GtkWindow of the current tooltip. This can be the GtkWindow created by default, or the custom tooltip window set using Gtk.Widget.Set_Tooltip_Window.

208.4.11 Creating new widgets

Although the core subprogram for creating new widgets is@* Glib.Gobjects.Initialize_Class_Record, it is often useful to override some internal pointers to functions. The functions below are not needed unless you are writting your own widgets, and should be reserved for advanced customization of the standard widgets.

```
procedure Set_Scroll_Adjustments_Signal
  (Widget             :      Glib.Object.GObject_Class;
   Signal             :      String);
```

Modify the signal to be sent when the adjustments are modified.
This is only useful when you are rewritting your own widget that can be embedded directly in a Gtk_Scrolled_Window, without any Gtk_Viewport.

Signal is the name of the signal that will be emitted when Widget is put inside a Gtk_Scrolled_Window.

Note that the handlers for this signal must take two arguments in addition to the widget (the horizontal and vertical adjustments to be used). See Gtk.Scrolled_Window and Gtk.Widget.Set_Scroll_Adjustment for more information on this signal.

```
procedure Set_Default_Size_Allocate_Handler
  (Klass              :      Glib.Object.GObject_Class;
   Handler            :      Size_Allocate_Handler);
```

Override the default size_allocate handler for this class. This handler is automatically called in several cases (when a widget is dynamically resized for instance), not through a signal. Thus, if you need to override the default behavior provided by one of the standard containers, you can not simply use Gtk.Handlers.Emit_Stop_By_Name, and you must override the default handler. Note also that this handler is automatically inherited by children of this class.

```
procedure Set_Allocation
  (Widget             : access Gtk_Widget_Record'Class;
   Alloc              :      Gtk_Allocation);
```

Modifies directly the internal field of Widget to register the new allocation. Beware that the only use of this method is inside a callback set by Set_Default_Size_Allocate_Handler. If you simply want to resize or reposition a widget, use Size_Allocate instead.

```
function Default_Expose_Event_Handler
  (Klass              :      GObject_Class)
   return Expose_Event_Handler;
```

Return the default expose event handler for the widget class Klass. The typical use for this function is when you are writting your own container class. You should then, from your own handler for expose_event, call the one of the parent class, so that all the children are automatically redrawn.

208.4.12 Flags

Some additional flags are defined for all the visual objects (widgets).@* They are defined in addition to the ones defined in Gtk.Object. These flags are important in that they define exactly the different states a widget can be in.

@itemize @bullet @item "Toplevel": Set if the widget is a toplevel widget, ie has no parent. This is mostly true for windows and dialogs.

@item "No_Window": Set if the widget does not have an associated X11 window, ie can not receive events directly. For instance, a Gtk_Toolbar does not have an associated window. These objects are more lightweight, but require more work from GtkAda. This flag is only set if the widget will never have a window, even after it is realized.

@item "Realized": Set if the widget has been realized, ie its associated X11 window has been created (providing the widget excepts a window, see the No_Window flag

@item "Mapped": Set if the widget is visible on the screen. This is only possible if the Visible flag is also set.

@item "Visible": Set if the widget will be displayed on the screen when mapped (see the functions Show and Hide in this package).

@item "Sensitive": Set if the widget is listening to events. See the function Set_Sensitive in this package. An insensitive widget will generally have a different visual aspect to clue that it is unavailable (for instance an insensitive item menu will be grayed)

@item "Parent_Sensitive": Set if the parent is sensitive. A widget is sensitive only if both the Sensitive and Parent_Sensitive are set.

@item "Can_Focus": Set if the widget can have the focus, ie get keyboard events. Most widgets can not have the focus.

@item "Has_Focus": Set if the widget currently has the focus. See the function Grab_Focus in this package. See also the subprogram Gtk.Widget.Is_Focus

@item "Can_Default": Set if the widget can be the default widget in a window, ie the one that will get the keyboard events by default. For instance, the default button in a dialog is the one that gets clicked on when the user pressed Enter anywhere in the dialog.

@item "Has_Default": Set if the widget is currently the default widget. See the function Grab_Default in this package.

@item "Has_Grab": Set if the widget currently grabs all mouse and keyboard events in the application, even if it does not have the focus. There can be only such widget per application at any given time.

@item "Rc_Style": Set if the widget's style is either the default style, or in a customization file. This is unset if the style has been modified by the user.

@item "Composite_Child": This indicates whether the widget is composed of other widgets

@item "No_Reparent": This flags is never used in gtk+.

@item "App_Paintable": For some containers (including Gtk_Window and Gtk_Layout), this is unset when the container itself has some special drawing routines. It indicates whether the application will paint directly on the widget.

@item "Receives_Default": Set when the widget receives the default at the time it receives the focus. This is how the default button in a dialog is automatically changed when you press another button. @end itemize

```
function Toplevel_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Toplevel flag is set.

```
function No_Window_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the No_Window flag is set.

```
function Realized_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Realized flag is set.

```
function Mapped_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Mapped flag is set.

```
function Visible_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Visible flag is set.

```
function Drawable_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

True if the widget is both visible and mapped.
In other words, if it does appear on the screen.

```
function Is_Sensitive
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the widget is Sensitive.

```
function Can_Focus_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Can_Focus flag is set.

```
function Has_Focus_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Has_Focus flag is set.

```
function Has_Default_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Has_Default flag is set.

```
function Has_Grab_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Has_Grab flag is set.

```
function Rc_Style_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Rc_Style flag is set.

```
function Double_Buffered_Is_Set
  (Widget             : access Gtk_Widget_Record'Class)
   return Boolean;
```

Test whether the Double_Buffered flag is set.

208.4.13 GValue support

```
function Get_Requisition
  (Value              :     Glib.Values.GValue)
   return Gtk_Requisition_Access;
```

Convert a value into a Gtk_Requisition_Access.

```
function Get_Allocation
  (Value              :     Glib.Values.GValue)
   return Gtk_Allocation_Access;
```

Convert a value into a Gtk_Allocation_Access.

208.4.14 Properties

The following properties are defined for this widget. See@* Glib.Properties for more information on properties.

```
procedure Child_Notify
  (Widget             : access Gtk_Widget_Record;
   Child_Property     :      String);
```

Emits a "child-notify" signal for the child property on Widget.
This signal indicates the the value of the child property has changed on the parent, and thus that Widget should refresh itself if needed.

Child_Property is the name of a child property installed on Widget's parent. You should use Glib.Propert_Name to get the name from the property declaration in each of the GtkAda packages

```
procedure Freeze_Child_Notify
  (Widget             : access Gtk_Widget_Record);
```

Stops emission of "child-notify" signals on Widget. The signals are
queued until Thaw_Child_Notify() is called on Wwidget.

```
procedure Thaw_Child_Notify
  (Widget             : access Gtk_Widget_Record);
```

Reverts the effect of a previous call to Freeze_Child_Notify.
This causes all queued "child-notify" signals on Widget to be emitted.

```
procedure Class_Install_Style_Property
  (Klass              :      Glib.Object.GObject_Class;
   Pspec              :      Glib.Param_Spec);
```

Installs a style property on a widget class. The parser for the
style property is determined by the value type of Pspec. A style property configures the look-and-feel of a widget class. They are generally modified by the current gtk+ theme, although users can also modify them in their own configuration file.

```
function Class_List_Style_Properties
  (Klass              :      Glib.Object.GObject_Class)
   return Glib.Param_Spec_Array;
```

Returns all style properties of a widget class.

```
function Class_Find_Style_Property
  (Klass              :      Glib.Object.GObject_Class;
   Property_Name      :      String)
   return Glib.Param_Spec;
```

Finds a style property of a widget class by name.
Klass must be a descendent of Gtk_Widget. You should use Glib.Property_Name to get the name from the property declaration in each of the GtkAda packages

```
procedure Style_Get_Property
  (Widget             : access Gtk_Widget_Record;
   Property_Name      :      String;
   Value              : out   Glib.Values.GValue);
```

Gets the value of a style property of Widget.
You should use Glib.Property_Name to get the name from the property declaration in each of the GtkAda packages

209 Package Gtk.Window

This widget implements a top level window. It is used as the base class for dialogs, ...

