GtkAda Reference Manual

Version 2.4.0
Document revision level 1.23
Date: 2004/07/15 22:22:59

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** 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 Gtk.Object.Get_Type.

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

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

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;

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 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.

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.

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.

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.

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:

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

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

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.Messages

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

7.1 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.

7.2 Subprograms

7.2.1 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.

8 Package Glib.Module

This package provides wrapper code for dynamic module loading

8.1 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.

8.2 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.

9 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

9.1 Signals

- **"notify"**

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

  Emitted when the property Name has been modified

9.2 Types

type GObject_Class **is private**;

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**

9.3 Subprograms

```
function Is_Created
       (Object              : in    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.

9.3.1 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.

9.3.2 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.

9.3.3 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 String;
```

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

9.3.4 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.

```
procedure Initialize_Class_Record
    (Object               : access GObject_Record'Class;
     Signals              :      Gtkada.Types.Chars_Ptr_Array;
     Class_Record         : in out GObject_Class;
     Type_Name            :      String;
     Parameters           :      Signal_Parameter_Types
                          := Null_Parameter_Types);
```

Create the class record for a new object type.
It is associated with Signals'Length new signals. A pointer to the newly created structure is also returned in Class_Record. If Class_Record /= System.Null_Address, no memory allocation is performed, we just reuse it. As a result, each instantiation of an object will share the same GObject_Class, exactly as is done for gtk+.

Note: The underlying C object must already have been initialized by a call to its parent's Initialize function. Parameters'Length should be the same as Signals'Length, or the result is undefined. As a special case, if Parameters has its default value, all signals are created with no argument. This is done for backward compatibility mainly, and you should instead give it an explicit value. Type_Name should be a unique name identifying the name of the new type.

Only the signals with no parameter can be connected from C code. However, any signal can be connected from Ada. This is due to the way we define default marshallers for the signals.

```
function Type_From_Class
    (Class_Record         :      GObject_Class)
     return GType;
```

Return the internal gtk+ type that describes the newly created Class_Record

function Class_From_Type
```
(Typ                  :      GType)
 return GObject_Class;
```
Return the class record for a specific type

9.3.5 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.

9.3.6 Lists

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

function C_Signal_Query_Size **return** Natural;

10 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

10.1 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;

10.2 Subprograms

```
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;
```

11 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.

11.1 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**

11.2 Subprograms

procedure Unref
```
(Param : Param_Spec);
```
Decrement the reference counter. If it reaches 0, the memory is freed.

11.2.1 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

11.2.2 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.

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

function Minimum
```
(Param              :      Param_Spec_Uchar)
 return Glib.Guint8;
```

function Default
```
(Param              :      Param_Spec_Uchar)
 return Glib.Guint8;
```

function Default
```
(Param              :      Param_Spec_Boolean)
 return Boolean;
```

```
function Minimum
       (Param                :      Param_Spec_Int)
        return Glib.Gint;
```

```
function Default
       (Param                :      Param_Spec_Int)
        return Glib.Gint;
```

```
function Minimum
       (Param                :      Param_Spec_Uint)
        return Glib.Guint;
```

```
function Default
       (Param                :      Param_Spec_Uint)
        return Glib.Guint;
```

```
function Minimum
       (Param                :      Param_Spec_Long)
        return Glib.Glong;
```

```
function Default
       (Param                :      Param_Spec_Long)
        return Glib.Glong;
```

```
function Minimum
       (Param                :      Param_Spec_Ulong)
        return Glib.Gulong;
```

```
function Default
       (Param                :      Param_Spec_Ulong)
        return Glib.Gulong;
```

```
function Default
       (Param                :      Param_Spec_Unichar)
        return Gunichar;
```

```
function Enumeration
       (Param               :      Param_Spec_Enum)
        return Enum_Class;
```

```
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 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 Default
       (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 Default
```
        (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_Nth
```
        (Param               :      Param_Spec_String)
         return String;
```

function Ensure_Non_Null
```
        (Param               :      Param_Spec_String)
         return Boolean;
```

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;
```

11.2.4 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.

12 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.

12.1 Types

```
type File_Conversion_Hook_Type is access function
        (Type_Name : String) return Glib.Object.GObject;
```

This variable can be point to one of your functions. It gets the name of a C widget (ex/ "GtkButton") and should return a newly allocated Ada widget.

```
type Hook_List is record
        Func : File_Conversion_Hook_Type;
        Next : Hook_List_Access := null;
        end record;
```

Internal structure used for the list.

```
type Hook_List_Access is access Hook_List;
```

12.2 Subprograms

```
procedure Add_Hook
      (Func                 :       File_Conversion_Hook_Type);
```

Add a new function to the list of hooks for file conversions.
All the hooks are called when GtkAda finds a type which is not one of the standard types.

```
function Conversion_Hooks     return Hook_List_Access;
```

Return the head of the hook list.

13 Package Glib.Unicode

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

13.1 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/UnicodeData.html

13.2 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.

13.2.1 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

13.2.2 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

13.2.3 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.

13.2.4 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.

14 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.

15 Package Glib.XML

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

15.1 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.

15.2 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

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.

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.
Return Default otherwise.

procedure Set_Attribute
```
(N                    :      Node_Ptr;
 Attribute_Name, Attribute_Value  :      UTF8_String);
```
Create a new attribute, or replace an existing one

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).

16 Package Gtkada

This is the parent package for the GtkAda specific units.

17 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:

- 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).
- 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 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%

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).

17.1 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.

- **"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

17.2 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;
```

type Selection_Iterator **is private**;

17.3 Subprograms

17.3.1 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 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.

17.3.2 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)
        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);
```

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.

17.3.3 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 by the zoom level (map from world
coordinates to canvas coordinates). 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).

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).

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.

17.3.4 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 the items will be automatically layed out
when inserted in the canvas, if no coordinates are specified.

procedure Layout
```
   (Canvas           : access Interactive_Canvas_Record;
    Force            :        Boolean := False;
    Vertical_Layout  :        Boolean := False);
```
Recompute the layout of the canvas.
If Force is False, only the items that don't already have a location are moved.

17.3.5 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.

17.3.6 Drawing links

Drawing of links can be controlled at several levels:

- 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.
- 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.
- 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.

```
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);
```

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;
        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.

17.3.7 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.

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

Return True if the item is currently selected

function Start
```
(Canvas              : access Interactive_Canvas_Record)
 return Selection_Iterator;
```

Return the first selected item

function Next
```
(Iterator            :        Selection_Iterator)
 return Selection_Iterator;
```

Move to the next selected item

function Get
```
(Iterator            :        Selection_Iterator)
 return Canvas_Item;
```

Return the current item, or null if there is no more selected item.

17.3.8 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. 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.

17.3.9 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%

17.3.10 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

17.4 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;
```

18 Package Gtkada.Dialogs

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

18.1 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.

18.2 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.

19 Package Gtkada.File_Selection

This package provides a high level support for creating file selection dialogs by handling the signals internally.

19.1 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.

20 Package Gtkada.Handlers

This package provides the most commonly used instantiations of Gtk.Handlers

Gate takes advantage of these pre-instantiated packages.

21 Package Gtkada.Intl

This package provides support for string internationalization using the libintl library.

To change the current locale setting, use on the environment variables "LANG" or "LC_MESSAGES". For example, to switch to the french locale using bash:

$ export LANG=fr

Depending on the specific implementation of gettext, the following environment variables may be set to change the default settings of locale parameters:

- LANG Specifies locale name.
- LC_MESSAGES Specifies messaging locale, and if present overrides LANG for messages.
- TEXTDOMAIN Specifies the text domain name, which is identical to the message object filename without .mo suffix.
- 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).

See the gettext documentation of your specific OS for more details.

The recommended way to use the gettext capability in your application is to use Dgettext with your own domain, and define the following shortcut:

function "-" (Msg : UTF8_String) return UTF8_String; – Convenient shortcut to the Gettext function.

function "-" (Msg : UTF8_String) return UTF8_String is begin return Dgettext ("my_domain", Msg); end "-";

Do not forget to call Bind_Text_Domain ("my_domain", "my locale prefix") at the beginning of your program. For example, if the prefix of your application is /usr, the standard location of the locale would be /usr/share/locale, e.g: Bind_Text_Domain ("GtkAda", "/usr/share/locale");

Under this locale directory, the functions provided by this package will look for the directory $LANG/LC_MESSAGES, /usr/share/locale/fr/LC_MESSAGES in our example; and in this directory, the file <domain>.mo will be used, e.g /usr/share/locale/fr/LC_MESSAGES/GtkAda.mo

The .mo files can be generated using the GNU tool msgfmt that takes a text file containing for each string the original and the translation. See msgfmt documentation for more details. 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.m

Then, to enable the string translation in your application, use the "-" function defined above, e.g:

Gtk_New (Label, -"Help"); Gtk_New (Label, -("Help ?") & ASCII.LF & -("Yes"));

21.1 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.

22 Package Gtkada.Pixmaps

This package provides a collection of "standard" pixmaps

22.1 Subprograms

```
function "+"
       (Str                  : in    String)
        return Gtkada.Types.Chars_Ptr;
```

23 Package Gtkada.Types

This package provides GtkAda specific types and their associated functions.

23.1 Types

subtype Chars_Ptr **is** Interfaces.C.Strings.chars_ptr;

subtype Chars_Ptr_Array **is** Interfaces.C.Strings.chars_ptr_array;

23.2 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.

23.2.1 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.

24 Package Canvas_Link

25 Package GObject

26 Package Gdk

This is the top level package of the Gdk hierarchy. It provides the type definitions used to access underlying C structures.

26.1 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_Visual **is new** C_Proxy;

subtype Gdk_Window **is** Gdk_Drawable;

type Gdk_Window_Attr **is new** C_Proxy;

27 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 Chapter 36 [Package_Gdk.Pixmap], page 153 for more details on pixmap handling.

27.1 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.

27.2 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.

28 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.

28.1 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.

28.2 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

28.2.1 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.

28.2.2 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.

28.3 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;
```

29 Package Gdk.Cursor

This package provides the capability to create predefined mouse cursors as well as user defined ones.

29.1 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);
```

29.2 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.
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 Destroy
       (Cursor               :       Gdk_Cursor);
```

Destroy a cursor, freeing any resources allocated for it.
Deprecated, use Unref instead.

```
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.

30 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 Chapter 33 [Package_Gdk.GC], page 129 for more information.

30.1 Types

subtype Gdk_Drawable **is** Gdk.Gdk_Drawable;

A screen area that can be drawn upon.

30.2 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;
```

30.3 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;
```

31 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...

31.1 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;

31.2 Subprograms

31.2.1 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.

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;
```
???

31.2.2 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.

31.2.3 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.

31.2.4 GValue support

function Get_Event
```
(Value : Glib.Values.GValue)
 return Gdk_Event;
```
Convert a value into a Gdk_Event.

31.2.5 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.

32 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:

- adobe : foundry
- courier : font family
- bold : weight (e.g. bold, medium)
- o : slant (e.g. roman, italic, oblique)
- normal : set width (e.g. normal, condensed, narrow, double)
- 10 : pixels
- 100 : points (in tenths of a point)
- 75 : horizontal resolution in dpi
- 75 : vertical resolution in dpi
- m : spacing (e.g. monospace or proportional)
- 60 : average width (in tenths of a pixel)
- iso8859-1 : character set

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.

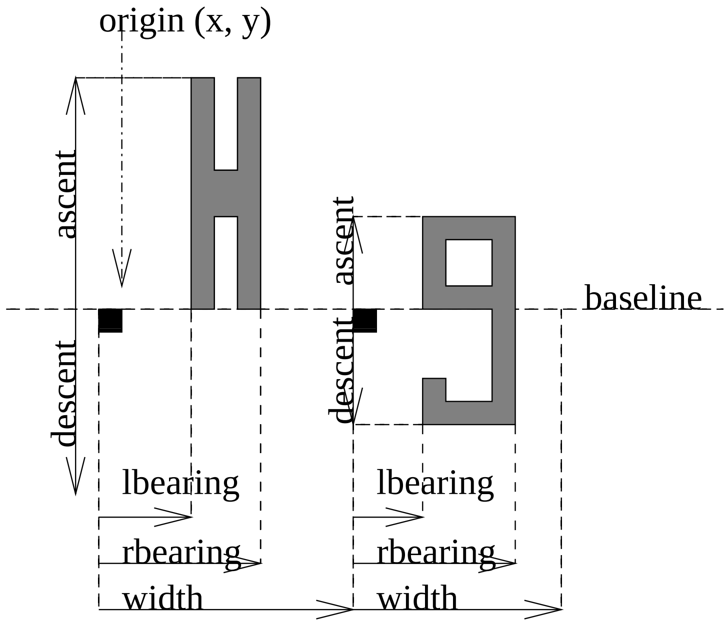
An easy way to select a font is by using some external programs, for instance xfontsel, xlsfont, gfontsel, or even the font selection dialog example in the testgtk/ directory of the GtkAda distribution.

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.



32.1 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).

32.2 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.

33 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.

33.1 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);
```

33.2 Subprograms

33.2.1 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_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);
```

33.2.2 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

34 Package Gdk.Main

This package provides routines to handle initialization and set up of the Gdk library.

34.1 Types

type Gdk_Grab_Status **is**
```
(Grab_Success,
 Grab_Already_Grabbed,
 Gdk_Grab_Invalid_Time,
 Gdk_Grab_Not_Viewable,
 Gdk_Grab_Frozen);
```

34.2 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 Chapter 21 [Package_Gtkada.Intl], page 82 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)
    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);
```

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)
    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);
```
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);
```

35 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.

35.1 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);
```

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 **is new** Glib.C_Proxy;

A very efficient client-side pixmap. This type can be adapted to all the possible screen depths (number of bits per pixel), and the algorithms are extremely efficient. You can also load a pixbuf directly from an external file in one of the standard image formats.

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.

35.2 Subprograms

35.2.1 Get_Type

function Get_Type **return** Glib.GType;

Return the internal value associated with a Gdk_Pixbuf.

35.2.2 Reference counting

procedure Ref
```
   (Pixbuf          :      Gdk_Pixbuf);
```

Increment the reference counting on the image.
The image is destroyed when its reference counting reaches 0. Note also that most of the

time you won't have to call this function yourself.

procedure Unref
```
(Pixbuf              :      Gdk_Pixbuf);
```
Decrement the reference counting on the image.

35.2.3 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.

35.2.4 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_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);
```

Save pixbuf to a file.
Quality is only taken into account for JPEG images. 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.

35.2.5 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);
```

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.

35.2.6 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.

35.2.7 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."

35.2.8 Iterators

function Get_Type_Animation_Iter**return** 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.

36 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 Chapter 27 [Package_Gdk.Bitmap], page 90 for more details on bitmap handling.

36.1 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.

36.2 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.

37 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).

37.1 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** (Natural) **of** Glib.Guchar;

type Rgb_Buffer_Access **is access all** 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, ...)

37.2 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_Visual (Gdk.Rgb.Get_Visual);
- Gtk.Widget.Push_Colormap (Gdk.Rgb.Get_Cmap);
- Gtk_New (....)
- Gtk.Widget.Pop_Visual;
- Gtk.Widget.Pop_Colormap;

37.2.1 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);

37.2.2 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.

37.2.3 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);
```

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);
```

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);
```

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);
```

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);
```

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);
```

Same kind of function as above, but for different buffer types (???).

38 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).

38.1 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).

39 Package Glade

This package is a binding to the libglade library that provides routines to create widgets dynamically from an XML definition file. see Chapter 40 [Package_Glade.XML], page 163.

39.1 Subprograms

39.1.1 dynamic loading of libglade extensions

procedure Require
```
(Library            :       String);
```

procedure Provide
```
(Library            :       String);
```

40 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 Chapter 39 [Package_Glade], page 162.

40.1 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 Chapter 21 [Package_Gtkada.Intl], page 82 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.

procedure Initialize_From_Buffer
```
(XML              : access Glade_XML_Record'Class;
 Buffer           :       String;
 Root             :       String := "";
 Domain           :       String := "");
```

Internal initialization function.
See the section "Creating your own widgets" in the documentation.

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.

41 Package Glade_XML

42 Package Gtk

This package provides some basic Gtk+ functionalities such as getting the version number. For general GtkAda initializations, see Chapter 102 [Package_Gtk.Main], page 404.

42.1 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.

42.2 Subprograms

function Major_Version **return** Guint;

Return the major version number for Gtk+.
Note that this is not necessarily the same as for GtkAda. 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.

43 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.

43.1 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)
```

43.2 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.

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.

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 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.

43.3 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);
```

44 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):

```
  [-------|================|-------------------]
 lower    value        value + page_size       upper
```

44.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Adjustment             (Package Gtk.Adjustment)
```

44.2 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

44.3 Subprograms

```
procedure Gtk_New
       (Adjustment         : out   Gtk_Adjustment;
        Value              :       Gdouble;
        Lower              :       Gdouble;
        Upper              :       Gdouble;
        Step_Increment     :       Gdouble;
        Page_Increment     :       Gdouble;
        Page_Size          :       Gdouble);
```

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 the size of the area that is currently visible (for instance in a Gtk_Scrolled_Window).

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Adjustment.

44.3.1 Read functions

function Get_Value
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the current value of the adjustment.

function Get_Lower
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the lower bound of the adjustment.

function Get_Upper
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the upper bound of the adjustment.

function Get_Step_Increment
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the step increment of the adjustment.

function Get_Page_Increment
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the page increment of the adjustment.

function Get_Page_Size
```
(Adjustment          : access Gtk_Adjustment_Record)
 return Gdouble;
```
Return the page size of the adjustment.

44.3.2 Write functions

procedure Set_Value
```
(Adjustment          : access Gtk_Adjustment_Record;
 Value               :        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);
```
Modify the lower bound of the adjustment.
You should call Changed after modifying this value.

procedure Set_Upper
```
(Adjustment        : access Gtk_Adjustment_Record;
 Upper             :        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);
```
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);
```
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);
```
Modify the page size of the adjustment.
You should call Changed after modifying this value.

44.3.3 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.

44.3.4 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.

44.4 Example

```
 [-----|==================|------------]
lower   value         value + page_size       upper
```

45 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.

45.1 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)
```

45.2 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.

46 Package Gtk.Arguments

This package is obsolete and replaced by Glib.Values (see Chapter 14 [Package_Glib.Values], page 55. 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.

47 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.

47.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Misc             (Package Gtk.Misc)
         \___ Gtk_Arrow         (Package Gtk.Arrow)
```

47.2 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.

48 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).

48.1 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)
```

48.2 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.

49 Package Gtk.Bin

Base class for containers that have only one child. This widget can not be instantiated directly.

49.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Widget                 (Package Gtk.Widget)
      \___ Gtk_Container           (Package Gtk.Container)
         \___ Gtk_Bin              (Package Gtk.Bin)
```

49.2 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.