A window has both a default widget (to which events are sent if no other widget has been selected and has the focus), and a focus widget (which gets the events and overrides the default widget).

You can set many hints on the window (its minimum and maximum size, its decoration, etc.) but these are only hints to the window manager, which might not respect them.

A useful hint, respected by most window managers, can be used to force some secondary windows to stay on top of the main window on the screen (for instance, so that a smaller window can not be hidden by a bigger one). See the function Set_Transient_For below.

A window can also be modal, i.e. grab all the mouse and keyboard events in the application while it is displayed.

209.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Bin         (Package Gtk.Bin)
            \___ Gtk_Window   (Package Gtk.Window)
```

209.2 Signals

- **"activate_default"**

  ```
  procedure Handler (Window : access Gtk_Window_Record'Class);
  ```

  Same as Activate_Default, but can be bound to a key binding

- **"activate_focus"**

  ```
  procedure Handler (Window : access Gtk_Window_Record'Class);
  ```

  You should emit this signal to request that the currently focused widget receives the "activate" signal. This is the same as calling Activate_Focus, but can be bound to a key binding

- **"frame_event"**

  ```
  function Handler
  (Window : access Gtk_Window_Record'Class;
  Event  : Gdk.Event.Gdk_Event) return Boolean;
  ```

  If this function is called on a window before it is realized or showed it will have a "frame" window around widget-window. Called when the "frame" window set around a window receives events. This is mainly used by the linux-fb port to implement managed windows, but it could concievably be used by X-programs that want to do their own window decorations.

- **"keys_changed"**

  ```
  procedure Handler (Window : access Gtk_Window_Record'Class);
  ```

  Emitted when the key accelerators or mnemonics are changed for the window.

- **"move_focus"**

```
procedure Handler
(Window    : access Gtk_Window_Record'Class;
Direction : Gtk_Direction_Type);
```

Emitted when a new child gains the focus

- **"set_focus"**

```
procedure Handler (Window : access Gtk_Window_Record'Class;
Widget : access Gtk_Widget_Record'Class);
```

Called when the widget that has the focus has changed. This widget gets all keyboard events that happen in the window. You should not block the emission of this signal, since most of the work is done in the default handler.

209.3 Subprograms

```
procedure Gtk_New
  (Window             : out   Gtk_Window;
   The_Type           :       Gtk.Enums.Gtk_Window_Type
                       := Gtk.Enums.Window_Toplevel);
```

Create a new window.
The_Type specifies the type of the window, and can be either a top level window, a dialog or a popup window. You will most often only need to use Window_Toplevel, the other types are mostly used internally by gtk+. A Popup window is used to display a temporary information window. It has no borders nor resizing handles.

```
function Get_Type                return Glib.GType;
```

Return the internal value associated with a Gtk_Window.

```
procedure Set_Title
  (Window             : access Gtk_Window_Record;
   Title              :       UTF8_String);

function Get_Title
  (Window             : access Gtk_Window_Record)
   return UTF8_String;
```

Change the title of the window, as it appears in the title bar.
Note that on some systems you might not be able to change it.

```
procedure Set_Wmclass
  (Window             : access Gtk_Window_Record;
   Wmclass_Name       :       String;
   Wmclass_Class      :       String);
```

Don't use this function. It sets the X Window System "class" and "name" hints for a window. According to the ICCCM, you should always set these to the same value for all windows in an application, and GTK sets them to that value by default, so calling this function is sort of pointless. However, you may want to call Set_Role on each window in your application, for the benefit of the session manager. Setting the role allows the window manager to restore window positions when loading a saved session.

```
procedure Set_Role
  (Window             : access Gtk_Window_Record;
   Role               :       String);

function Get_Role
  (Window             : access Gtk_Window_Record)
   return String;
```

In combination with the window title, the window role allows a
window manager to identify "the same" window when an application is restarted. So for example you might set the "toolbox" role on your app's toolbox window, so that when the user restarts their session, the window manager can put the toolbox back in the same place. If a window already has a unique title, you don't need to set the role, since the WM can use the title to identify the window when restoring the session. Role: unique identifier for the window to be used when restoring a session

```
function Activate_Focus
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Call Gtk.Widget.Activate on the widget that currently has the focus in
the window, ie sends an "activate" signal to that widget. Note that this signal does not really exists and is mapped to some widget-specific signal. Return True if the widget could be activated, False otherwise. The Focus widget is set through a signal "set_focus".

```
function Activate_Default
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Activate the default widget in the window.
In other words, send an "activate" signal to that widget. Note that this signal is a virtual one and is mapped to some widget specific signal. Return False is the widget could not be activated or if there was no default widget. You can set the default widget with the following calls:

Gtk.Widget.Set_Flags (Widget, Can_Default); Gtk.Widget.Grab_Default (Widget);

```
procedure Set_Transient_For
  (Window             : access Gtk_Window_Record;
   Parent             : access Gtk_Window_Record'Class);
function Get_Transient_For
  (Window             : access Gtk_Window_Record)
   return Gtk_Window;
```

Specify that Window is a transient window.
A transient window is a temporary window, like a popup menu or a dialog box). Parent is the toplevel window of the application to which Window belongs. A window that has set this can expect less decoration from the window manager (for instance no title bar and no borders). (see XSetTransientForHint(3) on Unix systems)

The main usage of this function is to force Window to be on top of Parent on the screen at all times. Most window managers respect this hint, even if this is not mandatory.

```
procedure Set_Type_Hint
  (Window             : access Gtk_Window_Record;
   Hint               :      Gdk.Window.Gdk_Window_Type_Hint);
function Get_Type_Hint
  (Window             : access Gtk_Window_Record)
   return Gdk.Window.Gdk_Window_Type_Hint;
```

allow the window manager to decorate and handle the window in a way
which is suitable to the function of the window in your application. This function should be called before the window becomes visible.

```
procedure Set_Keep_Above
  (Window             : access Gtk_Window_Record;
   Setting            :      Boolean);
```

```
procedure Set_Keep_Below
  (Window             : access Gtk_Window_Record;
   Setting            :      Boolean);
```

Asks to keep Window above, so that it stays on top. Note that you
shouldn't assume the window is definitely above afterward, because other entities (e.g. the user or window managers) could not keep it above, and not all window managers support keeping windows above. But normally the window will end kept above. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be kept above when it appears onscreen initially.

You can track the above state via the "window_state_event" signal on Window.

Note that, according to the "Extended Window Manager Hints" specification, the above state is mainly meant for user preferences and should not be used by applications e.g. for drawing attention to their dialogs.

```
procedure Set_Auto_Startup_Notification
  (Setting            :      Boolean);
```

By default, after showing the first Window for each screen, GTK+ calls
gdk_notify_startup_complete(). Call this function to disable the automatic startup notification. You might do this if your first window is a splash screen, and you want to delay notification until after your real main window has been shown, for example.

In that example, you would disable startup notification temporarily, show your splash screen, then re-enable it so that showing the main window would automatically result in notification.

Notification is used by the desktop environment to show the user that your application is still loading.

```
procedure Set_Destroy_With_Parent
  (Window             : access Gtk_Window_Record;
   Setting            :      Boolean := True);
function Get_Destroy_With_Parent
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Set whether destroying the transient parent of Window will also destroy
Window itself. This is useful for dialogs that shouldn't persist beyond the lifetime of the main window they're associated with, for example.

```
procedure Set_Geometry_Hints
  (Window             : access Gtk_Window_Record;
   Geometry_Widget    :      Gtk.Widget.Gtk_Widget;
   Geometry           :      Gdk.Window.Gdk_Geometry;
   Geom_Mask          :      Gdk.Window.Gdk_Window_Hints);
```

Specify some geometry hints for the window.
This includes its minimal and maximal sizes, ... These attributes are specified in Geometry. Geom_Mask indicates which of the fields in Geometry are set. Geometry_Widget can be null (and thus is not an access parameter). It adds some extra size to Geometry based on the actual size of Geometry_Widget (the extra amount is Window'Size - Geometry_Widget'Size)

Geometry.Base_* indicates the size that is used by the window manager to report the size: for instance, if Base_Width = 600 and actual width is 200, the window manager will indicate a width of -400.

If your window manager respects the hints (and its doesn't have to), then the user will never be able to resize the window to a size not in Geometry.Min_* .. Geometry.Max_*.

Geometry.*_Inc specifies by which amount the size will be multiplied. For instance, if Width_Inc = 50 and the size reported by the Window Manager is 2x3, then the actual width of the window is 100. Your window's size will always be a multiple of the *_Inc values.

Geometry.*_Aspect specifies the aspect ratio for the window. The window will always be resized so that the ratio between its width and its height remains in the range Min_Aspect .. Max_Aspect.

```
procedure Set_Decorated
  (Window              : access Gtk_Window_Record;
   Setting             :      Boolean := True);

function Get_Decorated
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

By default, windows are decorated with a title bar, resize
controls, etc. Some window managers allow GtkAda to disable these decorations, creating a borderless window. If you set the decorated property to False using this function, GtkAda will do its best to convince the window manager not to decorate the window.

```
procedure Set_Modal
  (Window              : access Gtk_Window_Record;
   Modal               :      Boolean := True);

function Get_Modal
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

Define the window as being Modal.
It will grab the input from the keyboard and the mouse while it is displayed and will release it when it is hidden. The grab is only in effect for the windows that belong to the same application, and will not affect other applications running on the same screen. In cunjunction with Gtk.Main.Main, this is the easiest way to show a dialog to which the user has to answer before the application can continue.

```
procedure Set_Skip_Pager_Hint
  (Window              : access Gtk_Window_Record;
   Setting             :      Boolean);

function Get_Skip_Taskbar_Hint
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

Windows may set a hint asking the desktop environment not to display
the window in the pager. This function sets this hint. (A "pager" is any desktop navigation tool such as a workspace switcher that displays a thumbnail representation of the windows on the screen).

```
procedure Set_Skip_Taskbar_Hint
  (Window              : access Gtk_Window_Record;
   Setting             :      Boolean);

function Get_Skip_Pager_Hint
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

Windows may set a hint asking the desktop environment not to display
the window in the task bar. This function sets this hint.

```
procedure Set_Urgency_Hint
  (Window              : access Gtk_Window_Record;
   Setting             :      Boolean);
function Get_Urgency_Hint
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

Windows may set a hint asking the desktop environment to draw the users attention to the window. This function sets this hint.

```
function List_Toplevels        return Gtk.Widget.Widget_List.Glist;
```

Return a list of all existing toplevel windows.
The widgets in the list are not individually referenced. If you want to iterate through the list and perform actions involving callbacks that might destroy the widgets, you must "ref"erence all the widgets in the list first and then unref all the widgets afterwards. The list itself must be freed by the caller

```
procedure Present
  (Window              : access Gtk_Window_Record);
```

Present a window to the user.
This may mean raising the window in the stacking order, deiconifying it, moving it to the current desktop, and/or giving it the keyboard focus, possibly dependent on the user's platform, window manager, and preferences.

If Window is hidden, this function calls Gtk.Widget.Show as well.

This function should be used when the user tries to open a window that's already open. Say for example the preferences dialog is currently open, and the user chooses Preferences from the menu a second time; use Present to move the already-open dialog where the user can see it.

If you are calling this function in response to a user interaction, it is preferable to use Present_With_Time.

```
procedure Present_With_Time
  (Window              : access Gtk_Window_Record;
   Timestamp           :      Guint32);
```

Present a window to the user in response to a user interaction.
Timestamp is the timestamp of the user interaction (typically a button or key press event) which triggered this call.

```
procedure Stick
  (Window              : access Gtk_Window_Record);
```

Ask to stick Window, which means that it will appear on all user desktops. Note that you shouldn't assume the window is definitely stuck afterward, because other entities (e.g. the user or window manager) could unstick it again, and some window managers do not support sticking windows. But normally the window will end up stuck.

It's permitted to call this function before showing a window.

You can track stickiness via the "window_state_event" signal on Gtk_Widget.

```
procedure Unstick
  (Window              : access Gtk_Window_Record);
```

Ask to unstick Window, which means that it will appear on only one of the user's desktops. Note that you shouldn't assume the window is definitely unstuck

afterward, because other entities (e.g. the user or window manager) could stick it again. But normally the window will end up stuck.

You can track stickiness via the "window_state_event" signal on Gtk_Widget.

209.3.1 Position

```
procedure Move
  (Window             : access Gtk_Window_Record;
   X, Y               :      Gint);
```

Asks the window manager to move Window to the given position. Window managers are free to ignore this; most window managers ignore requests for initial window positions (instead using a user-defined placement algorithm) and honor requests after the window has already been shown.

Note: the position is the position of the gravity-determined reference point for the window. The gravity determines two things: first, the location of the reference point in root window coordinates; and second, which point on the window is positioned at the reference point.

By default the gravity is GRAVITY_NORTH_WEST, so the reference point is simply the (x, y) supplied to Move. The top-left corner of the window decorations (aka window frame or border) will be placed at (x, y). Therefore, to position a window at the top left of the screen, you want to use the default gravity (which is GRAVITY_NORTH_WEST) and move the window to 0,0.

To position a window at the bottom right corner of the screen, you would set GRAVITY_SOUTH_EAST, which means that the reference point is at x + the window width and y + the window height, and the bottom-right corner of the window border will be placed at that reference point. So, to place a window in the bottom right corner you would first set gravity to south east, then write: Move (Window, Gdk_Screen_Width - Window_Width, Gdk_Screen_Height - Window_Height);

```
procedure Set_Position
  (Window             : access Gtk_Window_Record;
   Position           :      Gtk.Enums.Gtk_Window_Position);
```

Specify how the position of the window should be computed.
If Position is Win_Pos_Center_Always or Win_Pos_Center, then the window is centered on the screen. In the first case, it is also recentered when the window is resized with Gtk.Widget.Set_Usize (ie except on user action). If Position is Win_Pos_Mouse, then the window is positioned so that it centered around the mouse. If Position is Win_Pos_None, no calculation is done. If Gtk.Widget.Set_Uposition has been called, it is respected. This is the default case.

```
procedure Get_Position
  (Window             : access Gtk_Window_Record;
   Root_X, Root_Y     : out   Gint);
```

This function returns the position you need to pass to gtk.window.move to keep Window in its current position. This means that the meaning of the returned value varies with window gravity. See Gtk.Window.Move for more details.

If you haven't changed the window gravity, its gravity will be GRAVITY_NORTH_WEST. This means that Get_Position gets the position of the top-left corner of the window manager frame for the window. gtk.window.move sets the position of this same top-left corner.

Get_Position is not 100% reliable because the X Window System does not specify a way to obtain the geometry of the decorations placed on a window by the window manager. Thus GTK+ is using a "best guess" that works with most window managers.

Moreover, nearly all window managers are historically broken with respect to their handling of window gravity. So moving a window to its current position as returned by Get_Position tends to result in moving the window slightly. Window managers are slowly getting better over time.

If a window has gravity GRAVITY_STATIC the window manager frame is not relevant, and thus Get_Position will always produce accurate results. However you can't use static gravity to do things like place a window in a corner of the screen, because static gravity ignores the window manager decorations.

If you are saving and restoring your application's window positions, you should know that it's impossible for applications to do this without getting it somewhat wrong because applications do not have sufficient knowledge of window manager state. The Correct Mechanism is to support the session management protocol (see the "GnomeClient" object in the GNOME libraries for example) and allow the window manager to save your window sizes and positions.