50 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.

50.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Box         (Package Gtk.Box)
```

50.2 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.

```
procedure Initialize_Vbox
       (Box                : access Gtk_Box_Record'Class;
        Homogeneous        :        Boolean := False;
        Spacing            :        Gint := 0);
```
Internal initialization function.
See the section "Creating your own widgets" in the documentation.

```
procedure Initialize_Hbox
       (Box                : access Gtk_Box_Record'Class;
        Homogeneous        :        Boolean := False;
        Spacing            :        Gint := 0);
```
Internal initialization function.
See the section "Creating your own widgets" in the documentation.

```
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 Pack_Start_Defaults
```
(In_Box       : access Gtk_Box_Record;
 Child        : access Gtk.Widget.Gtk_Widget_Record'Class);
```
This is the same as Pack_Start if you use the default parameter values.
It is provided for backward compatibility only.

procedure Pack_End_Defaults
```
(In_Box       : access Gtk_Box_Record;
 Child        : access Gtk.Widget.Gtk_Widget_Record'Class);
```
This is the same as Pack_End if you use the default parameter values.
It is provided for backward compatibility only.

procedure Set_Homogeneous
```
(In_Box       : access Gtk_Box_Record;
 Homogeneous  :        Boolean);
```
Modify 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).

function Get_Homogeneous
```
(In_Box       : access Gtk_Box_Record)
 return Boolean;
```
Return the homogeneous parameter of the box.

procedure Set_Spacing
```
(In_Box       : access Gtk_Box_Record;
 Spacing      :        Gint);
```
Modify the spacing for the box.
I.e. the amount of space left between two adjacent children.

function Get_Spacing

```
        (In_Box              : access Gtk_Box_Record)
         return Gint;
```

Return the spacing of the box.

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 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.

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);
```

Modify the packing for a child.

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.

51 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:

```
+------------------------------------------------+
|                   BORDER                       |
|  +------------------------------------------+  |
|  |\\\\\\\\\\\\\\\\DEFAULT\\\\\\\\\\\\\\\\   |  |
|  |\\+------------------------------------+  |  |
|  |\\| |           SPACING       3      | |  |  |
|  |\\| +--------------------------------+ |  |  |
|  |\\| |########## FOCUS ###############| |  |  |
|  |\\| |#+----------------------------+#| |  |  |
|  |\\| |#|         RELIEF            \|#| |  |  |
|  |\\| |#|  +-----------------------+\|#| |  |  |
|  |\\|1|#|  +     THE TEXT          +\|#|2|  |  |
|  |\\| |#|  +-----------------------+\|#| |  |  |
|  |\\| |#| \\\\\ ythickness \\\\\\\\\\|#| |  |  |
|  |\\| |#+----------------------------+#| |  |  |
|  |\\| |########### 1 ##################| |  |  |
|  |\\| +--------------------------------+ |  |  |
|  |\\| |        default spacing   4     | |  |  |
|  |\\+------------------------------------+  |  |
|  |\            ythickness                   |  |
|  +------------------------------------------+  |
|                border_width                    |
+------------------------------------------------+
```

51.1 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)
```

51.2 Signals

- **"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.

51.3 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.

procedure Initialize_From_Stock
```
(Button             : access Gtk_Button_Record'Class;
 Stock_Id           :       String);
```
Internal initialization function.

procedure Initialize_With_Mnemonic
```
(Button             : access Gtk_Button_Record'Class;
 Label              :       UTF8_String);
```
Internal initialization function.

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);
```
Modify the relief style for the button.
This modifies only its visual aspect, not its behavior.

function Get_Relief
```
(Button             : access Gtk_Button_Record)
 return Gtk.Enums.Gtk_Relief_Style;
```
Get the current relief style for the button

procedure Set_Label
```
(Button             : access Gtk_Button_Record;
 Label              :       UTF8_String);
```
Set the label of the button.

function Get_Label
```
(Button             : access Gtk_Button_Record)
 return UTF8_String;
```
Return the label of the button.

procedure Set_Use_Underline
```
(Button           : access Gtk_Button_Record;
 Use_Underline    :        Boolean);
```
Set whether underscore is used to designate an accelerator.

function Get_Use_Underline
```
(Button           : access Gtk_Button_Record)
 return Boolean;
```
Return whether underscores are used to designate an accelerator.

procedure Set_Use_Stock
```
(Button           : access Gtk_Button_Record;
 Use_Stock        :        Boolean);
```
Set whether a stock item is used by the button.

function Get_Use_Stock
```
(Button           : access Gtk_Button_Record)
 return Boolean;
```
Return whether a stock item is used by the button.

51.3.1 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

51.4 Example

```
+--————————————————————-+
|                  BORDER                          |
|  +--————————————————-+  |
|  |\\\\\\\\\\\\\\\\DEFAULT\\\\\\\\\\\\\\\\  |  |
|  |\\+--———————————————-+ | |
|  |\\| |           SPACING       3      | |  |  |
|  |\\| +--——————————————+ | |  |
|  |\\| |########## FOCUS ###############| |  |  |
|  |\\| |#+--————————————+#| |  |  |
|  |\\| |#|         RELIEF            \|#| |  |  |
|  |\\| |#|  +--———————————+“|#| |  |  |
|  |\\|1|#|  +     THE TEXT          +\|#|2|  |  |
|  |\\| |#|  +--———————————+“|#| |  |  |
|  |\\| |#| \\\\\ ythickness \\\\\\\\\\|#| |  |  |
|  |\\| |#+--————————————+#| |  |  |
|  |\\| |########### 1 #################| |  |  |
|  |\\| +--——————————————+ | |  |
|  |\\| |        default spacing   4     | |  |  |
|  |\\+--———————————————-+ | |
|  |\            ythickness                    |  |
|  +--————————————————-+  |
|               border_width                      |
+--————————————————————-+
```

52 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.

52.1 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)
```

52.2 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);
```
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.

function Get_Layout
```
(Button_Box        : access Gtk_Button_Box_Record)
 return Enums.Gtk_Button_Box_Style;
```
Return the layout used 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);
```

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.

53 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.

53.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Calendar       (Package Gtk.Calendar)
```

53.2 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 a 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.

53.3 Types

type Gtk_Calendar_Display_Options **is private**;

53.4 Subprograms

function "and"
```
(Left, Right        :      Gtk_Calendar_Display_Options)
 return Gtk_Calendar_Display_Options;
```

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 Display_Options

```
(Calendar          : access Gtk_Calendar_Record;
 Flags             :        Gtk_Calendar_Display_Options);
```

Change the display options.
See individual Display_Option flags for more details.

procedure Get_Date

```
(Calendar          : access Gtk_Calendar_Record;
 Year              : out    Guint;
 Month             : out    Guint;
 Day               : out    Guint);
```

Return the date currently selected.

procedure Freeze

```
(Calendar          : access Gtk_Calendar_Record);
```

Lock the display of the calendar until it is thawed.

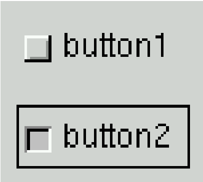
procedure Thaw

```
(Calendar          : access Gtk_Calendar_Record);
```

Defrost a calendar.
All the changes made since the last Freeze are displayed.

54 Package Gtk.Check_Button

A Gtk_Check_Button places a discrete Gtk_Toggle_Button next to a widget, (usually a Gtk_Label).



54.1 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)
```

54.2 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

```
procedure Initialize_With_Mnemonic
       (Check_Button       : access Gtk_Check_Button_Record'Class;
        Label              :      UTF8_String);
```

Internal initialization function.

function Get_Type **return** Glib.GType;

Return the internal value associated with a Gtk_Check_Button.

55 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.

55.1 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)
```

55.2 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.

55.3 Subprograms

procedure Gtk_New
```
(Check_Menu_Item    : out   Gtk_Check_Menu_Item;
 Label              :       UTF8_String := "");
```
Create a new Gtk_Check_Menu_Item with a label, if label isn't null.

procedure Gtk_New_With_Mnemonic
```
(Check_Menu_Item    : out   Gtk_Check_Menu_Item;
 Label              :       UTF8_String);
```
Create a new Gtk_Check_Menu_Item containing a label. The label
will be created will be created using Gtk.Label.New_With_Mnemonic, so underscores in the label indicate the mnemonic for the menu item.

procedure Initialize_With_Mnemonic
```
(Check_Menu_Item    : access Gtk_Check_Menu_Item_Record'Class;
 Label              :       UTF8_String);
```

Internal initialization function.
See the section "Creating your own widgets" in the documentation.

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);
```
Set the active state of the menu item's check box.

function Get_Active
```
(Check_Menu_Item    : access Gtk_Check_Menu_Item_Record)
 return Boolean;
```
Return True if the Item is active

procedure Toggled
```
(Check_Menu_Item    : access Gtk_Check_Menu_Item_Record);
```
Emit the "toggled" signal.

procedure Set_Inconsistent
```
(Check_Menu_Item    : access Gtk_Check_Menu_Item_Record;
 Setting            :        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.

function Get_Inconsistent
```
(Check_Menu_Item    : access Gtk_Check_Menu_Item_Record)
 return Boolean;
```
Return True if the Item is inconsistent.

56 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.

56.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Clist       (Package Gtk.Clist)
```

56.2 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.

56.3 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);
```

56.4 Subprograms

56.4.1 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.

56.4.2 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.

56.4.3 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.

56.4.4 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.

56.4.5 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.

56.4.6 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.

56.4.7 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).

56.4.8 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.

56.5 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 following
--  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;
```

57 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 Chapter 58 [Package_Gtk.Color_Selection_Dialog], page 221 for a version of this widget that comes with its own dialog.

see Chapter 70 [Package_Gtk.Extra.Color_Combo], page 269 for a different way to select colors.

57.1 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)
```

57.2 Signals

- **"color_changed"**

  procedure Handler
  (Selection : access Gtk_Color_Selection_Record'Class);

  Called every time a new color is selected in the dialog

57.3 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.

57.4 Subprograms

```
procedure Gtk_New
      (Widget              : out   Gtk_Color_Selection);
```

Create a new color selection widget.

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

Return the internal value associated with a Gtk_Color_Selection.

```
function Get_Has_Opacity_Control
      (Colorsel            : access Gtk_Color_Selection_Record)
       return Boolean;
```

Return True if the Colorsel has an opacity control.

```
procedure Set_Has_Opacity_Control
      (Colorsel            : access Gtk_Color_Selection_Record;
       Has_Opacity         :        Boolean);
```

Set the Colorsel to use or not use opacity.

```
function Get_Has_Palette
      (Colorsel            : access Gtk_Color_Selection_Record)
       return Boolean;
```

Return True if the Colorsel has a palette.

```
procedure Set_Has_Palette
      (Colorsel            : access Gtk_Color_Selection_Record;
       Has_Palette         :        Boolean);
```

If Has_Palette is True, then set the Colorsel to show the palette.
Hide the palette otherwise.

```
procedure Set_Current_Color
      (Colorsel            : access Gtk_Color_Selection_Record;
       Color               :        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_Current_Alpha
      (Colorsel            : access Gtk_Color_Selection_Record;
       Alpha               :        Guint16);
```

Set the current opacity to be Alpha. When called for the first time,
the original opacity will be set too.

```
procedure Get_Current_Color
      (Colorsel            : access Gtk_Color_Selection_Record;
```

```
 Color               : out   Gdk.Color.Gdk_Color);
```

Set Color to the value of the current color in the Colorsel.

```
function Get_Current_Alpha
   (Colorsel           : access Gtk_Color_Selection_Record)
    return Guint16;
```

Return the current opacity.

```
procedure Set_Previous_Color
   (Colorsel           : access Gtk_Color_Selection_Record;
    Color              :        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.

```
procedure Set_Previous_Alpha
   (Colorsel           : access Gtk_Color_Selection_Record;
    Alpha              :        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 Get_Previous_Color
   (Colorsel           : access Gtk_Color_Selection_Record;
    Color              : out   Gdk.Color.Gdk_Color);
```

Set Color to the original color value.

```
function Get_Previous_Alpha
   (Colorsel           : access Gtk_Color_Selection_Record)
    return Guint16;
```

Return the previous alpha value.

```
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.

58 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.

58.1 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)
```

58.2 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.

58.2.1 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.

59 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.

59.1 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)
```

59.2 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.

function Get_Entry
```
(Combo_Box          : access Gtk_Combo_Record)
 return Gtk.GEntry.Gtk_Entry;
```

Return the Gtk_Entry associated with a Combo_Box.

```
procedure Set_Entry
       (Combo_Box          : access Gtk_Combo_Record;
        GEntry             :      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.

59.3 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);
```

60 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.

60.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Container            (Package Gtk.Container)
```

60.2 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.

60.3 Types

type Forall_Function **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.

60.4 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);
```
Modify the size of the frame that surrounds the widget.
The exact visual impact depends on the specific widget class.

function Get_Border_Width
```
       (Container          : access Gtk_Container_Record)
        return Guint;
```
Return the size of the frame that surrounds the widget.

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);
```

Change the resizing behavior for the Container.
The default value is Resize_Parent.

function Get_Resize_Mode
```
        (Container           : access Gtk_Container_Record)
         return Gtk.Enums.Gtk_Resize_Mode;
```

Return the resizing behavior for the Container.

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.

function Children
```
        (Container           : access Gtk_Container_Record)
         return Gtk.Widget.Widget_List.Glist;
```

This function is deprecated.

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.

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

procedure Unset_Focus_Chain
```
        (Container           : access Gtk_Container_Record);
```

Unset the focus chain.

60.4.1 Foreach functions

procedure Forall
```
(Container          : access Gtk_Container_Record;
 Func               :        Forall_Function);
```
Execute Func for each of the children of Container.
See also the generic package Forall_Pkg if you want to pass some extra data to Func.

60.4.2 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.

procedure Set_Focus_Vadjustment
```
(Container          : access Gtk_Container_Record;
 Adjustment         :        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);
```
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).

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.

60.4.3 Signals emission

procedure Check_Resize
```
(Container          : access Gtk_Container_Record);
```
Emit the "check_resize" signal

procedure Set_Focus_Child
```
(Container          : access Gtk_Container_Record;
 Child              : access Gtk.Widget.Gtk_Widget_Record'Class);
```
Emit a "set_focus_child" signal.

function Get_Focus_Child
```
(Container          : access Gtk_Container_Record)
 return Gtk.Widget.Gtk_Widget;
```
Return the child that currently has the keyboard focus. Note that null can be returned.

61 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.

61.1 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)
```

61.2 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.

61.3 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;
Similar to Clist_Row, but for a Ctree.

61.4 Subprograms

61.4.1 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.

61.4.2 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

61.4.3 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)

61.4.4 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.

61.4.5 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;
```

Return the type of the cell at Node/Column.
This indicates which of the functions Node_Get_Text. Node_Get_Pixmap, etc. should be used with this cell.

```
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.

61.4.6 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.

61.4.7 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.

61.4.8 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.

62 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.

62.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Drawing_Area       (Package Gtk.Drawing_Area)
         \___ Gtk_Curve           (Package Gtk.Curve)
```

62.2 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.

62.3 Types

type Property_Gtk_Curve_Type **is new** Curve_Type_Properties.Property;

62.4 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);
```
Set the vector of points on the curve.
The curve type is set to Curve_Type_Free.

procedure Get_Vector
```
(Curve              : access Gtk_Curve_Record;
 Vector             : out    Gfloat_Array);
```
Return a vector of points representing the curve.

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.

63 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 Chapter 18 [Package_Gtkada.Dialogs], page 78 for a higher level dialog interface.

63.1 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)
```

63.2 Signals

- **"close"**

  procedure Handler (Dialog : access Gtk_Fialog_Record'Class);

  Emit this signal to force a closing of the dialog.
- **"response"**

  procedure Handler
  (Dialog : access Gtk_Fialog_Record'Class;
  Response_Id : Gint);

63.3 Types

type Gtk_Dialog_Flags **is mod** 3;

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;

63.4 Subprograms

63.4.1 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.

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);
```
Set whether the dialog has a separator above the buttons.

function Get_Has_Separator
```
   (Dialog              : access Gtk_Dialog_Record)
    return Boolean;
```
Return True if 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.

63.4.2 Signals

```
procedure Response
   (Dialog       : access Gtk_Dialog_Record;
    Response_Id  :        Gtk_Response_Type);
```

Emit the "response" signal

64 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.

- Defining a widget as a possible drag source

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.

- Defining a widget as a possible drop site

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.

64.1 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.

64.2 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.

64.3 Subprograms

64.3.1 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.

```
function Dest_Find_Target
       (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        Context            :        Drag_Context;
        Target_List        :        Gtk.Selection.Target_List)
        return Gdk.Types.Gdk_Atom;
```

```
function Dest_Get_Target_List
       (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class)
        return Target_List;
```

```
procedure Dest_Set_Target_List
       (Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        Target_List        :        Gtk.Selection.Target_List);
```

64.3.2 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, it 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.

64.3.3 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.

64.3.4 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_Default_Icon
```
(Colormap         :      Gdk.Color.Gdk_Colormap;
 Pixmap           :      Gdk.Pixmap.Gdk_Pixmap;
 Mask             :      Gdk.Bitmap.Gdk_Bitmap;
 Hot_X            :      Gint;
 Hot_Y            :      Gint);
```
Change the default drag icon. GtkAda retains a reference count for the arguments, and will release them when they are no longer needed. This procedure is deprecated.

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 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);
```
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.

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);
```

65 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.

65.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Drawing_Area         (Package Gtk.Drawing_Area)
```

65.2 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.

procedure Size
```
    (Darea               : access Gtk_Drawing_Area_Record;
     Width               :      Gint;
     Height              :      Gint);
```
Request a new size for the area.
This queues a resize request for the area. pragma Deprecated (Size);

66 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.

66.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Editable         (Package Gtk.Editable)
```

66.2 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.

66.3 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.

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);
```
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_Editable_Record)
 return Gint;
```
Return the position of the cursor.

procedure Set_Editable
```
(Widget             : access Gtk_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_Editable
```
(Editable           : access Gtk_Editable_Record)
 return Boolean;
```
Get the editable status of the entry.

67 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.

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_Event_Box (Package Gtk.Event_Box)
```

67.2 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.

68 Package Gtk.Extra

This is the top level package of the Gtk.Extra widget hierarchy.

69 Package Gtk.Extra.Border_Combo

A Gtk_Border_Combo is a special kind of combo box that allows the user to select the border to apply to cells in a spreadsheet. Its main usage seems to be with a Gtk_Sheet.

69.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Box           (Package Gtk.Box)
            \___ Gtk_Combo_Box  (Package Gtk.Extra.Combo_Box)
               \___ Gtk_Border_Combo (Package Gtk.Extra.Border_Combo)
```

69.2 Signals

- **"changed"**

  procedure Handler (Combo : access Gtk_Border_Combo_Record'Class;
  Selection : Gint);

  Emitted when a new font has been selected. Selection is the number of the selection font.