```
procedure Begin_Move_Drag
  (Window              : access Gtk_Window_Record;
   Button              :      Gint;
   Root_X              :      Gint;
   Root_Y              :      Gint;
   Timestamp           :      Guint32);
```

Starts moving a window. This function is used if an application has
window movement grips. When GDK can support it, the window movement will be done using the standard mechanism for the window manager or windowing system. Otherwise, GDK will try to emulate window movement, potentially not all that well, depending on the windowing system. (Root_X, Root_Y): Position where the user clicked to initiate the drag, in root window coordinates. Timestamp is the timestamp of the event that initiated the drag

```
function Parse_Geometry
  (Window              : access Gtk_Window_Record;
   Geometry            :      String)
   return Boolean;
```

Parses a standard X Window System geometry string - see the manual page
for X (type 'man X') for details on this. Parse_Geometry does work on all GTK+ ports including Win32 but is primarily intended for an X environment.

If either a size or a position can be extracted from the geometry string, Parse_Geometry returns True and calls Set_Default_Size and/or Move to resize/move the window.

If Parse_Geometry returns True, it will also set the HINT_USER_POS and/or HINT_USER_SIZE hints indicating to the window manager that the size/position of the window was user-specified. This causes most window managers to honor the geometry.

Note that for Parse_Geometry to work as expected, it has to be called when the window has its "final" size, i.e. after calling Show_All on the contents and Set_Geometry_Hints on the window.

209.3.2 Sizes

```
procedure Set_Resizable
  (Window              : access Gtk_Window_Record;
   Resizable           :      Boolean := True);

function Get_Resizable
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

Sets or gets whether the user can resize a window.
Windows are user resizable by default.

```
procedure Set_Gravity
  (Window              : access Gtk_Window_Record;
   Gravity             :      Gdk.Window.Gdk_Gravity);

function Get_Gravity
  (Window              : access Gtk_Window_Record)
   return Gdk.Window.Gdk_Gravity;
```

Window gravity defines the "reference point" to be used when
positioning or resizing a window. Calls to Gtk.Widget.Set_UPosition will position a different point on the window depending on the window gravity. When the window changes size the reference point determined by the window's gravity will stay in a fixed location.

See Gdk_Gravity for full details. To briefly summarize, Gravity_North_West means that the reference point is the northwest (top left) corner of the window frame. Gravity_South_East would be the bottom right corner of the frame, and so on. If you want to position the window contents, rather than the window manager's frame, Gravity_Static moves the reference point to the northwest corner of the Gtk_Window itself.

The default window gravity is Gravity_North_West.

```
procedure Set_Has_Frame
  (Window              : access Gtk_Window_Record);

function Get_Has_Frame
  (Window              : access Gtk_Window_Record)
   return Boolean;
```

If this function is called on a window before it is realized
or showed it will have a "frame" window around widget-window. Using the signal frame_event you can receive all events targeted at the frame.

This function is used by the linux-fb port to implement managed windows, but it could concievably be used by X-programs that want to do their own window decorations.

```
procedure Set_Frame_Dimensions
  (Window              : access Gtk_Window_Record;
   Left, Top, Right, Bottom  :      Gint);

procedure Get_Frame_Dimensions
  (Window              : access Gtk_Window_Record;
   Left, Top, Right, Bottom  : out   Gint);
```

Change the size of the frame border.
This has only an effect for windows with frames (see Set_Has_Frame).

```
procedure Fullscreen
  (Window              : access Gtk_Window_Record);

procedure Unfullscreen
  (Window              : access Gtk_Window_Record);
```

Ask to place Window in fullscreen state.
You shouldn't assume the window is definitely full screen afterward, because other entities (user or window manager) could unfullscreen it again and not all window managers honor requests to fullscreen windows. You can track the fullscreen state via the "window_state_event" signal.

```
procedure Iconify
  (Window             : access Gtk_Window_Record);
```

Ask to iconify Window.
Note that you shouldn't assume the window is definitely iconified afterward, because other entities (e.g. the user or window manager) could deiconify it again, or there may not be a window manager in which case iconification isn't possible, etc. But normally the window will end up iconified. Just don't write code that crashes if not.

It's permitted to call this function before showing a window, in which case the window will be iconified before it ever appears onscreen.

You can track iconification via the "window_state_event" signal on Gtk_Widget.

```
procedure Deiconify
  (Window             : access Gtk_Window_Record);
```

Ask to deiconify Window.
Note that you shouldn't assume the window is definitely deiconified afterward, because other entities (e.g. the user or window manager) could iconify it again before your code which assumes deiconification gets to run.

You can track iconification via the "window_state_event" signal on Gtk_Widget.

```
procedure Maximize
  (Window             : access Gtk_Window_Record);
```

Ask to maximize Window, so that it becomes full-screen.
Note that you shouldn't assume the window is definitely maximized afterward, because other entities (e.g. the user or window manager) could unmaximize it again, and not all window managers support maximization. But normally the window will end up maximized.

It's permitted to call this function before showing a window, in which case the window will be maximized when it appears onscreen initially.

You can track maximization via the "window_state_event" signal on Gtk_Widget.

```
procedure Unmaximize
  (Window             : access Gtk_Window_Record);
```

Ask to unmaximize Window.
Note that you shouldn't assume the window is definitely unmaximized afterward, because other entities (e.g. the user or window manager) could maximize it again, and not all window managers honor requests to unmaximize. But normally the window will end up unmaximized.

You can track maximization via the "window_state_event" signal on Gtk_Widget.

```
procedure Set_Default_Size
  (Window             : access Gtk_Window_Record;
   Width              :      Gint;
   Height             :      Gint);
procedure Get_Default_Size
  (Window             : access Gtk_Window_Record;
   Width              : out   Gint;
```

```
   Height              : out   Gint);
```

Sets the default size of a window. If the window's "natural" size (its size request) is larger than the default, the default will be ignored. More generally, if the default size does not obey the geometry hints for the window (Set_Geometry_Hints can be used to set these explicitly), the default size will be clamped to the nearest permitted size.

Unlike Gtk.Widget.Set_Size_Request, which sets a size request for a widget and thus would keep users from shrinking the window, this function only sets the initial size, just as if the user had resized the window themselves. Users can still shrink the window again as they normally would. Setting a default size of -1 means to use the "natural" default size (the size request of the window).

For more control over a window's initial size and how resizing works, investigate Set_Geometry_Hints.

For some uses, Resize is a more appropriate function. Resize changes the current size of the window, rather than the size to be used on initial display. Resize always affects the window itself, not the geometry widget.

The default size of a window only affects the first time a window is shown; if a window is hidden and re-shown, it will remember the size it had prior to hiding, rather than using the default size.

Windows can't actually be 0x0 in size, they must be at least 1x1, but passing 0 for Width and Height is OK, resulting in a 1x1 default size.

This has no effect on Popup windows (set in call to Gtk_New).

```
procedure Resize
  (Window              : access Gtk_Window_Record;
   Width, Height       :      Gint);
```

Resize the window as if the user had done so, obeying geometry constraints. The default geometry constraint is that windows may not be smaller than their size request; to override this constraint, call Gtk.Widget.Set_Size_Request to set the window's request to a smaller value.

If Resize is called before showing a window for the – first time, it overrides any default size set with – Set_Default_Size.

Windows may not be resized smaller than 1 by 1 pixels. However, as a special case, if both Width and Height are set to -1, the best requested size is recomputed for the window, and used.

```
procedure Get_Size
  (Window              : access Gtk_Window_Record;
   Width, Height       : out   Gint);
```

Obtains the current size of Window. If Window is not onscreen, it returns the size GTK+ will suggest to the window manager for the initial window size (but this is not reliably the same as the size the window manager will actually select). The size obtained by Get_Size is the last size received in Gdk_Event_Configure, that is, GTK+ uses its locally-stored size, rather than querying the X server for the size. As a result, if you call Resize then immediately call Get_Size, the size won't have taken effect yet. After the window manager processes the resize request, GTK+ receives notification that the size has changed via a configure event, and the size of the window gets updated.