69.3 Subprograms

procedure Gtk_New
```
(Widget : out Gtk_Border_Combo);
```
Create a new border combo.
The button contains the currently selected border.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Border_Combo.

70 Package Gtk.Extra.Color_Combo

A Gtk_Color_Combo is a widget that ease the selection of colors by the user. It is a special form of a Gtk_Combo_Box, that displays a special popup window, with a list of colors.

Note that nothing appears in the button, this your responsibility to update it when the user selects a new color (see the "changed" signal).

The recommended solution is to put a Gtk_Pixmap as the child of the button of the combo box ("Add (Get_Button (Combo), Pixmap)"), and updated it in the handler for this signal.

70.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Widget                 (Package Gtk.Widget)
      \___ Gtk_Container           (Package Gtk.Container)
         \___ Gtk_Box              (Package Gtk.Box)
            \___ Gtk_Combo_Box (Package Gtk.Extra.Combo_Box)
               \___ Gtk_Color_Combo (Package Gtk.Extra.Color_Combo)
```

70.2 Signals

- **"changed"**

  procedure Handler (Color_Combo : access Gtk_Color_Combo_Record'Class;
  Selection : Gint;
  Color_Name : String);

  Emitted when the color has selected a new color. Selection is the number of the selection (this is the total row * Ncols + col). Color_Name is the name of the selected color.

70.3 Subprograms

procedure Gtk_New
```
(Widget             : out   Gtk_Color_Combo);
```
Create a new default combo box.
It shows a list of 40 default colors.

procedure Gtk_New
```
(Widget             : out   Gtk_Color_Combo;
 Nrows              :       Gint;
 Ncols              :       Gint;
 Values             :       Gdk.Color.Gdk_Color_Array);
```
Create a new combo box with a specific list of colors.
Note that Color_Names must contain at least Nrows * Ncols elements.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Color_Combo.

function Get_Color_At
```
(Widget          : access Gtk_Color_Combo_Record;
 Row             :        Gint;
 Col             :        Gint)
 return Gdk.Color.Gdk_Color;
```

Return the name of the color at specific coordinates.

procedure Find_Color
```
(Color_Combo     : access Gtk_Color_Combo_Record;
 Color           :        Gdk.Color.Gdk_Color;
 Row             : out    Gint;
 Col             : out    Gint);
```

Return the coordinates in which a color appear in the popup window.
(-1, -1) is returned if the color was not found in the combo box.

function Get_Selection
```
(Color_Combo     : access Gtk_Color_Combo_Record)
 return Gdk.Color.Gdk_Color;
```

Return the current selection in the combo.

function Set_Color
```
(Color_Combo     : access Gtk_Color_Combo_Record;
 Name            :        String)
 return Boolean;
```

Set the new current color. If the color is not found in the list of
colors provided in the popup window, False is returned.

function Set_Color
```
(Color_Combo     : access Gtk_Color_Combo_Record;
 Color           :        Gdk.Color.Gdk_Color)
 return Boolean;
```

Set the new current color. Color must have been allocated first. If the
color is not found in the list of colors provided in the popup window, False is returned.

function Get_Ncols
```
(Color_Combo     : access Gtk_Color_Combo_Record)
 return Gint;
```

Return the number of columns in the popup window

function Get_Nrows
```
(Color_Combo     : access Gtk_Color_Combo_Record)
```

```
         return Gint;
```

Return the number of rows in the popup window

```
procedure Changed
       (Color_Combo        : access Gtk_Color_Combo_Record;
        Row                :      Gint;
        Col                :      Gint);
```

Emit the changed signal for the widget, as if the color at coordinates
(Row, Col) had been selected. Note that this doesn't change the internal state of the widget (use Set_Color for that).

71 Package Gtk.Extra.Combo_Box

A Gtk_Combo_Box is a general form for a combo box (ie a button associated with a popup window to select its value). This widget should be used only if you intend to write your own kind of combo box. You should look at the following widgets for specific implementation: Gtk_Combo, Gtk_Color_Combo, Gtk_Border_Combo.

71.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Widget                 (Package Gtk.Widget)
      \___ Gtk_Container           (Package Gtk.Container)
         \___ Gtk_Box              (Package Gtk.Box)
            \___ Gtk_Combo_Box     (Package Gtk.Extra.Combo_Box)
```

71.2 Subprograms

procedure Gtk_New
```
    (Widget              : out   Gtk_Combo_Box);
```

Create a new combo box.
This creates all the internal subwidgets (the popup window,...) but this is your responsibility to put something inside the button or the popup window.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Combo_Box.

procedure Hide_Popdown_Window
```
    (Combobox            : access Gtk_Combo_Box_Record);
```

Hide the popup window, release the mouse grabs, and restore the default aspect for the arrow.

function Get_Button
```
    (Combobox            : access Gtk_Combo_Box_Record)
     return Gtk.Button.Gtk_Button;
```

Return the button that shows the value of the combo.

function Get_Arrow
```
    (Combobox            : access Gtk_Combo_Box_Record)
     return Gtk.Arrow.Gtk_Arrow;
```

Return the arrow button.
The user has to click on it to open the popup window.

function Get_Frame

```
(Combobox            : access Gtk_Combo_Box_Record)
 return Gtk.Frame.Gtk_Frame;
```

The frame displayed in the popup window.
You should add whatever value the popup window should display in it.

72 Package Gtk.Extra.Font_Combo

A Gtk_Font_Combo is a small toolbar used to select fonts. This widget takes less real-estate on the screen than a Gtk_Font_Selection widget, and thus can be kept permanently on the screen. This widget only works with postscript fonts (see Gtk.Extra.PsFont).

72.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Toolbar     (Package Gtk.Toolbar)
            \___ Gtk_Font_Combo (Package Gtk.Extra.Font_Combo)
```

72.2 Signals

- **"changed"**
  procedure Handler (Combo : access Gtk_Font_Combo_Record'Class);

  Emitted when a new font was selected by the user.

72.3 Subprograms

procedure Gtk_New
```
    (Widget              : out   Gtk_Font_Combo);
```
Create a new combo box

function Get_Type **return** Guint;

Return the internal value associated with a Gtk_Font_Combo.

procedure Font_Combo_Select
```
    (Font_Combo          : access Gtk_Font_Combo_Record;
     Family              : in    String;
     Bold                : in    Boolean;
     Italic              : in    Boolean;
     Height              : in    Gint);
```
Selects a new font
Family is the name of the postscript font.

procedure Font_Combo_Select_Nth
```
    (Font_Combo          : access Gtk_Font_Combo_Record;
     N                   : in    Gint;
     Bold                : in    Boolean;
     Italic              : in    Boolean;
```

```
    Height               : in    Gint);
```

Selects the nth font in the combo box.

```
function Get_Font_Height
   (Font_Combo          : access Gtk_Font_Combo_Record)
    return Glib.Gint;
```

Return the height of the selected font

```
function Get_Font
   (Font_Combo          : access Gtk_Font_Combo_Record)
    return Gdk.Font.Gdk_Font;
```

Return the selected font.

```
function Get_Font_Description
   (Font_Combo          : access Gtk_Font_Combo_Record)
    return Pango.Font.Pango_Font_Description;
```

Return the selected font.

73 Package Gtk.Extra.Item_Entry

A Gtk_Item_Entry is a special kind of entry item used in a Gtk_Sheet to edit the current cell. It can be used independently, but you should rather use a more general Gtk_Entry widget.

73.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Editable           (Package Gtk.Editable)
         \___ Gtk_Entry           (Package Gtk.GEntry)
            \___ Gtk_IEntry       (Package Gtk.Extra.Item_Entry)
```

73.2 Types

subtype Gtk_Item_Entry **is** Gtk_IEntry;

This type is provided so as to be compatible with the new name used in gtk+extra. Both names are kept for backward compatibility.

73.3 Subprograms

procedure Gtk_New
```
(Widget            : out   Gtk_IEntry;
 Max               : in    Guint16 := 0);
```

Create a new entry item.
By default, the maximal length depends only on the size of the widget.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Item_Entry.

procedure Set_Justification
```
(Item_Entry        : access Gtk_IEntry_Record;
 Justification     : in    Gtk.Enums.Gtk_Justification);
```

Change the justification of the text in the entry.

procedure Set_Text
```
(Item_Entry        : access Gtk_IEntry_Record;
 Text              : in    String;
 Justification     : in    Gtk.Enums.Gtk_Justification);
```

Change the text in the entry.

procedure Set_Cursor_Visible
```
(Item_Entry          : access Gtk_IEntry_Record;
 Visible             :        Boolean);
```
Whether the cursor should be visible

function Get_Cursor_Visible
```
(Item_Entry          : access Gtk_IEntry_Record)
 return Boolean;
```
Whether the cursor is visible

74 Package Gtk.Extra.Plot

This package implements a high-level, general purpose plotting widget. You can display any set of data (set of points, curve defined by a parametric function, ...). This widget can automatically display them as a curve, along with labelled axis, axis tic marks, legends,...

This is the base class, that provides 2D graphics. Some children provide polar-coordinates and 3D graphics in addition.

It fully supports the drag-and-drop protocol for all of its children, which means that the user can interactively move them in the Gtk_Plot area.

A Gtk_Plot is closely associated with a Gdk_Drawable, on which all the drawings are done. It can be done anywhere within that drawable, its "position" is indicated by a tuple (X, Y), which are two values between 0.0 and 1.0 (from left to right, or from top to bottom). Its size is also given as a ratio other the drawable's size.

Most points in the plot have also this relative coordinates systems, which makes it really easy to handle resizing of a plot window.

See the package Gtk.Extra.Plot_Ps for a way to easily print a Gtk_Plot to a postscript file.

In this package, font parameters are sometimes required. Here is the list of possible fonts used by Gtk.Extra:

- "Times-Roman",
- "Times-Italic",
- "Times-Bold",
- "Times-BoldItalic",
- "AvantGarde-Book",
- "AvantGarde-BookOblique",
- "AvantGarde-Demi",
- "AvantGarde-DemiOblique",
- "Bookman-Light",
- "Bookman-LightItalic",
- "Bookman-Demi",
- "Bookman-DemiItalic",
- "Courier",
- "Courier-Oblique",
- "Courier-Bold",
- "Courier-BoldOblique",
- "Helvetica",
- "Helvetica-Oblique",
- "Helvetica-Bold",
- "Helvetica-BoldOblique",
- "Helvetica-Narrow",

- "Helvetica-Narrow-Oblique",
- "Helvetica-Narrow-Bold",
- "Helvetica-Narrow-BoldOblique",
- "NewCenturySchoolbook-Roman",
- "NewCenturySchoolbook-Italic",
- "NewCenturySchoolbook-Bold",
- "NewCenturySchoolbook-BoldItalic",
- "Palatino-Roman",
- "Palatino-Italic",
- "Palatino-Bold",
- "Palatino-BoldItalic",
- "Symbol",
- "ZapfChancery-MediumItalic",
- "ZapfDingbats",

74.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Plot           (Package Gtk.Extra.Plot)
```

74.2 Signals

- **"changed"**

  procedure Handler (Plot : access Gtk_Plot_Record'Class);

  Called every time some property of the widget is changed, or the widget is moved or resized.
- **"moved"**

  function Handler (Plot : access Gtk_Plot_Record'Class;
  X : Gdouble;
  Y : Gdouble)
  return Boolean;

  Called when the widget has been moved relative to its drawable. Its new position is given in parameters.
- **"resized"**

  function Handler (Plot : access Gtk_Plot_Record'Class;
  Width : Gdouble;
  Height : Gdouble)

return Boolean;

Called when the widget has been resized relative to its drawable. Its new size is given in parameters.

- **"tick_label"**

  function Handler (Axis : access Gtk_Plot_Axis_Record'Class;
  Tick : Gdouble_Access;
  Label : Interfaces.C.Strings.chars_ptr)
  return Boolean;

  Called when a label should be drawn. You can modify the contents of Label (up to 100 characters) a

74.3 Types

type Gtk_Plot_Line **is new** Gdk.C_Proxy;

A simple line drawn on the plot.

type Gtk_Plot_Text **is new** Gdk.C_Proxy;

A text that can be displayed anywhere on the plot.

type Plot3D_Function **is access function**

```
(Plot  : System.Address;
```

type Plot_Angle **is**

```
(Angle_0, Angle_90, Angle_180, Angle_270);
```

Valid values for the angles of texts and titles. pragma Convention (C, Plot_Angle);

type Plot_Axis_Pos **is**

```
(Axis_Left, Axis_Right, Axis_Top, Axis_Bottom);
```

Where the axis should be put pragma Convention (C, Plot_Axis_Pos);

type Plot_Border_Style **is**

```
(Border_None,
    -- No border is drawn

 Border_Line,
    -- A simple line on each side

 Border_Shadow
```

```
          -- The right and bottom lines are
          -- thicker
      );
```

Border types used for legends. pragma Convention (C, Plot_Border_Style);

```
type Plot_Error is
      (Error_Div_Zero, Error_Log_Neg);
```

Errors that can be encountered while calculating a graph. pragma Convention (C, Plot_Error);

```
type Plot_Label_Pos is new Integer;
```

Position of labels along an axis.

```
subtype Plot_Label_Style is Gtk.Extra.Plot_Data.Plot_Label_Style;
```

```
type Plot_Orientation is
      (Axis_X, Axis_Y, Axis_Z);
```

How to reference axis in 3D plots pragma Convention (C, Plot_Orientation);

```
subtype Plot_Scale is Gtk.Extra.Plot_Data.Plot_Scale;
```

```
type Plot_Ticks_Pos is new Integer;
```

The position and orientation of the ticks along an axis. See the constants below for the possible values. Note also that not all the values are valid with all types of axis.

74.4 Subprograms

74.4.1 Creating a plot

```
procedure Gtk_New
   (Plot                : out   Gtk_Plot;
    Drawable            : in    Gdk.Drawable.Gdk_Drawable
                         := Gdk.Drawable.Null_Drawable);
```

Create a new plot, that will be displayed in Drawable.
All the dataset, labels, axis,... associated with the plot will be drawn in that drawable, which must have been created beforehand. Note that the drawable can also be set later with Set_Drawable.

```
procedure Gtk_New
       (Plot                : out  Gtk_Plot;
        Width               : in   Gdouble;
        Height              : in   Gdouble;
        Drawable            : in   Gdk.Drawable.Gdk_Drawable
                             := Gdk.Drawable.Null_Drawable);
```

Create a new plot with a specific size.

```
function Get_Type            return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Plot.

```
procedure Set_Drawable
       (Plot                : access Gtk_Plot_Record;
        Drawable            : in   Gdk.Drawable.Gdk_Drawable);
```

Modify the drawable on which the graphs are displayed.
From now on, all the drawings will be done on that drawable. Note that they are not automatically copied to the new Drawable until the Plot needs to be redrawn.

```
function Get_Drawable
       (Plot                : access Gtk_Plot_Record)
        return Gdk.Drawable.Gdk_Drawable;
```

Return the drawable on which the graphs are plotted.

```
procedure Set_Background
       (Plot                : access Gtk_Plot_Record;
        Background          : in   Gdk.Color.Gdk_Color);
```

Change the background for the plot.
Note that this has no effect if the plot has been set to transparent (see the flags below). The Plot is also redrawn as soon as you modify this color.

```
procedure Set_Background_Pixmap
       (Plot                : access Gtk_Plot_Record;
        Pixmap              :      Gdk.Pixmap.Gdk_Pixmap);
```

Specificy a background pixmap to use for the plot

```
procedure Set_Transparent
       (Plot                : access Gtk_Plot_Record;
        Transparent         :      Boolean);
```

Whether the plot is transparent. If Transparent is True, all background
attributes are ignored (pixmap, color,...)

```
function Is_Transparent
       (Plot                : access Gtk_Plot_Record)
        return Boolean;
```

Whether the plot is current transparent

procedure Paint
```
(Plot              : access Gtk_Plot_Record);
```

Force an immediate repaint of the widget in its pixmap.
The modification won't appear on the screen until you call Refresh. It is probably not a good idea to call this function directly, and it is more efficient to queue a draw request (see the Gtk.Widget package for related functions).

procedure Refresh
```
(Plot              : access Gtk_Plot_Record;
 Area              : in    Gdk.Rectangle.Gdk_Rectangle);
```

Copy the plot's pixmap to the screen.
The same comment as for Paint applies here, and you probably don't have to call this function yourself, since queuing a draw request is more efficient.

procedure Set_Magnification
```
(Plot              : access Gtk_Plot_Record;
 Magnification     :        Gdouble);
```

Change the magnification level of the plot.
1.0 is the default magnification, higher values will zoom in while lower values will zoom out.

procedure Draw_Line
```
(Plot              : access Gtk_Plot_Record;
 Line              :        Gtk_Plot_Line;
 X1, Y1, X2, Y2    :        Gdouble);
```

Draw a line on the plot

74.4.2 Coordinates and sizes

procedure Get_Position
```
(Plot              : access Gtk_Plot_Record;
 X                 : out    Gdouble;
 Y                 : out    Gdouble);
```

Return the position of the Plot within its drawable.
X and Y are in the range 0.0 .. 1.0, where (0.0, 0.0) is the top-left corner and (1.0, 1.0) the bottom-right corner. The position can be modified by Move below.

procedure Get_Size
```
(Plot              : access Gtk_Plot_Record;
 Width             : out    Gdouble;
 Height            : out    Gdouble);
```

Return the size of the Plot.
Width and Height are both in the range 0.0 .. 1.0, where 1.0 means they occupy all the space available in the Drawable, 0.5 means they only occupy half of it.

function Get_Internal_Allocation
```
       (Plot                : access Gtk_Plot_Record)
        return Gtk.Widget.Gtk_Allocation;
```

Return the real position/size of the plot inside its parent container.
You should use this function instead of converting yourself the result of Get_Position and Get_Size.

procedure Move
```
       (Plot                : access Gtk_Plot_Record;
        X                   : in    Gdouble;
        Y                   : in    Gdouble);
```

Move the plot widget inside its drawable.
X and Y should both be in the range 0.0 .. 1.0 (from top-left corner to bottom-right corner).

procedure Resize
```
       (Plot                : access Gtk_Plot_Record;
        Width               : in    Gdouble;
        Height              : in    Gdouble);
```

Resize the widget.
Width and Height should both be in the range 0.0 .. 1.0, this indicates which ratio of the drawable's screen real-estate they should use.

procedure Move_Resize
```
       (Plot                : access Gtk_Plot_Record;
        X                   : in    Gdouble;
        Y                   : in    Gdouble;
        Width               : in    Gdouble;
        Height              : in    Gdouble);
```

Move and resize the widget in a single operation.
This is faster than doing each operation separately.

procedure Get_Pixel
```
       (Plot                : access Gtk_Plot_Record;
        Xx                  : in    Gdouble;
        Yy                  : in    Gdouble;
        X                   : out   Gdouble;
        Y                   : out   Gdouble);
```

Get the screen coordinate (relative to Plot's parent) of a point.
The initial coordinates (Xx, Yy) should be in the range 0.0 .. 1.0.

procedure Clip_Data

```
(Plot                : access Gtk_Plot_Record;
 Clip                :      Boolean);
```

If Clip is True, any drawing of a Gtk_Plot_Data will be limited to the area occupied by Plot. Otherwise, it might draw outside of Plot.

procedure Get_Point
```
(Plot                : access Gtk_Plot_Record;
 X                   : in    Gint;
 Y                   : in    Gint;
 Xx                  : out   Gdouble;
 Yy                  : out   Gdouble);
```

Convert from an absolute screen coordinate to a relative one.
(X, Y) should be relative to Plot's parent. This function is the opposite of Get_Pixel.

procedure Set_Xrange
```
(Plot                : access Gtk_Plot_Record;
 Xmin                : in    Gdouble := 0.0;
 Xmax                : in    Gdouble := 1.0);
```

Set the range of visible points for this plot.
Only the points of the graph those coordinates are in the range Xmin .. Xmax will be visible.

procedure Set_Yrange
```
(Plot                : access Gtk_Plot_Record;
 Ymin                : in    Gdouble := 0.0;
 Ymax                : in    Gdouble := 1.0);
```

Set the range of visible points for this plot.
Only the points of the graph those coordinates are in the range Xmin .. Xmax will be visible.

procedure Set_Range
```
(Plot                : access Gtk_Plot_Record;
 Xmin                : in    Gdouble := 0.0;
 Xmax                : in    Gdouble := 1.0;
 Ymin                : in    Gdouble := 0.0;
 Ymax                : in    Gdouble := 1.0);
```

Set both ranges at the same time

procedure Autoscale
```
(Plot                : access Gtk_Plot_Record);
```

Calculate automically the appropriate ranges for the plot.

procedure Get_Xrange
```
(Plot                : access Gtk_Plot_Record;
 Xmin                : out   Gdouble;
```

```
         Xmax                : out   Gdouble);
```

Get the current range for the X axis.

```
procedure Get_Yrange
        (Plot                : access Gtk_Plot_Record;
         Ymin                : out   Gdouble;
         Ymax                : out   Gdouble);
```

Get the current range for the X axis.

```
procedure Set_Xscale
        (Plot                : access Gtk_Plot_Record;
         Scale_Type          : in    Plot_Scale);
```

Set the type of the X axis (logarithmic, linear, ...).

```
procedure Set_Yscale
        (Plot                : access Gtk_Plot_Record;
         Scale_Type          : in    Plot_Scale);
```

Set the type of the Y axis (logarithmic, linear, ...).

```
function Get_Xscale
        (Plot                : access Gtk_Plot_Record)
         return Plot_Scale;
```

Get the type of the X axis.

```
function Get_Yscale
        (Plot                : access Gtk_Plot_Record)
         return Plot_Scale;
```

Get the type of the Y axis.

```
procedure Reflect_X
        (Plot                : access Gtk_Plot_Record;
         Reflect             :        Boolean);
```

Reverse the direction of the X axis

```
function Is_X_Reflected
        (Plot                : access Gtk_Plot_Record)
         return Boolean;
```

Whether the X axis is currently reflected

```
procedure Reflect_Y
        (Plot                : access Gtk_Plot_Record;
         Reflect             :        Boolean);
```

Reverse the direction of the Y axis

```
function Is_Y_Reflected
       (Plot                : access Gtk_Plot_Record)
        return Boolean;
```

Whether the Y axis is currently reflected

74.4.3 Axis

A Gtk_Plot has four axis, one one each of its sides. These axis can have ticks, labels for ticks, titles, ... associated with them.

```
procedure Gtk_New
       (Axis                : out   Gtk_Plot_Axis;
        Orientation         :      Plot_Orientation);
```

Create a new axis

```
function Axis_Get_Type          return Gtk_Type;
```

Return the internal value associated with a Gtk_Plot_Axis.

```
function Get_Axis
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos)
        return Gtk_Plot_Axis;
```

Get a pointer to an axis.

```
procedure Axis_Set_Visible
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Visible             : in    Boolean);
```

Indicate whether the axis should be visible or not.

```
function Axis_Visible
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos)
        return Boolean;
```

Return the visibility state of the axis

```
procedure Axis_Set_Title
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Title               : in    String);
```

Modify the title of the axis.
Each axis has a title that is displayed along its line (vertically for the left and right sides).

```
procedure Axis_Show_Title
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos);
```

Show the title associated with the axis.

```
procedure Axis_Hide_Title
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos);
```

Hide the title associated with the axis.

```
procedure Axis_Move_Title
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Angle               : in    Plot_Angle;
        X                   : in    Gdouble;
        Y                   : in    Gdouble);
```

Modify the position and orientation of the axis' title.
X and Y indicate a position relative to the location of the axis (0.0 to display it to the left (resp. top) of the axis, 1.0 to display it to the right (resp. bottom) of the axis.

```
procedure Axis_Justify_Title
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Justification       : in    Gtk.Enums.Gtk_Justification);
```

Modify the justification for the axis.

```
procedure Axis_Set_Attributes
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Line_Width          : in    Gfloat;
        Color               : in    Gdk.Color.Gdk_Color);
```

Modify the attributes of the lines of the axis.

```
procedure Axis_Get_Attributes
       (Plot                : access Gtk_Plot_Record;
        Axis                : in    Plot_Axis_Pos;
        Line_Width          : out   Gfloat;
        Color               : out   Gdk.Color.Gdk_Color);
```

Get the attributes of the axis.

```
procedure Axis_Set_Ticks
       (Plot                : access Gtk_Plot_Record;
        Orientation         : in    Plot_Orientation;
        Major_Step          : in    Gdouble;
        Num_Minor           : in    Gint);
```

Set up ticks for a specific orientation.
A horizontal orientation will match the left and right sides, whereas a vertical orientation will match the top and bottom sides. Major_Step is a percentage value of the widget size, and indicate the step between each big ticks. For instance, if Major_Step has a value of 0.2, there will be 5 big ticks drawn along the axis. Num_Minor is the number of minor ticks between each major one.

procedure Axis_Set_Major_Ticks
```
(Plot            : access Gtk_Plot_Record;
 Orientation     : in     Plot_Orientation;
 Major_Step      : in     Gdouble);
```
Modify the step for major ticks.
This is a percentage value that indicates how many major ticks are drawn along the axis. See also Axis_Set_Ticks.

procedure Axis_Set_Minor_Ticks
```
(Plot            : access Gtk_Plot_Record;
 Orientation     : in     Plot_Orientation;
 Num_Minor       : in     Gint);
```
Modify the number of minor ticks between each major one.
See also Axis_Set_Ticks.

procedure Axis_Set_Ticks_Length
```
(Plot            : access Gtk_Plot_Record;
 Axis            : in     Plot_Axis_Pos;
 Length          : in     Gint);
```
Set the length (in pixels) of the big ticks.
The small ticks will have half this length.

procedure Axis_Set_Ticks_Width
```
(Plot            : access Gtk_Plot_Record;
 Axis            : in     Plot_Axis_Pos;
 Width           : in     Gfloat);
```
Set the width (in pixels) of the ticks.
This width is common to both the long and short ticks.

procedure Axis_Show_Ticks
```
(Plot            : access Gtk_Plot_Record;
 Axis            : in     Plot_Axis_Pos;
 Major_Mask      : in     Plot_Ticks_Pos;
 Minor_Mask      : in     Plot_Ticks_Pos);
```
Set the style of the ticks.

procedure Axis_Set_Ticks_Limits
```
(Plot            : access Gtk_Plot_Record;
```

```
        Orientation        : in    Plot_Orientation;
        Ticks_Beg          : in    Gdouble;
        Ticks_End          : in    Gdouble);
```

Indicate the area of the axis that should have ticks.
Ticks will be displayed only from Ticks_Beg to Ticks_End.

```
procedure Axis_Unset_Ticks_Limits
       (Plot               : access Gtk_Plot_Record;
        Orientation        : in    Plot_Orientation);
```

Cancel the ticks limits set by a previous call to
Axis_Set_Ticks_Limits.

```
procedure Axis_Set_Break
       (Plot               : access Gtk_Plot_Record;
        Orient             :       Plot_Orientation;
        Min, Max           :       Gdouble;
        Step_After         :       Gdouble;
        Nminor_After       :       Gint;
        Scale_After        :       Plot_Scale);
```

???

```
procedure Axis_Remove_Break
       (Plot               : access Gtk_Plot_Record;
        Orient             :       Plot_Orientation);
```

???

```
procedure Axis_Show_Labels
       (Plot               : access Gtk_Plot_Record;
        Axis               : in    Plot_Axis_Pos;
        Labels_Mask        : in    Plot_Label_Pos);
```

Indicate whether a label should be drawn at each ticks to indicate
its value. Not all values of Labels_Mask are relevant for all axis. For instance, for a vertical axis, the relevant values are Axis_Right and Axis_Left.

```
procedure Axis_Title_Set_Attributes
       (Plot               : access Gtk_Plot_Record;
        Axis               : in    Plot_Axis_Pos;
        Font               : in    String;
        Height             : in    Gint;
        Angle              : in    Plot_Angle;
        Foreground         : in    Gdk.Color.Gdk_Color;
        Background         : in    Gdk.Color.Gdk_Color;
        Transparent        : in    Boolean;
        Justification      : in    Gtk.Enums.Gtk_Justification);
```

Set the attributes to be used for the title of the axis.
Font is a postscript font name (as listed in the beginning of this package).

procedure Axis_Set_Labels_Attributes
```
   (Plot              : access Gtk_Plot_Record;
    Axis              : in     Plot_Axis_Pos;
    Font              : in     String;
    Height            : in     Gint;
    Angle             : in     Plot_Angle;
    Foreground        : in     Gdk.Color.Gdk_Color;
    Background        : in     Gdk.Color.Gdk_Color;
    Transparent       : in     Boolean;
    Justification     : in     Gtk.Enums.Gtk_Justification);
```
Set the attributes to be used for the ticks labels.

procedure Axis_Use_Custom_Tick_Labels
```
   (Plot              : access Gtk_Plot_Record;
    Axis              : in     Plot_Axis_Pos;
    Custom            : in     Boolean := True);
```
Indicate which kind of labels should be used for major ticks.
If Custom is True, then the labels set by Axis_Set_Tick_Labels will be used.

procedure Axis_Set_Labels_Offset
```
   (Plot              : access Gtk_Plot_Record;
    Axis              :        Plot_Axis_Pos;
    Offset            :        Gint);
```
Set the distance between the axis and its labels

function Axis_Get_Labels_Offset
```
   (Plot              : access Gtk_Plot_Record;
    Axis              :        Plot_Axis_Pos)
    return Gint;
```
Get the distance between the axis and its labels.

procedure Axis_Set_Labels_Style
```
   (Plot              : access Gtk_Plot_Record;
    Axis              :        Plot_Axis_Pos;
    Style             :        Plot_Label_Style;
    Precision         :        Gint);
```
Set the style of labels.
This indicates whether the labels should be displayed as floating point values or in the scientific notation. Precision is the number of digits to be printed.

procedure Axis_Set_Labels_Suffix
```
   (Plot              : access Gtk_Plot_Record;
```

```
 Axis               :      Plot_Axis_Pos;
 Text               :      String);
```

Defines a suffix to add after each label on the axis

```
procedure Axis_Set_Labels_Prefix
   (Plot               : access Gtk_Plot_Record;
    Axis               :      Plot_Axis_Pos;
    Text               :      String);
```

Defines a prefix to add before each label on the axis

```
function Axis_Get_Labels_Suffix
   (Plot               : access Gtk_Plot_Record;
    Axis               :      Plot_Axis_Pos)
    return String;
```

Return the suffix added to each label.

```
function Axis_Get_Labels_Prefix
   (Plot               : access Gtk_Plot_Record;
    Axis               :      Plot_Axis_Pos)
    return String;
```

Return the prefix added to each label.

74.4.4 Grids

A grid can be displayed in the graph.
This makes it easier to understand a graphics in some situations. The grid has two simultaneous line styles, each with its own specific step (minor and major steps).

There are two special lines in the grid, that you can display even if you don't display the rest of the line. These are the origin of the coordinates system, ie the lines at X=0 and Y=0.

```
procedure X0_Set_Visible
   (Plot               : access Gtk_Plot_Record;
    Visible            : in    Boolean);
```

Indicate whether the line at X=0 should be displayed.

```
function X0_Visible
   (Plot               : access Gtk_Plot_Record)
    return Boolean;
```

Return the visibility state of the line at X=0

```
procedure Y0_Set_Visible
   (Plot               : access Gtk_Plot_Record;
    Visible            : in    Boolean);
```

Indicate whether the line at Y=0 should be displayed.

```
function Y0_Visible
   (Plot                : access Gtk_Plot_Record)
    return Boolean;
```

Return the visibility state of the line at Y=0

```
procedure X0line_Set_Attributes
   (Plot                : access Gtk_Plot_Record;
    Style               : in    Plot_Line_Style;
    Width               : in    Gfloat;
    Color               : in    Gdk.Color.Gdk_Color);
```

Set the attributes of the line at X=0

```
procedure Y0line_Set_Attributes
   (Plot                : access Gtk_Plot_Record;
    Style               : in    Plot_Line_Style;
    Width               : in    Gfloat;
    Color               : in    Gdk.Color.Gdk_Color);
```

Set the attributes of the line at Y=0

```
procedure Grids_Set_On_Top
   (Plot                : access Gtk_Plot_Record;
    On_Top              :       Boolean);
```

Whether the grid should be displayed on top of the plots

```
function Grids_On_Top
   (Plot                : access Gtk_Plot_Record)
    return Boolean;
```

Whether the gris is currently displayed on top of the plots

```
procedure Grids_Set_Visible
   (Plot                : access Gtk_Plot_Record;
    Vmajor              : in    Boolean;
    Vminor              : in    Boolean;
    Hmajor              : in    Boolean;
    Hminor              : in    Boolean);
```

Indicate whether the lines of the grids should be displayed.
You can decide separately whether the major and minor lines should be displayed.

```
procedure Grids_Visible
   (Plot                : access Gtk_Plot_Record;
    Vmajor              : out   Boolean;
    Vminor              : out   Boolean;
    Hmajor              : out   Boolean;
```

```
          Hminor              : out   Boolean);
```

Return the visibility state of the grid.

```
procedure Major_Hgrid_Set_Attributes
        (Plot                 : access Gtk_Plot_Record;
         Style                : in    Plot_Line_Style;
         Width                : in    Gfloat;
         Color                : in    Gdk.Color.Gdk_Color);
```

Set the attributes for the major horizontal lines in the grid.

```
procedure Major_Vgrid_Set_Attributes
        (Plot                 : access Gtk_Plot_Record;
         Style                : in    Plot_Line_Style;
         Width                : in    Gfloat;
         Color                : in    Gdk.Color.Gdk_Color);
```

Set the attributes for the major vertical lines in the grid.

```
procedure Minor_Hgrid_Set_Attributes
        (Plot                 : access Gtk_Plot_Record;
         Style                : in    Plot_Line_Style;
         Width                : in    Gfloat;
         Color                : in    Gdk.Color.Gdk_Color);
```

Set the attributes for the minor horizontal lines in the grid.

```
procedure Minor_Vgrid_Set_Attributes
        (Plot                 : access Gtk_Plot_Record;
         Style                : in    Plot_Line_Style;
         Width                : in    Gfloat;
         Color                : in    Gdk.Color.Gdk_Color);
```

Set the attributes for the minor vertical lines in the grid.

74.4.5 Legends

Each graph is associated with one legend, that is supposed to indicate what the plot represents.

```
procedure Show_Legends
        (Plot                 : access Gtk_Plot_Record);
```

Indicate that the legend should be displayed.

```
procedure Hide_Legends
        (Plot                 : access Gtk_Plot_Record);
```

Indicate that the legend should not be displayed.

procedure Set_Legends_Border
```
(Plot               : access Gtk_Plot_Record;
 Border             :        Plot_Border_Style;
 Shadow_Width       :        Gint);
```
Modify the way the borders of the legend look like.

procedure Legends_Move
```
(Plot               : access Gtk_Plot_Record;
 X                  : in     Gdouble;
 Y                  : in     Gdouble);
```
Move the legend relative to the widget's area.
X and Y are percentage values. (0.0, 0.0) indicates the top-left corner of the plot, (1.0, 1.0) indicates the bottom-right corner.

procedure Legends_Get_Position
```
(Plot               : access Gtk_Plot_Record;
 X                  : out    Gdouble;
 Y                  : out    Gdouble);
```
Return the current position of the legend.

function Legends_Get_Allocation
```
(Plot               : access Gtk_Plot_Record)
 return Gtk.Widget.Gtk_Allocation;
```
Return the exact coordinates and size in pixels of the legend.
The coordinates are relative to the widget's parent container.

procedure Legends_Set_Attributes
```
(Plot               : access Gtk_Plot_Record;
 Ps_Font            : in     String;
 Height             : in     Gint;
 Foreground         : in     Gdk.Color.Gdk_Color;
 Background         : in     Gdk.Color.Gdk_Color);
```
Set the attributes to use when displaying the legend.