Note 1: Nearly any use of this function creates a race condition, because the size of the window may change between the time that you get the size and the time that you perform some action assuming that size is the current size. To avoid race conditions, connect to "configure_event" on the window and adjust your size-dependent state to match the size delivered in the Gdk_Event_Configure.

Note 2: The returned size does *not* include the size of the window manager decorations (aka the window frame or border). Those are not drawn by GTK+ and GTK+ has no reliable method of determining their size.

Note 3: If you are getting a window size in order to position the window onscreen, there may be a better way. The preferred way is to simply set the window's semantic type with Set_Type_Hint, which allows the window manager to e.g. center dialogs. Also, if you set the transient parent of dialogs with Set_Transient_For window managers will often center the dialog over its parent window. It's much preferred to let the window manager handle these things rather than doing it yourself, because all apps will behave consistently and according to user prefs if the window manager handles it. Also, the window manager can take the size of the window decorations/border into account, while your application cannot.

In any case, if you insist on application-specified window positioning, there's *still*> a better way than doing it yourself - Set_Position will frequently handle the details for you.

```
procedure Reshow_With_Initial_Size
  (Window             : access Gtk_Window_Record);
```

Hide Window, then reshows it, resetting the default size and position.
Used by GUI builders only.

```
procedure Begin_Resize_Drag
  (Window             : access Gtk_Window_Record;
   Edge               :      Gdk.Window.Gdk_Window_Edge;
   Button             :      Gint;
   Root_X             :      Gint;
   Root_Y             :      Gint;
   Timestamp          :      Guint32);
```

Starts resizing a window. This function is used if an application has
window resizing controls. When GDK can support it, the resize will be done using the standard mechanism for the window manager or windowing system. Otherwise, GDK will try to emulate window resizing, potentially not all that well, depending on the windowing system.

209.3.3 Icons

```
procedure Set_Icon_Name
  (Window             : access Gtk_Window_Record;
   Name               :      String);
function Get_Icon_Name
  (Window             : access Gtk_Window_Record)
   return String;
```

Set the icon for the window from a named themed icon. See
Gtk.Icon_Them for more details. This has nothing to do with the WM_ICON_NAME property which is mentioned in the ICCCM (and related to window managers)

```
procedure Set_Icon
  (Window             : access Gtk_Window_Record;
   Icon               :      Gdk.Pixbuf.Gdk_Pixbuf);
```

```
function Get_Icon
  (Window              : access Gtk_Window_Record)
   return Gdk.Pixbuf.Gdk_Pixbuf;
```

Sets up the icon representing Window. This icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts.

The icon should be provided in whatever size it was naturally drawn; that is, don't scale the image before passing it to GTK+. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality.

If you have your icon hand-drawn in multiple sizes, use Set_Icon_List. Then the best size will be used.

This function is equivalent to calling Set_Icon_List with a single element.

See also Set_Default_Icon_List to set the icon for all windows in your application in one go.

```
procedure Set_Icon_List
  (Window              : access Gtk_Window_Record;
   List                :      Glib.Object.Object_Simple_List.Glist);
function Get_Icon_List
  (Window              : access Gtk_Window_Record)
   return Glib.Object.Object_Simple_List.Glist;
```

Sets up the icon representing Window. The icon is used when the window is minimized (also known as iconified). Some window managers or desktop environments may also place it in the window frame, or display it in other contexts.

Set_Icon_List allows you to pass in the same icon in several hand-drawn sizes. The list should contain the natural sizes your icon is available in; that is, don't scale the image before passing it to GTK+. Scaling is postponed until the last minute, when the desired final size is known, to allow best quality.

By passing several sizes, you may improve the final image quality of the icon, by reducing or eliminating automatic image scaling.

Recommended sizes to provide: 16x16, 32x32, 48x48 at minimum, and larger images (64x64, 128x128) if you have them.

Note that transient windows (those who have been set transient for another window using Set_Transient_For) will inherit their icon from their transient parent. So there's no need to explicitly set the icon on transient windows.

```
function Set_Icon_From_File
  (Window              : access Gtk_Window_Record;
   Filename            :      String)
   return Boolean;
```

Equivalent to calling Set_Icon with a pixbuf loaded from Filename. return False on failure.

```
procedure Set_Default_Icon_List
  (List                :      Glib.Object.Object_Simple_List.Glist);
function Get_Default_Icon_List return Glib.Object.Object_Simple_List.Glist;
```

Sets an icon list to be used as fallback for windows that haven't had Set_Icon_List called on them to setup a window-specific icon list.

```
procedure Set_Default_Icon
  (Icon               :      Gdk.Pixbuf.Gdk_Pixbuf);
```

Sets an icon to be used as a fallback for windows that haven't had Set_Icon called on them

```
function Set_Default_Icon_From_File
  (Filename           :      String)
   return Boolean;
```

Same as Set_Default_Icon, loads the pixbuf automatically.

```
procedure Set_Default_Icon_Name
  (Name               :      String);
```

Sets an icon to be used as fallback for windows that haven't had a themed icon set (set Set_Icon_Name).

209.3.4 Groups

```
procedure Gtk_New
  (Group              : out   Gtk_Window_Group);
```

Create a new window group.
Grabs added with Gtk.Main.Grab_Add only affect windows within the same group.

```
function Group_Get_Type         return GType;
```

Return the internal type used for window groups

```
procedure Group_Add_Window
  (Window_Group       : access Gtk_Window_Group_Record;
   Window             : access Gtk_Window_Record'Class);
```

Add a window to Window_Group

```
procedure Group_Remove_Window
  (Window_Group       : access Gtk_Window_Group_Record;
   Window             : access Gtk_Window_Record'Class);
```

Remove a specific window from the group

209.3.5 Focus

```
function Get_Focus
  (Window             : access Gtk_Window_Record)
   return Gtk.Widget.Gtk_Widget;
```

Return the widget that would have the keyboard focus if Window itself has the focus. It currently has the focus only if Has_Focus_Is_Set returns True. To know whether the Window itself currently has the focus, check the Has_Toplevel_Focus_Property property described below

```
procedure Set_Focus
  (Window             : access Gtk_Window_Record;
   Focus              :      Gtk.Widget.Gtk_Widget);
```

Set the focus child for Window.
If Focus is not the current focus widget, and is focusable, sets it as the focus widget for the window. If Focus is null, unsets the focus widget for this window. To set the focus to a particular widget in the toplevel, it is usually more convenient to use gtk_widget_grab_focus() instead of this function.

```
procedure Set_Accept_Focus
  (Window             : access Gtk_Window_Record;
   Setting            :      Boolean);
```

```
function Get_Accept_Focus
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Windows may set a hint asking the desktop environment not to receive the input focus.

```
procedure Set_Focus_On_Map
  (Window             : access Gtk_Window_Record;
   Setting            :      Boolean);
function Get_Focus_On_Map
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Windows may set a hint asking the desktop environment not to receive the input focus when the window is mapped.

```
function Has_Toplevel_Focus
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Returns whether the input focus is within this Window. For real toplevel windows, this is identical to Is_Active, but for embedded windows the results will differ

```
function Is_Active
  (Window             : access Gtk_Window_Record)
   return Boolean;
```

Returns whether the window is part of the current active toplevel. (That is, the toplevel window receiving keystrokes.) The return value is True if the window is active toplevel itself, but also if it is, say, a Gtk_Plug embedded in the active toplevel. You might use this function if you wanted to draw a widget differently in an active window from a widget in an inactive window.

209.3.6 Keys and shortcuts

```
procedure Set_Default
  (Window             : access Gtk_Window_Record;
   Default_Widget     : access Gtk.Widget.Gtk_Widget_Record'Class);
```

The default widget is the widget that's activated when the user presses Enter in a dialog (for example). This function sets or unsets the default widget for a Window. When setting (rather than unsetting) the default widget it's generally easier to call Grab_Focus on the widget. Before making a widget the default widget, you must set the CAN_DEFAULT flag on the widget you'd like to make the default using GTK.WIDGET.SET_FLAGS.