74.4.6 Line

procedure Set_Line_Style
```
(Line               :        Gtk_Plot_Line;
 Style              :        Plot_Line_Style);
```
Change the style of the line.

procedure Set_Line_Width
```
(Line               :        Gtk_Plot_Line;
 Width              :        Gfloat);
```

Change the width of the line.

procedure Set_Color
```
(Line                   :      Gtk_Plot_Line;
 Color                  :      Gdk.Color.Gdk_Color);
```
Change the color used to draw the line

74.4.7 Text

procedure Put_Text
```
(Plot                   : access Gtk_Plot_Record;
 X                      : in    Gdouble;
 Y                      : in    Gdouble;
 Font                   : in    String := "";
 Font_Height            : in    Gint := 10;
 Angle                  : in    Plot_Angle;
 Foreground             : in    Gdk.Color.Gdk_Color
                         := Gdk.Color.Null_Color;
 Background             : in    Gdk.Color.Gdk_Color
                         := Gdk.Color.Null_Color;
 Transparent            : in    Boolean := False;
 Justification          : in    Gtk.Enums.Gtk_Justification
                         := Gtk.Enums.Justify_Center;
 Text                   : in    String := "");
```
Print some text in Plot.
The text will be drawn at the relative coordinates (X, Y), with a specified Angle. If Font is the empty string, a default font and default Font_Height will be used. Likewise, default colors will be used if you don't specify any. Font should be the name of a postscript font, the list of which can be found in Gtk.Plot.Psfont. If Transparent is True, then no background will be drawn for the text.

procedure Remove_Text
```
(Plot                   : access Gtk_Plot_Record;
 Text                   : in    Gtk_Plot_Text);
```
Remove some text that is currently visible on the plot.
Nothing is done if Text is currently not visible.

procedure Text_Get_Area
```
(Text                   : in    Gtk_Plot_Text;
 Angle                  : in    Gint;
 Just                   : in    Gtk.Enums.Gtk_Justification;
 Font_Name              : in    String;
 Font_Size              : in    Gint;
 X                      : out   Gint;
 Y                      : out   Gint;
 Width                  : out   Gint;
```

```
     Height              : out   Gint);
```

Return the area currently occupied by a text.
The coordinates are relative to the top-left corner of the plot in which the text was put.

```
procedure Text_Get_Size
    (Text                : in    Gtk_Plot_Text;
     Angle               : in    Gint;
     Font_Name           : in    String;
     Font_Size           : in    Gint;
     Width               : out   Gint;
     Height              : out   Gint;
     Ascent              : out   Gint;
     Descent             : out   Gint);
```

Return the size in pixels occupied by a text in the plot.
see Chapter 78 [Package_Gtk.Extra.Plot_Canvas], page 309 for a function that returns a Gtk_Plot_Text.

```
function Get_Texts
    (Plot                : access Gtk_Plot_Record)
     return Texts_List.Glist;
```

Return the list of all the texts associated with the plot

```
procedure Get_Text_Position
    (Text                : in    Gtk_Plot_Text;
     X                   : out   Gdouble;
     Y                   : out   Gdouble);
```

Return the location of the Text.

```
function Get_Text_String
    (Text                : in    Gtk_Plot_Text)
     return String;
```

Return the string of the text.

```
procedure Text_Set_Attributes
    (Text                : in    Gtk_Plot_Text;
     Font                : in    String;
     Height              : in    Gint;
     Angle               : in    Plot_Angle;
     Fg                  : in    Gdk.Color.Gdk_Color;
     Bg                  : in    Gdk.Color.Gdk_Color;
     Transparent         : in    Boolean := False;
     Justification       : in    Gtk.Enums.Gtk_Justification
                          := Gtk.Enums.Justify_Center;
     Str                 : in    String := "");
```

Change the attributes of Text.

```
procedure Draw_Text
       (Plot               : access Gtk_Plot_Record;
        Text               :        Gtk_Plot_Text);
```

Draw the text

```
procedure Text_Set_Border
       (Text               :        Gtk_Plot_Text;
        Border             :        Plot_Border_Style;
        Border_Space       :        Gint;
        Border_Width       :        Gint;
        Shadow_Width       :        Gint);
```

Set the border attributes for the text

74.4.8 Datasets

A dataset is a set of points, either given explicitly by your application or calculated with a specific function, and that can be plotted on the screen. In Gtk_Plot, such a set is represented with symbols (special points in the graph, that can be manipulated interactively if you so wish), linked by connectors, which are either straight lines, splines, sets, ... Multiple data sets can of course be printed on a single graph.

```
procedure Add_Data
       (Plot               : access Gtk_Plot_Record;
        Data               : access Gtk_Plot_Data_Record'Class);
```

Add an existing set of data to the plot.
This set will automatically be drawn the next time the Plot itself is drawn.

```
function Remove_Data
       (Plot               : access Gtk_Plot_Record;
        Data               : access Gtk_Plot_Data_Record'Class)
        return Boolean;
```

Remove the dataset from Plot.
This function returns True if the dataset was indeed found and could be removed, False otherwise.

```
function Add_Function
       (Plot               : access Gtk_Plot_Record;
        Func               : in    Plot_Function)
        return Gtk_Plot_Data;
```

Allocate a new dataset, whose point are automatically calculated.
Func is a function that takes the X coordinate value, and should return the Y coordinate value. The newly allocated set should be freed by calling Free above. The set is automatically added to the plot, so you don't need to explicitly call Add_Dataset.

75 Package Gtk.Extra.Plot_3D

A special plot that draws its data in three dimension. The data associated with such plots should either be a function or a Gtk.Extra.Plot_Surface.

75.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Widget                 (Package Gtk.Widget)
      \___ Gtk_Plot                (Package Gtk.Extra.Plot)
         \___ Gtk_Plot_3D          (Package Gtk.Extra.Plot_3D)
```

75.2 Types

type Plot_Plane **is new** Integer;

type Plot_Side **is mod** 2 ** 32;

type Plot_Vector **is record**

```
X, Y, Z : Gdouble;
end record;
```

75.3 Subprograms

procedure Gtk_New

```
(Widget            : out   Gtk_Plot_3D;
 Drawable          :      Gdk.Drawable.Gdk_Drawable;
 Width, Height     :      Gdouble := 0.0);
```

Create a new 3D plot.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with this widget.

procedure Autoscale

```
(Plot              : access Gtk_Plot_3D_Record);
```

Chooses the best ranges for all coordinates depending on the data sets
put in Plot. Note that only data sets whose points you have set explicitely are taken into account, not the ones based on functions.

```
function Get_Xfactor
   (Plot              : access Gtk_Plot_3D_Record)
    return Gdouble;
```

```
function Get_Zfactor
   (Plot              : access Gtk_Plot_3D_Record)
    return Gdouble;
```

Get the scaling factor along each of the coordinates.

```
procedure Set_Xfactor
   (Plot              : access Gtk_Plot_3D_Record;
    Xfactor           :        Gdouble);
```

```
procedure Set_Zfactor
   (Plot              : access Gtk_Plot_3D_Record;
    Zfactor           :        Gdouble);
```

Set the scaling factor along each of the coordinates

```
procedure Set_Xrange
   (Plot              : access Gtk_Plot_3D_Record;
    Min, Max          :        Gdouble);
```

```
procedure Set_Zrange
   (Plot              : access Gtk_Plot_3D_Record;
    Min, Max          :        Gdouble);
```

Set the minimal and maximal values for each axis.

75.3.1 Axis

```
procedure Axis_Hide_Title
   (Plot              : access Gtk_Plot_3D_Record;
    Side              :        Plot_Side);
```

Hide the title associated with the axis.

```
procedure Axis_Set_Major_Ticks
   (Plot              : access Gtk_Plot_3D_Record;
    Axis              :        Gtk.Extra.Plot.Plot_Orientation;
    Major_Step        :        Gdouble);
```

Modify the step for major ticks.
This is a percentage value that indicates how many major ticks are drawn along the axis.

See also Axis_Set_Ticks.

procedure Axis_Set_Minor_Ticks
```
(Plot          : access Gtk_Plot_3D_Record;
 Axis          :        Gtk.Extra.Plot.Plot_Orientation;
 Nminor        :        Gint);
```
Modify the number of minor ticks between each major one.
See also Axis_Set_Ticks.

procedure Axis_Set_Ticks
```
(Plot          : access Gtk_Plot_3D_Record;
 Axis          :        Gtk.Extra.Plot.Plot_Orientation;
 Major_Step    :        Gdouble;
 Nminor        :        Gint);
```
Set up ticks for a specific orientation.
A horizontal orientation will match the left and right sides, whereas a vertical orientation will match the top and bottom sides. Major_Step is a percentage value of the widget size, and indicate the step between each big ticks. For instance, if Major_Step has a value of 0.2, there will be 5 big ticks drawn along the axis. Num_Minor is the number of minor ticks between each major one.

procedure Axis_Set_Ticks_Length
```
(Plot          : access Gtk_Plot_3D_Record;
 Axis          :        Gtk.Extra.Plot.Plot_Orientation;
 Length        :        Gint);
```
Set the length (in pixels) of the big ticks.
The small ticks will have half this length.

procedure Axis_Set_Ticks_Width
```
(Plot          : access Gtk_Plot_3D_Record;
 Axis          :        Gtk.Extra.Plot.Plot_Orientation;
 Width         :        Gfloat);
```
Set the width (in pixels) of the ticks.
This width is common to both the long and short ticks.

procedure Axis_Show_Labels
```
(Plot          : access Gtk_Plot_3D_Record;
 Side          :        Plot_Side;
 Label_Mask    :        Gint);
```
Indicate whether a label should be drawn at each ticks to indicate
its value. Not all values of Labels_Mask are relevant for all axis. For instance, for a vertical axis, the relevant values are Label_Right and Label_Left.

procedure Axis_Show_Major_Ticks
```
(Plot          : access Gtk_Plot_3D_Record;
```

```
 Side                :      Plot_Side;
 Ticks_Mask          :      Gtk.Extra.Plot.Plot_Ticks_Pos);
```

Set the style of the major ticks along one of the axis

```
procedure Axis_Show_Minor_Ticks
       (Plot                : access Gtk_Plot_3D_Record;
        Side                :      Plot_Side;
        Ticks_Mask          :      Gtk.Extra.Plot.Plot_Ticks_Pos);
```

Set the style of the minor ticks along one of the axis

```
procedure Axis_Show_Ticks
       (Plot                : access Gtk_Plot_3D_Record;
        Side                :      Plot_Side;
        Major_Mask          :      Gtk.Extra.Plot.Plot_Ticks_Pos;
        Minor_Mask          :      Gtk.Extra.Plot.Plot_Ticks_Pos);
```

Set the style of the ticks.

```
procedure Axis_Show_Title
       (Plot                : access Gtk_Plot_3D_Record;
        Side                :      Plot_Side);
```

Show the title associated with the axis.

```
function Get_Axis
       (Plot                : access Gtk_Plot_3D_Record;
        Orientation         :      Gtk.Extra.Plot.Plot_Orientation)
        return Gtk.Extra.Plot.Gtk_Plot_Axis;
```

Return a handle to a specific axis.

```
function Get_Side
       (Plot                : access Gtk_Plot_3D_Record;
        Side                :      Plot_Side)
        return Gtk.Extra.Plot.Gtk_Plot_Axis;
```

Get the axis for a specific side.

```
procedure Axis_Set_Scale
       (Plot                : access Gtk_Plot_3D_Record;
        Axis                :      Gtk.Extra.Plot.Plot_Orientation;
        Scale               :      Gtk.Extra.Plot.Plot_Scale);
```

Set the scale for the axis

```
function Axis_Get_Scale
       (Plot                : access Gtk_Plot_3D_Record;
        Axis                :      Gtk.Extra.Plot.Plot_Orientation)
        return Gtk.Extra.Plot.Plot_Scale;
```

Get the current sale for the axis

75.3.2 Grid

procedure Major_Grids_Set_Visible
```
(Plot             : access Gtk_Plot_3D_Record;
 X, Y, Z          :        Boolean);
```
Indicate whether the grid should be displayed for each coordinate

procedure Major_Grids_Visible
```
(Plot             : access Gtk_Plot_3D_Record;
 X, Y, Z          : out    Boolean);
```
Indicate whether the grid is currently displayed.

procedure Minor_Grids_Set_Visible
```
(Plot             : access Gtk_Plot_3D_Record;
 X, Y, Z          :        Boolean);
```
Indicate whether the grid should be displayed for each coordinate

procedure Minor_Grids_Visible
```
(Plot             : access Gtk_Plot_3D_Record;
 X, Y, Z          : out    Boolean);
```
Indicate whether the grid is currently displayed.

procedure Major_Zgrid_Get_Attributes
```
(Plot             : access Gtk_Plot_3D_Record;
 Style            : out    Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width            : out    Gfloat;
 Color            : out    Gdk.Color.Gdk_Color);
```
Get the attributes of the major grid

procedure Major_Zgrid_Set_Attributes
```
(Plot             : access Gtk_Plot_3D_Record;
 Style            :        Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width            :        Gfloat;
 Color            :        Gdk.Color.Gdk_Color);
```
Set the attributes of the major grid

procedure Minor_Zgrid_Get_Attributes
```
(Plot             : access Gtk_Plot_3D_Record;
 Style            : out    Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width            : out    Gfloat;
 Color            : out    Gdk.Color.Gdk_Color);
```
Get the attributes of the minor grid

procedure Minor_Zgrid_Set_Attributes
```
(Plot                  : access Gtk_Plot_3D_Record;
 Style                 :      Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width                 :      Gfloat;
 Color                 :      Gdk.Color.Gdk_Color);
```
Set the attributes of the minor grid

75.3.3 Rotating

procedure Reset_Angles
```
(Plot                  : access Gtk_Plot_3D_Record);
```
reset all the angles to their default values

procedure Rotate
```
(Plot                  : access Gtk_Plot_3D_Record;
 Angle_X, Angle_Y, Angle_Z  :      Gdouble);
```
Rotate the plot along the three axis at the same time.
The angles are specified in degrees.

procedure Rotate_Vector
```
(Plot                  : access Gtk_Plot_3D_Record;
 Vector                :      Plot_Vector;
 A1, A2, A3            :      Gdouble);
```
Rotate Vector along the three axis.
The three angles A1, A2 and A3 are specified in degrees.

procedure Rotate_X
```
(Plot                  : access Gtk_Plot_3D_Record;
 Angle                 :      Gdouble);
```

procedure Rotate_Z
```
(Plot                  : access Gtk_Plot_3D_Record;
 Angle                 :      Gdouble);
```
Rotate the plot along a specific axis.
Angle is specific in degrees.

75.3.4 Planes

A 3D plot is associated, as usual, with three axis (one per coordinate
X, Y and Z). These three axis, together, define 3 planes that can be shown or hidden, and on which a grid can be displayed to make it easy to visualize the value of the data.

```
procedure Plane_Set_Color
       (Plot                : access Gtk_Plot_3D_Record;
        Plane               :      Plot_Plane;
        Color               :      Gdk.Color.Gdk_Color);
```

Define the background color to use for one of the planes. Each plane can have its own color.

```
procedure Plane_Set_Visible
       (Plot                : access Gtk_Plot_3D_Record;
        Plane               :      Plot_Plane;
        Visible             :      Boolean);
```

Indicate whether each plane should be displayed or not.

```
function Plane_Visible
       (Plot                : access Gtk_Plot_3D_Record;
        Plane               :      Plot_Plane)
        return Boolean;
```

Indicate whether a plane is currently visible or not.

75.3.5 Corners

In addition to drawing the three planes defined by the axis, a 3D plot can also draw some lines to draw a cube around the plot (although the three new planes defined by these lines are left transparent so that the plot is visible.

```
procedure Corner_Get_Attributes
       (Plot                : access Gtk_Plot_3D_Record;
        Style               : out   Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               : out   Gfloat;
        Color               : out   Gdk.Color.Gdk_Color);
```

Get the style of the corner lines.

```
procedure Corner_Set_Attributes
       (Plot                : access Gtk_Plot_3D_Record;
        Style               :      Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               :      Gfloat;
        Color               :      Gdk.Color.Gdk_Color);
```

Define the style of the corner lines.

```
procedure Corner_Set_Visible
       (Plot                : access Gtk_Plot_3D_Record;
        Visible             :      Boolean);
```

Whether corners should be visible

```
function Corner_Visible
   (Plot               : access Gtk_Plot_3D_Record)
    return Boolean;
```

Indicate whether corners are visible

75.3.6 Misc

```
procedure Frame_Get_Attributes
   (Plot               : access Gtk_Plot_3D_Record;
    Style              : out    Gtk.Extra.Plot_Data.Plot_Line_Style;
    Width              : out    Gfloat;
    Color              : out    Gdk.Color.Gdk_Color);
```

```
procedure Frame_Set_Attributes
   (Plot               : access Gtk_Plot_3D_Record;
    Style              :        Gtk.Extra.Plot_Data.Plot_Line_Style;
    Width              :        Gfloat;
    Color              :        Gdk.Color.Gdk_Color);
```

```
procedure Get_Pixel
   (Plot               : access Gtk_Plot_3D_Record;
    X, Y, Z            :        Gdouble;
    Px, Py, Pz         : out    Gdouble);
```

```
function Get_Titles_Offset
   (Plot               : access Gtk_Plot_3D_Record)
    return Gint;
```

```
procedure Set_Titles_Offset
   (Plot               : access Gtk_Plot_3D_Record;
    Offset             :        Gint);
```

76 Package Gtk.Extra.Plot_Bar

This special type of data set displays itself with bar (also known as histograms).

76.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Plot_Data            (Package Gtk.Extra.Plot_Data)
         \___ Gtk_Plot_Bar          (Package Gtk.Extra.Plot_Bar)
```

76.2 Subprograms

```
procedure Gtk_New
       (Bar                  : out    Gtk_Plot_Bar;
        Orientation          :        Gtk.Enums.Gtk_Orientation);
```

Create a new Plot bar.

```
function Get_Type             return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Bar.

```
procedure Set_Width
       (Bar                  : access Gtk_Plot_Bar_Record'Class;
        Width                :        Gdouble);
```

Set the width of the bars

```
function Get_Width
       (Bar                  : access Gtk_Plot_Bar_Record'Class)
        return Gdouble;
```

Return the width used to draw the bars

77 Package Gtk.Extra.Plot_Box

This special type of data set displays itself with boxes

77.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Plot_Data            (Package Gtk.Extra.Plot_Data)
         \___ Gtk_Plot_Box          (Package Gtk.Extra.Plot_Box)
```

77.2 Subprograms

procedure Gtk_New
```
       (Box                : out   Gtk_Plot_Box;
        Orientation        :      Gtk.Enums.Gtk_Orientation);
```
Create a new Plot bar.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Box

78 Package Gtk.Extra.Plot_Canvas

A Gtk_Plot_Canvas is a special kind of drawing area used with Gtk_Plot widgets. It provides drag-and-drop capabilities for the texts, legends, points... available in a Gtk_Plot. Note that this widget is specifically designed for Gtk_Plot widgets, and won't provide any other capability for other kinds of widgets.

Like any child of Gtk_Layout, this widget can have an almost unlimited size for its children, and provides scrolling.

78.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Container       (Package Gtk.Container)
         \___ Gtk_Fixed        (Package Gtk.Fixed)
            \___ Gtk_Plot_Canvas (Package Gtk.Extra.Plot_Canvas)
```

78.2 Signals

- **"changed"**

  procedure Handler (Canvas : access Gtk_Plot_Canvas_Record'Class);

  Called when the contents of the canvas has changed (an item was moved interactively by the user).

  A region of the canvas was selected by the user.

- **"delete_item"**

  procedure Handler (Canvas : access Gtk_Plot_Canvas_Record'Class;
  Item : Gtk_Plot_Canvas_Child);

  Called when an item is being removed from the canvas

- **"move_item"**

  function Handler (Canvas : access Gtk_Plot_Canvas_Record'Class;
  Item : Gtk_Plot_Canvas_Child;
  New_X : Gdouble;
  New_Y : Gdouble)
  return Boolean;

  An item was moved on the canvas. Its coordinates have not changed yet, but if the handler returns True they will become (New_X, New_Y). If the handler returns False, nothing happens.

- **"resize_item"**

  function Handler (Canvas : access Gtk_Plot_Canvas_Record'Class;
  Item : Gtk_Plot_Canvas_Child;

New_Width : Gdouble;
New_Height : Gdouble)
return Boolean;

An item is being resized. Its size has not changed yet, but if the handler returns True it will become (New_Width, New_Height). If the handler returns False, nothing happens.

- **"select_item"**

  function Handler (Canvas : access Gtk_Plot_Canvas_Record'Class;
  Event : Gdk_Button_Event;
  Item : Gtk_Plot_Canvas_Child)
  return Boolean;

  Called when an item was selected. An item can be anything, from a text to a plot When this signal is called, the item was simply selected, but not dragged. The handler should return False if the item can not be selected.

- **"select_region"**

  procedure Handler (Canvas : access Gtk_Plot_Canvas_Record'Class;
  X_Min : Gdouble;
  Y_Min : Gdouble;
  X_Max : Gdouble;
  Y_Max : Gdouble);

78.3 Types

```
type Child_Draw_Func is access procedure
         (Canvas : System.Address;
          Child  : Gtk_Plot_Canvas_Child);
```

Generic format of functions used to draw a child of the canvas. Canvas is a System.Address since these functions are called directly from C and GtkAda can't insert its hooks. However, you can use the Convert function below to convert to a Gtk_Plot_Canvas.

type Gtk_Plot_Canvas_Child **is new** Gdk.C_Proxy;

```
type Plot_Canvas_Action is
         (Action_Inactive,
          Action_Selection,
          Action_Drag,
          Action_Resize);
```

The action being performed on the canvas. pragma Convention (C, Plot_Canvas_Action);

type Plot_Canvas_Arrow **is new** Gint;

type Plot_Canvas_Flag **is new** Gint;

```
type Plot_Canvas_Pos is
         (Canvas_Out,
          Canvas_In,
          Canvas_Left,
          Canvas_Right,
          Canvas_Top,
          Canvas_Bottom,
          Canvas_Top_Left,
          Canvas_Top_Right,
          Canvas_Bottom_Left,
          Canvas_Bottom_Right);
```

The position of the items in the canvas. pragma Convention (C, Plot_Canvas_Pos);

```
type Plot_Canvas_Type is
         (None,
          Plot,
          Axis,
          Legends,
          Title,
          Text,
          Data,
          Line,
          Rectangle,
          Ellipse,
          Custom);
```

The type of data that can be put in a canvas. Plot is only for a Gtk.Extra.Plot.Gtk_Plot widget. pragma Convention (C, Plot_Canvas_Type);

78.4 Subprograms

78.4.1 Creating and manipulating the canvas

```
procedure Gtk_New
       (Widget              : out   Gtk_Plot_Canvas;
        Width               : in    Gint;
        Height              : in    Gint;
        Magnification       : in    Gdouble := 1.0);
```

Create a new Gtk_Plot_Canvas, with a specific screen size.
Since the widget can have an unlimited internal size, it does not try to set its size to accommodate all of its children.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Plot_Canvas.

function Child_Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Plot_Canvas_Child.

procedure Refresh
```
(Canvas              : access Gtk_Plot_Canvas_Record);
```

Force a refresh of the canvas on the screen. The screen is updated from the contents of the double-buffer.

procedure Paint
```
(Canvas              : access Gtk_Plot_Canvas_Record);
```

Redraw each of the items included in the canvas. The painting is done in the double-buffer, and must be drawn on the screen with Refresh.

procedure Freeze
```
(Canvas              : access Gtk_Plot_Canvas_Record);
```

Freeze all graphical updates to the screen. This significanly speeds up the updates to the plot

procedure Thaw
```
(Canvas              : access Gtk_Plot_Canvas_Record);
```

Reactivate all graphical updates to the screen

function Get_Pixmap
```
(Canvas              : access Gtk_Plot_Canvas_Record)
 return Gdk.Pixmap.Gdk_Pixmap;
```

Return the pixmap associated with the Canvas.
If you add your own items on the canvas (see Child_New below), you can draw them on this pixmap to make them visible on the canvas. You need to call Refresh to send this pixmap to the screen.

procedure Grid_Set_Visible
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Visible             : in    Boolean);
```

Indicate whether the grid should be visible or not.

procedure Grid_Set_Step
```
(Canvas          : access Gtk_Plot_Canvas_Record;
 Step            : in    Gint);
```

Set the space between two lines of the grid.

procedure Grid_Set_Attributes
```
(Canvas          : access Gtk_Plot_Canvas_Record;
 Style           : in    Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width           : in    Gint;
 Color           : in    Gdk.Color.Gdk_Color);
```

Set the attributes of the grid.

procedure Add_Plot
```
(Plot_Canvas     : access Gtk_Plot_Canvas_Record;
 Plot            : access Gtk.Extra.Plot.Gtk_Plot_Record'Class;
 X               : in    Gdouble;
 Y               : in    Gdouble);
```

Add a new plot to the list handled by the canvas.
The canvas will then provide drag-and-drop functionalities for that plot. The plot is displayed at the relative screen coordinates (X, Y). Plot becomes the new active plot, and its associated pixmap becomes the Plot_Canvas itself.

procedure Set_Active_Plot
```
(Plot_Canvas     : access Gtk_Plot_Canvas_Record;
 Plot            : access Gtk.Extra.Plot.Gtk_Plot_Record'Class);
```

Modify the active plot in the canvas.
The active plot is generally the one that received the last click event (drag-and-drop, etc.). This should be set before emitting any of the signals in this class.

procedure Cancel_Action
```
(Plot_Canvas     : access Gtk_Plot_Canvas_Record);
```

Cancel the current action.
This can be called in the user callbacks to ignore temporarily some of the signals below.

function Get_Active_Plot
```
(Canvas          : access Gtk_Plot_Canvas_Record)
 return Gtk.Extra.Plot.Gtk_Plot;
```

Return the active plot.
In the callbacks for the signals below, this is the widget that got the signal.

function Get_Active_Data
```
(Canvas          : access Gtk_Plot_Canvas_Record)
 return Gtk.Extra.Plot_Data.Gtk_Plot_Data;
```

Return the active dataset (which of course belongs to the active plot).
This is the dataset that was last clicked on.

procedure Get_Active_Point
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 X                   : out   Gdouble;
 Y                   : out   Gdouble;
 Index               : out   Gint);
```
Return the relative coordinates of the active point in the
active dataset. Index will contain the index of the active point in the list of the canvas'children, or -1 if there is no active point. This is the index in the coordinates arrays of Get_Active_Data. This is the one that was last clicked on.

procedure Set_Size
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Width               : in    Gint;
 Height              : in    Gint);
```
Modify the size allocated for the canvas, and the size of the pixmap
the plots are displayed on.

procedure Unselect
```
(Canvas              : access Gtk_Plot_Canvas_Record);
```
Unselect the currently selected item.

procedure Set_Magnification
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Magnification       :        Gdouble := 1.0);
```
Changes the magnification for the canvas.
1.0 is the default value. Higher values will zoom in, while lower values will zoom out.

procedure Set_Background
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Background          :        Gdk.Color.Gdk_Color);
```
Set the background color for the canvas.

procedure Set_Transparent
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Transparent         :        Boolean);
```
Whether the canvas should be transparent. If Transparent is True, all
background attributes are ignored

function Transparent
```
(Canvas              : access Gtk_Plot_Canvas_Record)
 return Boolean;
```

Whether the canvas is currently transparent

procedure Get_Pixel
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Px                  : in    Gdouble;
 Py                  : in    Gdouble;
 X                   : out   Gint;
 Y                   : out   Gint);
```
Convert from relative coordinates to absolute ones.

procedure Get_Position
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 X                   : in    Gint;
 Y                   : in    Gint;
 Px                  : out   Gdouble;
 Py                  : out   Gdouble);
```
Convert from absolute coordinates to relative ones.

78.4.2 Canvas items

There are several different types of items that can be put on the canvas, and then manipulated interactively by the user.

function Convert
```
(Canvas              :      System.Address)
 return Gtk_Plot_Canvas;
```
Convert from a System.Address returned by C to a real Gtk_Plot_Canvas structure.

procedure Set_Draw_Func
```
(Child               :      Gtk_Plot_Canvas_Child;
 Draw                :      Child_Draw_Func);
```
Set the function used to draw the item.
This should be used only for items whose type is Custom, since other items have their own drawing functions.

function Get_Active_Item
```
(Canvas              : access Gtk_Plot_Canvas_Record)
 return Gtk_Plot_Canvas_Child;
```
Return the currently selected item.

procedure Remove_Child
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Child               :      Gtk_Plot_Canvas_Child);
```
Remove a child from the canvas

```
function Put_Text
       (Canvas              : access Gtk_Plot_Canvas_Record;
        X                   : in    Gdouble;
        Y                   : in    Gdouble;
        Ps_Font             : in    String;
        Height              : in    Gint;
        Angle               : in    Gint;
        Fg                  : in    Gdk.Color.Gdk_Color;
        Bg                  : in    Gdk.Color.Gdk_Color;
        Transparent         : in    Boolean;
        Justification       : in    Gtk.Enums.Gtk_Justification;
        Text                : in    String)
        return Gtk_Plot_Canvas_Child;
```

Put an arbitrary text in the layout.
Ps_Font should be the name of a postscript font. (X, Y) are the relative coordinates to which the text should be drawn. The only legal values for Angle are 0, 90, 180 and 270 degrees.

Text can contain some special characters, that change is renderering. They all begin with a '\' (backslash) character, followed by one of:

- '0' .. '9' : Change the font (take the nth font in the family
- 'g' : Select the "Symbol" font
- 'B' : Activate bold characters.
- 'i' : Activate italic characters.
- 'S' or '^' : Activate superscripts.
- 's' or '_' : Activate subscripts.
- '+' : Increment the fontsize by 3 pixels.
- '-' : Decrement the fontsize by 3 pixels.
- 'N' : Restore the default characteristics of the font.
- 'b' : Move back one character.

```
function Put_Line
       (Canvas              : access Gtk_Plot_Canvas_Record;
        X1                  :        Gdouble;
        Y1                  :        Gdouble;
        X2                  :        Gdouble;
        Y2                  :        Gdouble;
        Style               :        Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               :        Gfloat;
        Color               :        Gdk.Color.Gdk_Color;
        Arrow_Mask          :        Plot_Canvas_Arrow)
        return Gtk_Plot_Canvas_Child;
```

Draw a line in the background of the canvas.

```
function Put_Rectangle
       (Canvas              : access Gtk_Plot_Canvas_Record;
        X1                  :      Gdouble;
        Y1                  :      Gdouble;
        X2                  :      Gdouble;
        Y2                  :      Gdouble;
        Style               :      Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               :      Gfloat;
        Fg                  :      Gdk.Color.Gdk_Color;
        Bg                  :      Gdk.Color.Gdk_Color;
        Border              :      Gtk.Extra.Plot.Plot_Border_Style;
        Fill                :      Boolean := False)
        return Gtk_Plot_Canvas_Child;
```

Draw a rectangle in the canvas.

```
function Put_Ellipse
       (Canvas              : access Gtk_Plot_Canvas_Record;
        X1                  :      Gdouble;
        Y1                  :      Gdouble;
        X2                  :      Gdouble;
        Y2                  :      Gdouble;
        Style               :      Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               :      Gfloat;
        Fg                  :      Gdk.Color.Gdk_Color;
        Bg                  :      Gdk.Color.Gdk_Color;
        Fill                :      Boolean := False)
        return Gtk_Plot_Canvas_Child;
```

Draw an ellipse in the canvas.

```
function Put_Pixmap
       (Canvas              : access Gtk_Plot_Canvas_Record;
        Pixmap              :      Gdk.Pixmap.Gdk_Pixmap;
        X1, Y1              :      Gdouble)
        return Gtk_Plot_Canvas_Child;
```

Draw a pixmap

```
procedure Line_Set_Attributes
       (Child               :      Gtk_Plot_Canvas_Child;
        Style               :      Gtk.Extra.Plot_Data.Plot_Line_Style;
        Width               :      Gfloat;
        Color               :      Gdk.Color.Gdk_Color;
        Mask                :      Plot_Canvas_Arrow);
```

Change the attributes of a line.

```
procedure Rectangle_Set_Attributes
       (Child               :      Gtk_Plot_Canvas_Child;
```

```
 Style          :      Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width          :      Gfloat;
 Fg             :      Gdk.Color.Gdk_Color;
 Bg             :      Gdk.Color.Gdk_Color;
 Border         :      Gtk.Extra.Plot.Plot_Border_Style;
 Fill           :      Boolean := False);
```

Change the attributes of a rectangle.

procedure Ellipse_Set_Attributes

```
(Child          :      Gtk_Plot_Canvas_Child;
 Style          :      Gtk.Extra.Plot_Data.Plot_Line_Style;
 Width          :      Gfloat;
 Fg             :      Gdk.Color.Gdk_Color;
 Bg             :      Gdk.Color.Gdk_Color;
 Fill           :      Boolean := False);
```

Change the attributes for an ellipse.

78.4.3 Custom children

You can insert your own items in a canvas.
While the canvas will take care of moving the item, it is your responsability to provide a visual rendering for it.

function Child_New

```
(Child_Type     :      Plot_Canvas_Type := Custom)
 return Gtk_Plot_Canvas_Child;
```

Create a new child.

procedure Put_Child

```
(Canvas         : access Gtk_Plot_Canvas_Record;
 Child          :      Gtk_Plot_Canvas_Child;
 X1             :      Gdouble;
 Y1             :      Gdouble;
 X2             :      Gdouble;
 Y2             :      Gdouble);
```

Insert a new item in the canvas. It will occupy the area defined by
the four coordinates.

procedure Child_Move

```
(Canvas         : access Gtk_Plot_Canvas_Record;
 Child          :      Gtk_Plot_Canvas_Child;
 X1             :      Gdouble;
 Y1             :      Gdouble);
```

Move an item, but does not change its size.

procedure Child_Move_Resize
```
(Canvas              : access Gtk_Plot_Canvas_Record;
 Child               :      Gtk_Plot_Canvas_Child;
 X1                  :      Gdouble;
 Y1                  :      Gdouble;
 X2                  :      Gdouble;
 Y2                  :      Gdouble);
```
Move an resize an item in the canvas.

function Get_Item_Type
```
(Item                :      Gtk_Plot_Canvas_Child)
 return Plot_Canvas_Type;
```
Return the type of the item.

function Get_Allocation_Width
```
(Child               :      Gtk_Plot_Canvas_Child)
 return Guint;
```
Return the current width of the child.

function Get_Allocation_Height
```
(Child               :      Gtk_Plot_Canvas_Child)
 return Guint;
```
Return the current height of the child.

function Get_Allocation_X
```
(Child               :      Gtk_Plot_Canvas_Child)
 return Gint;
```
Return the current position of the child, relative to its canvas.

function Get_Allocation_Y
```
(Child               :      Gtk_Plot_Canvas_Child)
 return Gint;
```
Return the current position of the child, relative to its canvas.

function Get_Flags
```
(Child               :      Gtk_Plot_Canvas_Child)
 return Plot_Canvas_Flag;
```
Return the list of actions currently possible on the child.

procedure Set_Flags
```
(Child               :      Gtk_Plot_Canvas_Child;
 Flags               :      Plot_Canvas_Flag);
```
Modify the list of actions possible for a child.

78.4.4 Flags

Some flags are defined for this widget. You can not access them through the usual interface in Gtk.Object.Flag_Is_Set since this widget is not part of the standard gtk+ packages. Instead, use the functions below.

- "can_select" True if it is possible to select a region of the canvas
- "can_select_item" True if it is possible to select any of the item on the canvas.
- "can_select_point" True if the individual points in the plots can be selected and interactively moved by the user.
- "can_dnd" True if it is possible to drag an item on the canvas.
- "can_dnd_point" True if the points of the plots can be moved interactively.

function Plot_Canvas_Flag_Is_Set

```
(Plot_Canvas        : access Gtk_Plot_Canvas_Record;
 Flag               : in    Guint16)
 return Boolean;
```

Test whether one of the flags for a Gtk_Plot_Canvas widget or its children is set.

procedure Plot_Canvas_Set_Flags

```
(Plot_Canvas        : access Gtk_Plot_Canvas_Record;
 Flags              : in    Guint16);
```

Set the flags for a Gtk_Plot_Canvas widget or its children.
Note that the flags currently set are not touched by this function. This can only be used for the flags defined in the Gtk.Extra.Gtk_Plot_Canvas package.

procedure Plot_Canvas_Unset_Flags

```
(Plot_Canvas        : access Gtk_Plot_Canvas_Record;
 Flags              : in    Guint16);
```

Unset the flags for a Gtk_Plot_Canvas.

79 Package Gtk.Extra.Plot_Data

This package defines the root of the plot hierarchy. It defines several display strategies that can be used to show scientific data on the screen (see the children for 3D, polar, bars,...)

All coordinates are in percent of the total size allocates for the data set (ie the actual position is (x * width, y * height), where (x, y) is the value stored in the data set and (width, height) its allocated screen size.