```
procedure Set_Mnemonic_Modifier
  (Window             : access Gtk_Window_Record;
   Modifier           :      Gdk.Types.Gdk_Modifier_Type);
function Get_Mnemonic_Modifier
  (Window             : access Gtk_Window_Record)
   return Gdk.Types.Gdk_Modifier_Type;
```

Sets the mnemonic modifier for this window.
Modifier is the mask used to active mnemonics in this window

```
procedure Add_Mnemonic
  (Window             : access Gtk_Window_Record;
   Keyval             :      Gdk.Types.Gdk_Key_Type;
   Target             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

```
procedure Remove_Mnemonic
  (Window             : access Gtk_Window_Record;
   Keyval             :      Gdk.Types.Gdk_Key_Type;
   Target             : access Gtk.Widget.Gtk_Widget_Record'Class);
```

Add a mnemonic to this window.
Target will receive the "activate" signal when Keyval is pressed inside the window. In addition to keyval, the user must press the special key defined through Set_Mnemonic_Modifier

```
function Mnemonic_Activate
  (Window             : access Gtk_Window_Record;
   Keyval             :      Gdk.Types.Gdk_Key_Type;
   Modifier           :      Gdk.Types.Gdk_Modifier_Type)
   return Boolean;
```

Activates the targets associated with the mnemonic. This sends the
"activate" signal to the corresponding signal

```
function Activate_Key
  (Window             : access Gtk_Window_Record;
   Event              :      Gdk.Event.Gdk_Event_Key)
   return Boolean;
```

Activates mnemonics and accelerators for this window. This is normally
called by the default key_press_event_handler for toplevel windows, however in some cases it may be useful to call this directly when overriding the standard key handling for a toplevel window. Return True if the mnemonic was found and activated.

```
function Propagate_Key_Event
  (Window             : access Gtk_Window_Record;
   Event              :      Gdk.Event.Gdk_Event_Key)
   return Boolean;
```

Propagate a key press or release event to the focus widget and up the
focus container chain until a widget handles Event. This is normally called by the default key_press_event handler, but might be useful when overriding the standard key handling for a toplevel window.

```
procedure Add_Accel_Group
  (Window             : access Gtk_Window_Record;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group);
procedure Remove_Accel_Group
  (Window             : access Gtk_Window_Record;
   Accel_Group        :      Gtk.Accel_Group.Gtk_Accel_Group);
```

Adds or Removes the specified accelerator group for the window, such
that calling Gtk.Accel_Groups.Active on Window will activate accelerators in Accel_Group.

209.4 Example

```
 --  This example shows how you can display a banner while your applica-
tion is
 --  loading

 with Gtk.Window, Gtk.Enums, Gtk.Main, Gtk.Label;
 use Gtk.Window,  Gtk.Enums, Gtk.Main, Gtk.Label;
```

```
procedure Banner is
   Win   : Gtk_Window;
   Label : Gtk_Label;
begin
   Gtk.Main.Init;

   Gtk_New (Win, Window_Popup);
   Set_Policy (Win,
               Allow_Shrink => False,
               Allow_Grow   => False,
               Auto_Shrink  => False);
   Set_Position (Win, Win_Pos_Center);
   Set_Size_Request (Win, 300, 300);

   Gtk_New (Label, "You should show a pixmap instead...");
   Add (Win, Label);

   Show_All (Win);
   Gtk.Main.Main;
end Banner;
```

210 Package MDI_Child

211 Package Pango

This is the top level package of the Pango hierarchy.

212 Package Pango.Attributes

This package provides a set of types and subprograms to manipulate the attributes of text displayed in a pango_layout

212.1 Types

type Pango_Attr_List **is new** Glib.C_Proxy;

type Pango_Attribute **is new** Glib.C_Proxy;

212.2 Subprograms

212.2.1 Attributes

```
function Attr_Underline_New
  (Underline          :      Pango.Enums.Underline)
   return Pango_Attribute;
```

Create a new underline attribute

212.2.2 Attributes list

```
procedure Gdk_New
  (Attr_List          : out   Pango_Attr_List);
```

Create a new empty list of attributes

```
procedure Ref
  (Attr_List          :      Pango_Attr_List);
```

Increment the reference count of the attribute list

```
procedure Unref
  (Attr_List          :      Pango_Attr_List);
```

Decrement the reference count of the attribute list. When it reaches 0, the list is destroyed.

```
procedure Insert
  (Attr_List          :      Pango_Attr_List;
   Attribute          :      Pango_Attribute);
```

Insert a new attribute in the list

213 Package Pango.Font

This package provides high-level, system-independent handling of fonts. It supercedes the old Gdk.Font package, which should no longer be used.

Fonts are defined through several attributes, like their family, weight, size, style, ...

The Pango_Font_Description objects created by this package can either be used directly to draw text through Pango.Layout.Pango_Layout objects (and the associated Gdk.Drawable.Draw_Layout procedure), or by converting them to a Gdk_Font. The first method is the preferred one, and provides high-level handling of multi-line texts or tabs, when you have to handle this yourself in the second case.

213.1 Types

type Pango_Font_Description **is new** Glib.C_Proxy;

type Pango_Font_Metrics **is new** Glib.C_Proxy;

type Pango_Language **is new** Glib.C_Proxy;

type Property_Font_Description **is new** Desc_Properties.Property;

213.2 Subprograms

```
function Get_Type              return Glib.GType;
```

Return the internal gtk+ type associated with font descriptions.

```
function Copy
  (Desc               :      Pango_Font_Description)
   return Pango_Font_Description;
```

Return a newly allocated font description.
This Pango_Font_Description needs to be free'ed after use.

```
function Equal
  (Desc1              :      Pango_Font_Description;
   Desc2              :      Pango_Font_Description)
   return Boolean;
```

Return True if the two font descriptions are identical.
Note that two font description may result in identical fonts being loaded, but still compare False.

```
procedure Free
  (Desc               : in out Pango_Font_Description);
```

Deallocate the given font description.

```
function From_String
  (Str                 :      String)
   return Pango_Font_Description;
```

Create a new font description from the given string representation
of the given form: "[FAMILY-LIST] [STYLE-OPTIONS] [SIZE]". Any one of the options may be omitted.

- FAMILY-LIST is a comma separated list (spaces are not allowed) of font families optionally terminated by a comma. If absent, the font family of the font that will be used is unspecified.
- STYLE_OPTIONS is a whitespace separated list of words where each word describes either style, variant, weight, or stretch. Any unspecified style option is defaulted to "Normal", which respectively corresponds to Pango_Style_Normal, Pango_Weight_Normal, Pango_Variant_Normal, and Pango_Stretch_Normal.
- SIZE is a decimal number describing the size of the font in points. If unspecified, a size of 0 will be used.

```
function To_Font_Description
  (Family_Name         :      String := "";
   Style               :      Pango.Enums.Style
                       := Pango.Enums.Pango_Style_Normal;
   Variant             :      Pango.Enums.Variant
                       := Pango.Enums.Pango_Variant_Normal;
   Weight              :      Pango.Enums.Weight
                       := Pango.Enums.Pango_Weight_Normal;
   Stretch             :      Pango.Enums.Stretch
                       := Pango.Enums.Pango_Stretch_Normal;
   Size                :      Gint := 0)
   return Pango_Font_Description;
```

Create a new font decription from the given parameters.

```
function To_String
  (Desc                :      Pango_Font_Description)
   return String;
```

Create a string representation of a font description. The format
of the string produced follows the syntax used by From_String. The family-list in the string description will have a terminating comma only if the last word of the list is a valid style option.

```
function To_Filename
  (Desc                :      Pango_Font_Description)
   return String;
```