79.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Plot_Data          (Package Gtk.Extra.Plot_Data)
```

79.2 Types

type Gdouble_Array_Access **is access all** Glib.Gdouble_Array;

The reason we use this type in the functions below is because gtk+-extra does not keep a copy of the arrays, but points to the one given in argument. Thus, the Ada arrays should not be allocated on the stack, or at least they should be at library level. Using this 'Access will force the compiler to do the check for us.

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

type No_Range_Gdouble_Array **is array** (Natural) **of** Gdouble;

An array of values. This is used to represent the data values displayed in the plot. This array does not have any range information (so that it can be easily returned from a C function, without requiring an extra copy of the table). You can not use 'Range on this array.

type No_Range_Gdouble_Array_Access **is access all** No_Range_Gdouble_Array;

An access to a flat array.

type Plot_Connector **is**
```
        (Connect_None,
            -- No connection

         Connect_Straight,
            -- straight line

         Connect_Spline,
            -- spline or Bezier curve
```

```
Connect_Hv_Step,
   -- Horizontal then vertical

Connect_Vh_Step,
   -- Vertical then horizontal

Connect_Middle_Step
   -- Split in the middle
);
```

The type of connection between two adjacent points in a graph. pragma Convention (C, Plot_Connector);

type Plot_Function **is access function**

```
(Plot  : System.Address;
```

type Plot_Gradient **is new** Integer;

Indicate which color components vary along the gradient

type Plot_Label_Style **is**

```
(Label_Float, Label_Exp, Label_Pow);
```

The style of labels (floating point, or scientific notation) pragma Convention (C, Plot_Label_Style);

type Plot_Line_Style **is**

```
(Line_None,
 Line_Solid,
 Line_Dotted,
 Line_Dashed,
 Line_Dot_Dash,
 Line_Dot_Dot_Dash,
 Line_Dot_Dash_Dash);
```

Lines used to connect two adjacent points in a graph. pragma Convention (C, Plot_Line_Style);

type Plot_Scale **is**

```
(Scale_Linear, Scale_Log10);
```

Type of scale used for each axis of a graph. pragma Convention (C, Plot_Scale);

type Plot_Symbol_Style **is**

```
         (Symbol_Empty,
          Symbol_Filled,
          Symbol_Opaque);
```

Style used to draw the points in a graph. pragma Convention (C, Plot_Symbol_Style);

type Plot_Symbol_Type **is**
```
         (Symbol_None,
          Symbol_Square,
          Symbol_Circle,
          Symbol_Up_Triangle,
          Symbol_Down_Triangle,
          Symbol_Right_Triangle,
          Symbol_Left_Triangle,
          Symbol_Diamond,
          Symbol_Plus,
          Symbol_Cross,
          Symbol_Star,
          Symbol_Dot,
          Symbol_Impulse);
```

Type of symbol used to represent the points in a graph. pragma Convention (C, Plot_Symbol_Type);

type Points_Array **is record**
```
         Points     : No_Range_Gdouble_Array_Access;
         Num_Points : Gint := 0;
         end record;
```

The points are indexed from 0 to Num_Points-1. Note that you can't use 'Range, 'First or 'Last on Points.

79.3 Subprograms

79.3.1 Creating a Data set

procedure Gtk_New
```
       (Data                 : out   Gtk_Plot_Data;
        Func                 :       Plot_Function := null);
```
Creates a new data set. Its values can either be generated automatically
from Func, or will have to be set explicitely using the other subprograms in this package.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Plot_Data.

procedure Set_Name
```
(Data                : access Gtk_Plot_Data_Record;
 Name                :        String);
```
Set the name used internally for that dataset.
This name does not appear anywhere on the screen, but it is easier to find the dataset afterward by using this name.

79.3.2 Drawing a set

Although a set is basically a list of values, it is closely associated with its representation on the screen (see the children of Gtk_Plot_Data for various possible representations). The Gtk.Extra packages are designed so that the drawing can be done either to the screen (through a Gdk adapter), to a postscript file for easy printing, or to any other media.

procedure Paint
```
(Data                : access Gtk_Plot_Data_Record);
```
Emits the "draw_data" signal to request a redrawing of the data set.

procedure Update
```
(Data                : access Gtk_Plot_Data_Record);
```
Indicates that the data has changed, and the graphical view should reflect this.

procedure Draw_Points
```
(Data                : access Gtk_Plot_Data_Record;
 N                   :        Gint);
```
Draw the N last (most recent) values of the Data set on the screen.
If N is greater than the actual number of values in Data, then they are all displayed. This subprogram should be used when you want to periodically update the contents of a dataset (you would then modify the number of points in the dataset with a call to Set_Numpoints, then register the new points with Set_X and Set_Y, and finally refresh the dataset with a call to Draw_Points and Gtk.Plot.Refresh).

procedure Draw_Symbol
```
(Data                : access Gtk_Plot_Data_Record;
 X, Y                :        Gdouble);
```
Draw the current symbol (see Set_Symbol) at specific coordinates on the screen.

79.3.3 Manipulating values

procedure Set_Points
```
(Data                : access Gtk_Plot_Data_Record;
```

```
 X                     :      Gdouble_Array_Access;
 Y                     :      Gdouble_Array_Access;
 Dx                    :      Gdouble_Array_Access;
 Dy                    :      Gdouble_Array_Access);
```

Set some explicit points in the set.
Note that the set must not be associated with a function, or the points will simply be ignored. All of the arrays must have the same length, the behavior is undefined otherwise. X and Y are the list of coordinates of the points. Dx and Dy are the list of size (precision) of these points. A bigger symbol will be displayed for the point whose (Dx, Dy) value is bigger.

procedure Get_Points

```
(Data                  : access Gtk_Plot_Data_Record;
 X                     : out   Points_Array;
 Y                     : out   Points_Array;
 Dx                    : out   Points_Array;
 Dy                    : out   Points_Array);
```

Return the value of the points in the set.
Null-length arrays are returned if the set is associated with a function, since no explicit point has been set. See Set_Points for a definition of X, Y, Dx and Dy.

procedure Set_X

```
(Data                  : access Gtk_Plot_Data_Record;
 X                     :      Gdouble_Array_Access);
```

procedure Set_Z

```
(Data                  : access Gtk_Plot_Data_Record;
 Z                     :      Gdouble_Array_Access);
```

procedure Set_A_Scale

```
(Data                  : access Gtk_Plot_Data_Record;
 A_Scale               :      Gdouble);
```

procedure Set_Dx

```
(Data                  : access Gtk_Plot_Data_Record;
 Dx                    :      Gdouble_Array_Access);
```

procedure Set_Dz

```
(Data                  : access Gtk_Plot_Data_Record;
 Dz                    :      Gdouble_Array_Access);
```

Set the precision of the points in the set. A bigger symbol is displayed for the points whose (Dx, Dy, Dz) is bigger. The array must have a length of Get_Numpoints (if GtkAda was compiled with assertions enabled, an exception will be raised if the length are different). No copy of the array is made for efficiency reasons, thus modifying the array content later on will also modify the plot.

procedure Set_Da
```
(Data                 : access Gtk_Plot_Data_Record;
 Da                   :        Gdouble_Array_Access);
```
Specifies the colors to use for the points.
The color of the symbols is detemined using the gradient. the gradient has (min, max) values, and corresponding colors. The symbol's color is interpolated between these values using hue/saturation/value depending on the gradient_mask.

function Get_X
```
(Data                 : access Gtk_Plot_Data_Record)
 return Points_Array;
```

function Get_Z
```
(Data                 : access Gtk_Plot_Data_Record)
 return Points_Array;
```

function Get_Dx
```
(Data                 : access Gtk_Plot_Data_Record)
 return Points_Array;
```

function Get_Dz
```
(Data                 : access Gtk_Plot_Data_Record)
 return Points_Array;
```

procedure Set_Numpoints
```
(Data                 : access Gtk_Plot_Data_Record;
 Num                  :        Gint);
```
Set the number of points that should be expected in the graph.
Note that this does not automatically resize all the internal structure, it just indicates what size the parameters to Set_X, Set_Y,... should have.

function Get_Numpoints
```
(Data                 : access Gtk_Plot_Data_Record)
 return Gint;
```

Return the number of points expected in the graph.

79.3.4 Labels

Each point in the data set can be associated with a label that describes it. This is only relevant for data sets where you explicitely give values, not when the values are generated by a function.

procedure Set_Labels
```
(Data          : access Gtk_Plot_Data_Record;
 Labels        :        Gtkada.Types.Chars_Ptr_Array);
```
Set the labels associated which each point in the canvas.
There must be at least Get_Numpoints elements in Labels, or the behavior is undefined

function Get_Labels
```
(Data          : access Gtk_Plot_Data_Record)
 return Gtkada.Types.Chars_Ptr_Array;
```
Return the labels associated with the points in the data set.
Note that this returns a *copy* of the actual array, and thus might be expensive to call.

procedure Show_Labels
```
(Data          : access Gtk_Plot_Data_Record;
 Show          :        Boolean);
```
Indicate whether the labels should be displayed next to each point in the data set. This has no effect if no labels were specified.

procedure Labels_Set_Attributes
```
(Data          : access Gtk_Plot_Data_Record;
 Font          :        String;
 Height        :        Gint;
 Angle         :        Gint;
 Foreground    :        Gdk.Color.Gdk_Color;
 Background    :        Gdk.Color.Gdk_Color);
```
Set the properties of the labels

79.3.5 Symbols and Connectors

Each point that is explicitely set in the data set through the Set_X, Set_Y,... subprograms is visually associated with a symbol. There are several representations for the symbols.

All these symbols are then connected by a line, a curve or any other link. These are called connectors.

Each symbol, in addition to being connected to the next one with a connector, can also be linked to the axis X=0, Y=0 or Z=0 so that it is easier to read its coordinates. These are called errbars, and they must be explicitely shown.

```
procedure Set_Symbol
       (Data             : access Gtk_Plot_Data_Record;
        The_Type         :        Plot_Symbol_Type;
        Style            :        Plot_Symbol_Style;
        Size             :        Gint;
        Line_Width       :        Gfloat;
        Color            :        Gdk.Color.Gdk_Color;
        Border_Color     :        Gdk.Color.Gdk_Color);
```

Set the visual aspect of the symbols.

```
procedure Get_Symbol
       (Data             : access Gtk_Plot_Data_Record;
        The_Type         : out    Plot_Symbol_Type;
        Style            : out    Plot_Symbol_Style;
        Size             : out    Gint;
        Line_Width       : out    Gint;
        Color            : out    Gdk.Color.Gdk_Color;
        Border_Color     : out    Gdk.Color.Gdk_Color);
```

Return the visual characteristics of the symbols.

```
procedure Set_Connector
       (Data             : access Gtk_Plot_Data_Record;
        Connector        :        Plot_Connector);
```

Set the style of the connectors.

```
function Get_Connector
       (Data             : access Gtk_Plot_Data_Record)
        return Plot_Connector;
```

Return the connector style used for the data set.

```
procedure Set_Line_Attributes
       (Data             : access Gtk_Plot_Data_Record;
        Style            :        Plot_Line_Style;
        Cap_Style        :        Gdk.GC.Gdk_Cap_Style;
        Join_Style       :        Gdk.GC.Gdk_Join_Style;
        Width            :        Gfloat;
        Color            :        Gdk.Color.Gdk_Color);
```

Set the line style used for the connectors.

```
procedure Get_Line_Attributes
       (Data             : access Gtk_Plot_Data_Record;
        Style            : out    Plot_Line_Style;
        Cap_Style        : out    Gdk.GC.Gdk_Cap_Style;
        Join_Style       : out    Gdk.GC.Gdk_Join_Style;
        Width            : out    Gfloat;
        Color            : out    Gdk.Color.Gdk_Color);
```

Return the line attributes used for the connectors.

```
procedure Set_X_Attributes
   (Data             : access Gtk_Plot_Data_Record;
    Style            :        Plot_Line_Style;
    Cap_Style        :        Gdk.GC.Gdk_Cap_Style;
    Join_Style       :        Gdk.GC.Gdk_Join_Style;
    Width            :        Gfloat;
    Color            :        Gdk.Color.Gdk_Color);
```

Set the style of the lines used to connect the symbols to the X axis.

```
procedure Set_Y_Attributes
   (Data             : access Gtk_Plot_Data_Record;
    Style            :        Plot_Line_Style;
    Cap_Style        :        Gdk.GC.Gdk_Cap_Style;
    Join_Style       :        Gdk.GC.Gdk_Join_Style;
    Width            :        Gfloat;
    Color            :        Gdk.Color.Gdk_Color);
```

Set the style of the lines used to connect the symbols to the Y axis.

```
procedure Set_Z_Attributes
   (Data             : access Gtk_Plot_Data_Record;
    Style            :        Plot_Line_Style;
    Cap_Style        :        Gdk.GC.Gdk_Cap_Style;
    Join_Style       :        Gdk.GC.Gdk_Join_Style;
    Width            :        Gfloat;
    Color            :        Gdk.Color.Gdk_Color);
```

Set the style of the lines used to connect the symbols to the Z axis.

```
procedure Show_Xerrbars
   (Data             : access Gtk_Plot_Data_Record);
```

```
procedure Show_Zerrbars
   (Data             : access Gtk_Plot_Data_Record);
```

Indicate that each symbol should be connected to the various axis

```
procedure Hide_Xerrbars
   (Data             : access Gtk_Plot_Data_Record);
```

```
procedure Hide_Zerrbars
   (Data             : access Gtk_Plot_Data_Record);
```

Indicate the the symbol should not be connected to the axis.

```
procedure Fill_Area
   (Data                 : access Gtk_Plot_Data_Record;
    Fill                 :        Boolean);
```

Indicate whether the area between two points should be filled or not.

```
function Area_Is_Filled
   (Data                 : access Gtk_Plot_Data_Record)
    return Boolean;
```

Indicate whether the area between two points is filled.

79.3.6 Legends

In addition to the drawing corresponding to the data set, it is possible
to display a box that contains a legend. This is particulary useful when multiple data sets are displayed on the same plot.

```
procedure Set_Legend
   (Data                 : access Gtk_Plot_Data_Record;
    Legend               :        String);
```

Set the string printed in the legend for that data set.
Note that an entry can exist in the legend even if there is no name associated with the graph.

```
procedure Show_Legend
   (Data                 : access Gtk_Plot_Data_Record);
```

An entry will be made in the plot's legend for that dataset.

```
procedure Hide_Legend
   (Data                 : access Gtk_Plot_Data_Record);
```

No entry will appear in the plot's legend for that dataset.

```
procedure Set_Legend_Precision
   (Data                 : access Gtk_Plot_Data_Record;
    Precision            :        Gint);
```

Number of digits to display when the legends is associated with values,
as is the case for gradients.

```
function Get_Legend_Precision
   (Data                 : access Gtk_Plot_Data_Record)
    return Gint;
```

Return the number of digits used for values in the legend

79.3.7 Gradients

The symbols displayed in the plot can be assigned specific colors. But they can also compute their own color by picking it in a gradient, depending on the value.

procedure Reset_Gradient
```
(Data                  : access Gtk_Plot_Data_Record);
```
Reset the gradient to its default value

procedure Reset_Gradient_Colors
```
(Data                  : access Gtk_Plot_Data_Record);
```
Reset the colors of the gradient to their default values

procedure Set_Gradient_Mask
```
(Data                  : access Gtk_Plot_Data_Record;
 Mask                  :        Plot_Gradient);
```
Indicates which component of the colors vary along the gradient.

function Get_Gradient_Mask
```
(Data                  : access Gtk_Plot_Data_Record)
 return Plot_Gradient;
```
Return the mask used for the gradient.

procedure Gradient_Set_Visible
```
(Data                  : access Gtk_Plot_Data_Record;
 Visible               :        Boolean);
```
Indicates whether the gradient should be visible

function Gradient_Visible
```
(Data                  : access Gtk_Plot_Data_Record)
 return Boolean;
```
Return True if the gradient is currently visible

procedure Set_Gradient_Colors
```
(Data                  : access Gtk_Plot_Data_Record;
 Min, Max              :        Gdk.Color.Gdk_Color);
```
Set the colors that define the gradient. The colors will vary from Min to Max along the components specified in Set_Gradient_Mask.

procedure Get_Gradient_Colors
```
(Data                  : access Gtk_Plot_Data_Record;
 Min, Max              : out    Gdk.Color.Gdk_Color);
```
Return the colors that define the range

procedure Set_Gradient_Nth_Color
```
(Data          : access Gtk_Plot_Data_Record;
 Level         :        Guint;
 Color         :        Gdk.Color.Gdk_Color);
```
Set the nth color in the gradient

function Get_Gradient_Nth_Color
```
(Data          : access Gtk_Plot_Data_Record;
 Level         :        Guint)
 return Gdk.Color.Gdk_Color;
```
Get the nth color of the gradient

procedure Set_Gradient_Outer_Colors
```
(Data          : access Gtk_Plot_Data_Record;
 Min, Max      :        Gdk.Color.Gdk_Color);
```
Set the outer colors for the gradient

procedure Set_Gradient
```
(Data          : access Gtk_Plot_Data_Record;
 Min, Max      :        Gdouble;
 Nlevels       :        Gint);
```
Define the values associated with the minimal color and the maximal
color. Any value in between will have a color computed in between. Nlevels is the number of ticks to display in the gradient.

procedure Get_Gradient
```
(Data          : access Gtk_Plot_Data_Record;
 Min, Max      : out    Gdouble;
 Nlevels       : out    Gint);
```
Return the values associated with the minimal and maximal colors.

procedure Get_Gradient_Level
```
(Data          : access Gtk_Plot_Data_Record;
 Level         :        Gdouble;
 Color         : out    Gdk.Color.Gdk_Color);
```
Return the color associated with a specific level.
The color depends on the parameters to Set_Gradient and Set_Gradient_Colors.

procedure Draw_Gradient
```
(Data          : access Gtk_Plot_Data_Record;
 X, Y          :        Gint);
```
Draw the gradient ast specific coordinates

procedure Gradient_Autoscale_A

```
       (Data                : access Gtk_Plot_Data_Record);
```
???

procedure Gradient_Autoscale_Da
```
       (Data                : access Gtk_Plot_Data_Record);
```
???

procedure Gradient_Autoscale_Z
```
       (Data                : access Gtk_Plot_Data_Record);
```
???

procedure Gradient_Set_Style
```
       (Data                : access Gtk_Plot_Data_Record;
        Style               :        Plot_Label_Style;
        Precision           :        Gint);
```
???

procedure Gradient_Set_Scale
```
       (Data                : access Gtk_Plot_Data_Record;
        Scale               :        Plot_Scale);
```
Set the scale of the gradient

79.3.8 Markers

function Add_Marker
```
       (Data                : access Gtk_Plot_Data_Record;
        Point               :        Guint)
        return Gtk_Plot_Marker;
```
Add a new marker

procedure Remove_Marker
```
       (Data                : access Gtk_Plot_Data_Record;
        Marker              :        Gtk_Plot_Marker);
```
Remove a marker from the plot

procedure Remove_Markers
```
       (Data                : access Gtk_Plot_Data_Record);
```
Remove all markers

procedure Show_Markers
```
       (Data                : access Gtk_Plot_Data_Record;
        Show                :        Boolean);
```
Whether markers should be shown

```
function Markers_Visible
        (Data                : access Gtk_Plot_Data_Record)
         return Boolean;
```

Whether markers are currently visible

79.3.9 User Data

It is possible to associated your own user data with a plot. This is the mechanism provided by the C version of gtkextra. However, the best way to do this in Ada is to inherit from Gtk_Plot_Data_Record (or one of its children), and add your own fields.