Create a filename representation of a font description. The filename
is identical to the result from calling To_String, but with underscores instead of characters that are untypical in filenames, and in lower case only.

```
function Get_Family
  (Desc                :      Pango_Font_Description)
   return String;
```

Return the Family_Name of the given Pango_Font_Description.

```
procedure Set_Family
  (Desc                :      Pango_Font_Description;
   Name                :      String);
```

Set the Family_Name of the given Pango_Font_Description.

```
function Get_Style
  (Desc                :      Pango_Font_Description)
   return Pango.Enums.Style;
```

Return the Style of the given Pango_Font_Description.

```
procedure Set_Style
  (Desc                :      Pango_Font_Description;
   Style               :      Pango.Enums.Style);
```

Set the Style of the given Pango_Font_Description.

```
function Get_Variant
  (Desc                :      Pango_Font_Description)
   return Pango.Enums.Variant;
```

Return the Variant of the given Pango_Font_Description.

```
procedure Set_Variant
  (Desc                :      Pango_Font_Description;
   Variant             :      Pango.Enums.Variant);
```

Set the Variant of the given Pango_Font_Description.

```
function Get_Weight
  (Desc                :      Pango_Font_Description)
   return Pango.Enums.Weight;
```

Return the Weight of the given Pango_Font_Description.

```
procedure Set_Weight
  (Desc                :      Pango_Font_Description;
   Weight              :      Pango.Enums.Weight);
```

Set the Weight of the given Pango_Font_Description.

```
function Get_Stretch
  (Desc                :      Pango_Font_Description)
   return Pango.Enums.Stretch;
```

Return the Stretch of the given Pango_Font_Description.

```
procedure Set_Stretch
  (Desc                :      Pango_Font_Description;
   Stretch             :      Pango.Enums.Stretch);
```

Set the Stretch of the given Pango_Font_Description.

```
function Get_Size
  (Desc                :      Pango_Font_Description)
   return Gint;
```

Return value: the size for the font description in pango units. (PANGO_SCALE pango units equals one point). Returns 0 if Desc hasn't been initialized.

```
procedure Set_Size
  (Desc                :      Pango_Font_Description;
   Size                :      Gint);
```

Set the size for the font description in pango units. (PANGO_SCALE pango units equals one point)

213.2.1 Languages

The following section provides types and subprograms to identify a@* specific script and language inside a font (Not all characters of a font are used for all languages)

```
function Pango_Language_Get_Typereturn Glib.GType;
```

Return the internal value used to identify a Pango_Language

```
function From_String
  (Language           :      String)
   return Pango_Language;
```

Take a RFC-3066 format language tag as a string and convert it to a Pango_Language pointer that can be efficiently copied (copy the pointer) and compared with other language tags (compare the pointer). Language is something like "fr" (french), "ar" (arabic), "en" (english), "ru" (russian), ...

This function first canonicalizes the string by converting it to lowercase, mapping '_' to '-', and stripping all characters other than letters and '-'.

The returned value need not be freed, it is stored internally by gtk+ in a hash-table.

213.2.2 Metrics

The following subprograms can be used to retrieve the metrics associated@* with the font. Note that such metrics might depend on the specific script/language in use.

```
function Get_Metrics
  (Font               : access Pango_Font_Record'Class;
   Language           :      Pango_Language := null)
   return Pango_Font_Metrics;
```

Gets overall metric information for a font. Since the metrics may be substantially different for different scripts, a language tag can be provided to indicate that the metrics should be retrieved that correspond to the script(s) used by that language.

The returned value must be Unref'ed by the caller.

Language determines which script to get the metrics for, or null to indicate the metrics for the entire font.

```
procedure Ref
  (Metrics            :      Pango_Font_Metrics);
procedure Unref
  (Metrics            :      Pango_Font_Metrics);
```

Ref or unref Metrics When the reference counter reaches 0, the memory is deallocated.

```
function Get_Ascent
  (Metrics            :      Pango_Font_Metrics)
   return Gint;
```

Gets the ascent from a font metrics structure. The ascent is the distance from the baseline to the logical top of a line of text. (The logical top may be above or below the top of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be).

The returned value is expressed in pango units, and must be divided by Pango_Scale to get the value in pixels.

```
function Get_Descent
  (Metrics            :      Pango_Font_Metrics)
   return Gint;
```

Gets the descent from a font metrics structure. The descent is the distance from the baseline to the logical bottom of a line of text. (The logical bottom may be above or below the bottom of the actual drawn ink. It is necessary to lay out the text to figure where the ink will be.)

The returned value is expressed in pango units, and must be divided by Pango_Scale to get the value in pixels.

```
function Get_Approximate_Char_Width
  (Metrics            :      Pango_Font_Metrics)
   return Gint;
```

Gets the approximate character width for a font metrics structure. This is merely a representative value useful, for example, for determining the initial size for a window. Actual characters in text will be wider and narrower than this.

The returned value is expressed in pango units, and must be divided by Pango_Scale to get the value in pixels.

```
function Get_Approximate_Digit_Width
  (Metrics            :      Pango_Font_Metrics)
   return Gint;
```

Gets the approximate digit width for a font metrics structure. This is merely a representative value useful, for example, for determining the initial size for a window. Actual digits in text can be wider and narrower than this, though this value is generally somewhat more accurate than the result of Get_Approximate_Char_Width.

The returned value is expressed in pango units, and must be divided by Pango_Scale to get the value in pixels.

```
function Font_Metrics_Get_Type return Glib.GType;
```

Return the internal value associated with a Pango_Font_Metrics

213.2.3 Properties

See the package Glib.Properties for more information on how to@* use properties

```
function To_Address
  (F                  :      Pango_Font_Description;
   Add                :      System.Address)
   return System.Address;
```

214 Package Pango.Layout

This package provides a high-level object that is capable of arranging text in a visually correct manner. It supports international character sets, although all strings should be Utf8, supports left-to-right and right-to-left writing systems, is capable of handling multi-line texts, and properly aligns tab characters in the text.

This is the base type that is used in the standard gtk+ widget for all the widgets that display some text on the screen.

Since it works directly with Pango.Font.Pango_Font_Description fonts, it is also much better at handling resizing of text, wrapping,... than direct calls to Gdk.Drawable.Draw_Text.

The idea is that this widget is used to compute the layout of the characters (ie their screen position). It doesn't do any rendering, however, and should be used in conjonction with Gdk.Drawable.Draw_Layout to actually display something on the screen.

This widget is independent from any specific drawing systems, and might for instance be used to create postscript documents, for direct access to the win32 API,...

This widget represents one of the fundamental additions to gtk+ 2.0 over what previously existed in the gtk+ 1.x series. It obsoletes the package Gdk.Font, which should only be used for legacy applications.

214.1 Types

```
type Pango_Alignment is
          (Pango_Align_Left,
           Pango_Align_Center,
           Pango_Align_Right);
```

```
type Pango_Ellipsize_Mode is
          (Ellipsize_None,
           Ellipsize_Start,
           Ellipsize_Middle,
           Ellipsize_End);
```

This type describes what sort of (if any) ellipsization should be applied to a line of text. In the ellipsization process characters are removed from the text in order to make it fit to a given width and replaced with an ellipsis.

```
type Pango_Layout_Line is new Glib.C_Proxy;
```

```
type Pango_Wrap_Mode is
          (Pango_Wrap_Word,
           Pango_Wrap_Char);
```

214.2 Subprograms

214.2.1 Creating a layout

A layout can be created in two ways: either from a widget@* (Gtk.Widget.Create_Pango_Layout), from which it will inherit the font and various other attributes, or directly from a Pango_Context.

```
procedure Gdk_New
  (Layout             : out   Pango_Layout;
   Context            : access Pango.Context.Pango_Context_Record'Class);
```

Create a new layout, based on context.

214.2.2 Contexts

```
function Get_Context
  (Layout             : access Pango_Layout_Record)
   return Pango.Context.Pango_Context;
```

Return the context of the layout. The returned value is the internal
context itself, so you must Glib.Object.Ref it if you need to keep a reference. You shouldn't Unref it.

```
procedure Set_Font_Description
  (Layout             : access Pango_Layout_Record;
   Font               :      Pango.Font.Pango_Font_Description);
```

Change the font used in the layout.
If not font description is set for the layout, the font description from the layout's context is used.

```
procedure Context_Changed
  (Layout             : access Pango_Layout_Record);
```

Forces recomputation of any state in Layout that might depend
on the layout's context. This function should be called if you make changes to the context subsequent to creating the layout.

214.2.3 Lines

```
function Get_Line
  (Layout             : access Pango_Layout_Record;
   Line               :      Natural)
   return Pango_Layout_Line;
```

Retrieve a particular line from Layout.
Line must be between 0 and Get_Line_Count (Layout) - 1. null is returned if the index is out of range. The layout line can be Ref'ed and retained, but will become invalid if changes are made to Layout.

```
procedure Line_Ref
  (Line               :      Pango_Layout_Line);
```

Increase the reference count of Line by 1.

```
procedure Line_Unref
  (Line               :      Pango_Layout_Line);
```

Decrease the reference count of Line by 1. If the result is 0, the line
and all associated memory will be destroyed.

```
function Line_Index_To_X
  (Line               :      Pango_Layout_Line;
   Index              :      Integer;
   Trailing           :      Integer)
   return Glib.Gint;
```

Convert an index within a line to an X position.
Index is the byte offset of a graphem within the layout. Trailing is an integer indicating the edge of the grapheme to retrieve the position of. If 0, the trailing edge of the grapheme, otherwise, the leading of the grapheme. The returned value is in pango units.

214.2.4 Getting the size

Pango internally stores its sizes in pango units, which are a number of@* pixels (device units, when not drawing on the screen) multiplied by Pango_Scale. There are generally equivalent subprograms to get the sizes directly in pixels.

```
procedure Get_Extents
  (Layout             : access Pango_Layout_Record;
   Ink_Rect           : out   Gdk.Rectangle.Gdk_Rectangle;
   Logical_Rect       : out   Gdk.Rectangle.Gdk_Rectangle);
```

Compute the logical and ink extents of Layout. Logical extents
are usually what you want for positioning things. The extents are given in pango units; layout coordinates begin at the top left corner of the layout. Logical_Rect is the overall size of the layout, ie it includes Ink_Rect (where the text is actually drawn) and a small border around it.

```
procedure Get_Size
  (Layout             : access Pango_Layout_Record;
   Width              : out   Glib.Gint;
   Height             : out   Glib.Gint);
```

Return the logical size, in pango units, of the layout. This is a
convenience function around Get_Extents.

```
procedure Get_Pixel_Extents
  (Layout             : access Pango_Layout_Record;
   Ink_Rect           : out   Gdk.Rectangle.Gdk_Rectangle;
   Logical_Rect       : out   Gdk.Rectangle.Gdk_Rectangle);
```

Same as Get_Extents, but the returned values are in pixels (or device
units when not drawing on the screen).

```
procedure Get_Pixel_Size
  (Layout             : access Pango_Layout_Record;
   Width              : out   Glib.Gint;
   Height             : out   Glib.Gint);
```

Same as Get_Size, but the returned values are in pixels.

```
procedure XY_To_Index
  (Layout             : access Pango_Layout_Record;
   X_Pango, Y_Pango   :      Glib.Gint;
   Byte_Index         : out   Integer;
   Trailing           : out   Integer;
   Exact              : out   Boolean);
```

Convert from X and Y positions within a layout to the byte index of the
character at that logical position. X and Y are given in pango units, not pixels. If the position is not inside the layout, the closest position is chosen, and Exact is set to False. Trailing is the position in the grapheme where the user clicked. It will either be 0 (left side)

or the number of characters in the grapheme. In some character sets, a given character can be represented by multiple signs on the screen, which is what Trailing relates to.

214.2.5 Manipulating the text

When initially created with Gtk.Widget.Create_Pango_Layout, the layout@* contains some text. Of course, this text may also be changed later in the life of the layout.

```
procedure Set_Text
  (Layout             : access Pango_Layout_Record;
   Text               :      String);
```

Change the text that the layout displays
Text must be a valid UTF8 string. See Glib.Convert for useful conversion functions.

```
function Get_Text
  (Layout             : access Pango_Layout_Record)
   return String;
```

Return the text currently displayed in the layout.
It is more efficient to use the iterators on the layout than get text if you do not need Ada-specific subprograms to act on the text.

```
function Get_Text
  (Layout             : access Pango_Layout_Record)
   return Gtkada.Types.Chars_Ptr;
```

Same a Get_Text, but return directly the C string, which is more
efficient. The returned value should not be freed or modified.

```
function Get_Line_Count
  (Layout             : access Pango_Layout_Record)
   return Glib.Gint;
```

Return the number of lines in Layout

```
procedure Set_Markup
  (Layout             : access Pango_Layout_Record;
   Markup             :      Glib.UTF8_String);
```

Change the text that layout displays.
Markup must be a valid UTF8 String, and might contain markups as defined in the pango markup language.

214.2.6 Layouting the text

```
procedure Set_Justify
  (Layout             : access Pango_Layout_Record;
   Justify            :      Boolean);
```

Set whether or not each complete line should be stretched to fill the
entire width of the layout. This stretching is typically done by adding whitespace, but for some scripts (such as Arabic), the justification is done by extending the characters.

```
function Get_Justify
  (Layout             : access Pango_Layout_Record)
   return Boolean;
```

Return True if each complete line should be stretched to fill the entire
width of the layout.

```
procedure Set_Alignment
  (Layout             : access Pango_Layout_Record'Class;
   Alignment          :      Pango_Alignment);
```

Set the alignment for the layout (how partial lines are positioned
within the horizontal space available).

```
function Get_Alignment
  (Layout             : access Pango_Layout_Record)
   return Pango_Alignment;
```

Return the alignment for the layout.

```
procedure Set_Width
  (Layout             : access Pango_Layout_Record;
   Width              :      Glib.Gint);
```

Set the width to which the lines of Layout should be wrapped. No
wrapping will be performed if Width is -1. Width is given in pango units.

```
function Get_Width
  (Layout             : access Pango_Layout_Record)
   return Glib.Gint;
```

Return the wrapping width of Layout

```
procedure Set_Wrap
  (Layout             : access Pango_Layout_Record;
   Mode               :      Pango_Wrap_Mode);
```

Sets the wrap style; the wrap style only has an effect if a width is set
on the layout with pango_layout_set_width(). To turn off wrapping, set the width to -1.

```
function Get_Wrap
  (Layout             : access Pango_Layout_Record)
   return Pango_Wrap_Mode;
```

Return the current wrap style

```
procedure Set_Tabs
  (Layout             : access Pango_Layout_Record;
   Tabs               :      Pango.Tabs.Pango_Tab_Array);
```

Sets the tabs to use for Layout, overriding the default tabs (by
default, tabs are every 8 spaces). If Tabs is Null_Tab_Array, the default tabs are reinstated.
tabs is copied into the layout; you must free your copy of Tabs yourself.

```
function Get_Tabs
  (Layout             : access Pango_Layout_Record)
   return Pango.Tabs.Pango_Tab_Array;
```

Get the current tab array used by Layout. If no tab array
has been set, then the default tabs are in use and Null_Tab_Array is returned. Default tabs
are every 8 spaces. The return value should be freed with Pango.Tabs.Free.

214.2.7 Attributes

```
procedure Set_Attributes
  (Layout             : access Pango_Layout_Record;
   Attributes         :      Pango.Attributes.Pango_Attr_List);
```

Set the text attributes for a layout object.
Passing null removes the current list of attributes

```
function Get_Attributes
  (Layout             : access Pango_Layout_Record)
   return Pango.Attributes.Pango_Attr_List;
```

Get the text attributes from a layout object

215 Package Pango_Layout

Index

B

C

G

M

P

Table of Contents