```
procedure Set_Link
        (Data                : access Gtk_Plot_Data_Record;
         Link                :        System.Address);
```

Associate some user data with Data.
It is the responsability of the user to do some convert conversion to System.Address.

```
function Get_Link
        (Data                : access Gtk_Plot_Data_Record)
         return System.Address;
```

Return the user data associated with Data, or Null_Address if there is none.

```
procedure Remove_Link
        (Data                : access Gtk_Plot_Data_Record);
```

Remove the user data associated with Data.

80 Package Gtk.Extra.Plot_Polar

This special type of data set displays itself in polar coordinates.

80.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Plot             (Package Gtk.Extra.Plot)
         \___ Gtk_Plot_Polar    (Package Gtk.Extra.Plot_Polar)
```

80.2 Subprograms

```
procedure Gtk_New
       (Polar                : out   Gtk_Plot_Polar;
        Drawable             :       Gdk.Drawable.Gdk_Drawable
                               := null;
        Width, Height        :       Gdouble := 0.0);
```

Create a new polar plot.
If Width and Height are 0, they are left unspecified when calling the C function.

```
function Get_Type              return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Box

```
procedure Rotate
       (Polar                : access Gtk_Plot_Polar_Record;
        Angle                :       Gdouble);
```

Rotate the graph by a given amount of radians.

```
function Get_Angle
       (Polar                : access Gtk_Plot_Polar_Record)
        return Gdouble;
```

Return the current angle for the polar plot

81 Package Gtk.Extra.Plot_Ps

This package does not implement any new widget. Instead, if provides postscript support for Gtk_Plot widgets, and can create a postscript file from any Gtk_Plot widget.

81.1 Types

```
type Ps_Orientation is
        (Plot_Portrait, Plot_Landscape);
```

Portrait format means that the vertical size is longer than the horizontal size. Landscape is the reverse. pragma Convention (C, Ps_Orientation);

```
type Ps_Page_Size is
        (Plot_Letter,
         Plot_Legal,
         Plot_A4,
         Plot_Executive,
         Plot_Custom);
```

The formats that can be used for paper sizes. pragma Convention (C, Ps_Page_Size);

```
type Ps_Units is
        (Plot_Inches, Plot_Mm, Plot_Cm, Plot_Pspoints);
```

Units of measure for paper sizes. pragma Convention (C, Ps_Units);

81.2 Subprograms

```
procedure Plot_Export_Ps
       (Plot              : access Gtk.Extra.Plot.Gtk_Plot_Record'Class;
        Psfile            : in     String;
        Orientation       : in     Ps_Orientation;
        Epsflag           : in     Boolean;
        Page_Size         : in     Ps_Page_Size);
```

Create a new postscript file PsFile with the content of Plot.
Epsflag should be true if the generated file should be in Encapsulated Postscript format instead of simple Postscript.

```
procedure Plot_Export_Ps_With_Size
       (Plot              : access Gtk.Extra.Plot.Gtk_Plot_Record'Class;
        Psfile            : in     String;
        Orientation       : in     Ps_Orientation;
        Epsflag           : in     Boolean;
        Units             : in     Ps_Units;
```

```
     Width              : in    Gint;
     Height             : in    Gint);
```

Create a new postscript file PsFile with the content of Plot.
Epsflag should be true if the generated file should be in Encapsulated Postscript format instead of simple Postscript. The page has a custom size.

procedure Plot_Canvas_Export_Ps
```
    (Canvas             : access Gtk_Plot_Canvas_Record'Class;
     File_Name          : in    String;
     Orientation        : in    Ps_Orientation;
     Epsflag            : in    Boolean;
     Page_Size          : in    Ps_Page_Size);
```

Create a new postscript file PsFile with the content of Canvas.
Every plot on it is exported to the postscript file. Epsflag should be true if the generated file should be in Encapsulated Postscript format instead of simple Postscript.

procedure Plot_Canvas_Export_Ps_With_Size
```
    (Canvas             : access Gtk_Plot_Canvas_Record'Class;
     File_Name          : in    String;
     Orientation        : in    Ps_Orientation;
     Epsflag            : in    Boolean;
     Units              : in    Ps_Units;
     Width              : in    Gint;
     Height             : in    Gint);
```

Create a new postscript file PsFile with the content of Canvas.
Every plot on it is exported to the postscript file. Epsflag should be true if the generated file should be in Encapsulated Postscript format instead of simple Postscript. The page has a custom size.

82 Package Gtk.Extra.Plot_Surface

A special kind of data set that stores three-dimensional data.

82.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Plot_Data          (Package Gtk.Extra.Plot_Data)
         \___ Gtk_Plot_Surface    (Package Gtk.Extra.Plot_Surface)
```

82.2 Subprograms

procedure Gtk_New
```
(Widget              : out   Gtk_Plot_Surface;
 Func                :       Gtk.Extra.Plot.Plot3D_Function
                      := null);
```

Create a new surface.
If Func is null, you have to explicitely specify the set of points found in the data set. Otherwise, the points will be generated automatically from Func.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with this widget.

function Get_Grid_Visible
```
(Data                : access Gtk_Plot_Surface_Record)
 return Boolean;
```

Return True if the grid is currently visible for this data set.

procedure Set_Grid_Visible
```
(Data                : access Gtk_Plot_Surface_Record;
 Visible             :        Boolean);
```

Choose whether the grid should be visible

function Get_Mesh_Visible
```
(Data                : access Gtk_Plot_Surface_Record)
 return Boolean;
```

Return True if the wireframe mesh should be drawn.

procedure Set_Mesh_Visible
```
(Data                : access Gtk_Plot_Surface_Record;
 Visible             :        Boolean);
```

Indicate whether the wireframe mesh should be visible.

```
function Get_Nx
       (Data                 : access Gtk_Plot_Surface_Record)
        return Gint;
```

```
procedure Set_Nx
       (Data                 : access Gtk_Plot_Surface_Record;
        Nx                   :        Gint);
```

```
procedure Set_Points
       (Data                 : access Gtk_Plot_Surface_Record;
        X                    :        Gdouble_Array_Access;
        Y                    :        Gdouble_Array_Access;
        Z                    :        Gdouble_Array_Access;
        Dx                   :        Gdouble_Array_Access;
        Dy                   :        Gdouble_Array_Access;
        Dz                   :        Gdouble_Array_Access);
```

Set all the values of the data set at once.

```
procedure Get_Points
       (Data                 : access Gtk_Plot_Surface_Record;
        X                    : out    Points_Array;
        Y                    : out    Points_Array;
        Z                    : out    Points_Array;
        Dx                   : out    Points_Array;
        Dy                   : out    Points_Array;
        Dz                   : out    Points_Array);
```

Return the values contained in the data set.

```
function Get_X
       (Data                 : access Gtk_Plot_Surface_Record)
        return Points_Array;
```

```
function Get_Z
       (Data                 : access Gtk_Plot_Surface_Record)
        return Points_Array;
```

Return the values contained in the data set.

```
function Get_Xstep
       (Data                 : access Gtk_Plot_Surface_Record)
        return Gdouble;
```

procedure Set_Xstep
```
(Data                 : access Gtk_Plot_Surface_Record;
 Step                 :        Gdouble);
```

procedure Set_Color
```
(Data                 : access Gtk_Plot_Surface_Record;
 Color                :        Gdk.Color.Gdk_Color);
```
Set the color to use for the surface

procedure Set_Grid_Background
```
(Data                 : access Gtk_Plot_Surface_Record;
 Background           :        Gdk.Color.Gdk_Color);
```
Set the background color to use for the grid

procedure Set_Shadow
```
(Data                 : access Gtk_Plot_Surface_Record;
 Color                :        Gdk.Color.Gdk_Color);
```
Set the color to use for the shadows.

procedure Set_Grid_Foreground
```
(Data                 : access Gtk_Plot_Surface_Record;
 Foreground           :        Gdk.Color.Gdk_Color);
```
Set the foreground color to use for the grid

procedure Set_Transparent
```
(Data                 : access Gtk_Plot_Surface_Record;
 Transparent          :        Boolean);
```
Whether the plot is transparent. If Transparent is true, all background attributes are ignored

procedure Build_Mesh
```
(Data                 : access Gtk_Plot_Surface_Record);
```
???

procedure Recalc_Nodes
```
(Data                 : access Gtk_Plot_Surface_Record);
```
???

82.2.1 Lightning model

procedure Set_Ambient
```
(Data                 : access Gtk_Plot_Surface_Record;
```

```
         Ambient              :      Gdouble);
```

Set the ambient

procedure Set_Light

```
        (Data                 : access Gtk_Plot_Surface_Record;
         X, Y, Z              :      Gdouble);
```

Set the orientation of the light vector

procedure Use_Height_Gradient

```
        (Data                 : access Gtk_Plot_Surface_Record;
         Use_Gradient         :      Boolean);
```

???

procedure Use_Amplitud

```
        (Data                 : access Gtk_Plot_Surface_Record;
         Amplitud             :      Boolean);
```

83 Package Gtk.Extra.Sheet

A Gtk_Sheet is a table like the one you can find in most spreadsheets. Each cell can contain some text or any kind of widgets.

83.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Sheet         (Package Gtk.Extra.Sheet)
```

83.2 Signals

- **"activate"**

  function Handler (Sheet : access Gtk_Sheet_Record'Class;
  Row : Gint;
  Column : Gint)
  return Boolean;

  Emitted when the user wants to activate a specific cell. The callback should return True if the cell can be activated, False otherwise. See the subprogram Activate_Cell.

- **"changed"**

  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
  Row : Gint;
  Column : Gint);

  Emitted when the content of the cell is modified (either the text itself, or its properties, alignment,...) A value of -1 for Row or Column means the row title, the column title, or their intersection.

- **"clear_cell"**

  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
  Row : Gint;
  Column : Gint);

  Emitted when the content of the cell has been deleted (the text is now the empty string).

- **"clip_range"**

  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
  Clip_Range : Gtk_Sheet_Range);

  Emitted when the clip area is set to a new value.

- **"deactivate"**

  ```
  function Handler (Sheet : access Gtk_Sheet_Record'Class;
                    Row : Gint;
                    Column : Gint)
                    return Boolean;
  ```

  Emitted when the user wants to deactivate a specific cell. The callback should return True if the cell can be deactivated, False otherwise. See the subprogram Deactivate_Cell.

- **"move_range"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Old_Range : Gtk_Sheet_Range;
                     New_Range : Gtk_Sheet_Range);
  ```

  Emitted when the current range of selected cell is moved (ie the top-left cell is changed, but the size is not modified).

- **"new_column_width"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Column : Gint;
                     Width : Guint);
  ```

  Emitted whenever the width of the column is changed, either by the user or automatically if the cells should automatically resize themselves depending on their contents).

- **"new_row_height"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Row : Gint;
                     Height : Guint);
  ```

  Emitted whenever the height of the row is changed.

- **"resize_range"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Old_Range : Gtk_Sheet_Range;
                     New_Range : Gtk_Sheet_Range);
  ```

  Emitted when the current range of selected cell is resized (ie new cells are added to it or removed from it).

- **"select_column"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Column : Gint);
  ```

  Emitted when a new column is selected.

- **"select_range"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     The_Range : Gtk_Sheet_Range);
  ```

  Emitted when a new range of cells is selected.
- **"select_row"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Row : Gint);
  ```

  Emitted when a new row is selected.
- **"set_cell"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Row : Gint;
                     Column : Gint);
  ```

  Emitted from Hide_Active_Cell, when the cell is non-empty. ???
- **"set_scroll_adjustments"**

  ```
  procedure Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Hadj : Gtk_Adjustement;
                     Vadj : Gtk_Adjustment);
  ```

  Emitted when the adjustments used to indicate which area of the sheet is visible are set or changed. This is not called when their value is changed, only when a new one is set.
- **"traverse"**

  ```
  function Handler (Sheet : access Gtk_Sheet_Record'Class;
                     Row : Gint;
                     Column : Gint;
                     New_Row : Gint_Access;
                     New_Column : Gint_Access)
                     return Boolean;
  ```

  Emitted when the user wants to make a new cell active. The coordinates of the currently active cell are passed in (Row, Column), the coordinates of the cell that the user would like to select are passed in (New_Row, New_Column). The callback can modify the new values, and should return True if the new coordinates are accepted, False if the selection should be refused.

83.3 Types

type Gtk_Sheet_Border **is new** Integer;

Mask that indicates which borders should be visible in a cell.

type Gtk_Sheet_Child **is new** Gdk.C_Proxy;

A widget insert in the sheet. This structure includes both a widget pointer and the position in the table in which it is put.

type Sheet_State **is**
```
         (Sheet_Normal,
          Sheet_Row_Selected,
          Sheet_Column_Selected,
          Sheet_Range_Selected);
```

The state of the selection. pragma Convention (C, Sheet_State);

83.4 Subprograms

83.4.1 Creation and modification

procedure Gtk_New
```
       (Sheet                : out   Gtk_Sheet;
        Rows                 : in    Guint;
        Columns              : in    Guint;
        Title                : in    UTF8_String := "";
        Entry_Type           : in    Gtk_Type := GType_Invalid);
```

Create a new sheet with a specific number of rows and columns.
You can fully specify which type the entry used to modify the value of cells should have. The value of Entry_Type can be found by using one of the Get_Type subprograms in the GtkAda packages. The Title is internal, and does not appear on the screen.

procedure Gtk_New_Browser
```
       (Sheet                : out   Gtk_Sheet;
        Rows                 : in    Guint;
        Columns              : in    Guint;
        Title                : in    UTF8_String := "");
```

Create a new sheet browser with a specific number of rows and columns.
This is a standard Gtk_Sheet, except that it is read-only and that its cells will automatically resize themselves depending on their contents.

procedure Initialize_Browser
```
       (Sheet                : access Gtk_Sheet_Record'Class;
        Rows                 : in    Guint;
        Columns              : in    Guint;
        Title                : in    UTF8_String := "");
```

Internal initialization function.
See the section "Creating your own widgets" in the documentation.

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Sheet.

```
procedure Set_Hadjustment
       (Sheet              : access Gtk_Sheet_Record;
        Adjustment         : access Gtk_Adjustment_Record'Class);
```

Change the horizontal adjustment.
It indicates what range of columns is visible.

```
procedure Set_Vadjustment
       (Sheet              : access Gtk_Sheet_Record;
        Adjustment         : access Gtk_Adjustment_Record'Class);
```

Change the vertical adjustment.
It indicates what range of rows is visible.

```
function Get_Vadjustment
       (Sheet              : access Gtk_Sheet_Record)
        return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment used to indicate the range of visible rows.

```
function Get_Hadjustment
       (Sheet              : access Gtk_Sheet_Record)
        return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment used to indicate the range of visible columns.

```
procedure Change_Entry
       (Sheet              : access Gtk_Sheet_Record;
        Entry_Type         : in    Gtk_Type);
```

Change the type of widget used to interactively modify the value of
the cells.

```
function Get_Entry
       (Sheet              : access Gtk_Sheet_Record)
        return Gtk.Widget.Gtk_Widget;
```

Return the entry used to modify the content of the cells.

```
procedure Set_Title
       (Sheet              : access Gtk_Sheet_Record;
        Title              : in    UTF8_String);
```

Change the title of the sheet.

```
procedure Freeze
       (Sheet              : access Gtk_Sheet_Record);
```

Freeze all visual updates of the sheet, until you thaw it.
The update will occur in a more efficient way.

procedure Thaw
```
   (Sheet              : access Gtk_Sheet_Record);
```
Thaw the sheet, so that visual updates occur again.
Note that you have to call Thaw as many times as you have called Freeze to actually thaw the widget.

procedure Moveto
```
   (Sheet              : access Gtk_Sheet_Record;
    Row                : in     Gint;
    Column             : in     Gint;
    Row_Align          : in     Gfloat;
    Col_Align          : in     Gfloat);
```
Scroll the viewing area to (Row, Column).
(Row_Align, Col_Align) represent the location on the screen that the cell should appear at. (0.0, 0.0) is at the top-left of the screen, whereas (1.0, 1.0) is at the bottom-right corner. If Row or Column is negative, there is no change.

procedure Set_Background
```
   (Sheet              : access Gtk_Sheet_Record;
    Color              :        Gdk.Color.Gdk_Color);
```
Change the background color of the cells.

procedure Set_Grid
```
   (Sheet              : access Gtk_Sheet_Record;
    Color              :        Gdk.Color.Gdk_Color);
```
Set the color to use for the grid.

procedure Show_Grid
```
   (Sheet              : access Gtk_Sheet_Record;
    Show               :        Boolean);
```
Whether the grid should be made visible

function Grid_Visible
```
   (Sheet              : access Gtk_Sheet_Record)
    return Boolean;
```
Whether the grid is currently visible

83.4.2 Selection and Clipping

function Get_State
```
   (Sheet              : access Gtk_Sheet_Record)
```

```
    return Sheet_State;
```

Return the status of the selection in the sheet.

```
function Get_Range
   (Sheet              : access Gtk_Sheet_Record)
    return Gtk_Sheet_Range;
```

Return the selected range.

```
procedure Get_Visible_Range
   (Sheet              : access Gtk_Sheet_Record;
    The_Range          : out   Gtk_Sheet_Range);
```

Return the range visible on the screen.

```
procedure Set_Selection_Mode
   (Sheet              : access Gtk_Sheet_Record;
    Mode               : in    Gtk.Enums.Gtk_Selection_Mode);
```

Change the selection mode.

```
procedure Select_Column
   (Sheet              : access Gtk_Sheet_Record;
    Column             : in    Gint);
```

Replace the current selection with a specific column.
The range is highlighted.

```
procedure Select_Row
   (Sheet              : access Gtk_Sheet_Record;
    Row                : in    Gint);
```

Replace the current selection with a specific row.
The range is highlighted.

```
procedure Set_Autoresize
   (Sheet              : access Gtk_Sheet_Record;
    Autoresize         :       Boolean);
```

Whether cells should automatically resize to fit their contents

```
function Autoresize
   (Sheet              : access Gtk_Sheet_Record)
    return Boolean;
```

Whether cells automatically resize to fit their contents

```
procedure Set_Autoscroll
   (Sheet              : access Gtk_Sheet_Record;
    Autoscroll         :       Boolean);
```

Whether the sheet should automatically scroll to show the active cell at all times.

function Autoscroll
```
(Sheet              : access Gtk_Sheet_Record)
 return Boolean;
```
Whether the sheet automatically scrolls to show the active cell at all times.

procedure Set_Clip_Text
```
(Sheet              : access Gtk_Sheet_Record;
 Clip               :        Boolean);
```
Set when the text contained in the cells is automatically clipped to their width.

function Clip_Text
```
(Sheet              : access Gtk_Sheet_Record)
 return Boolean;
```
Whether the text contained in the cells is automatically clipped to their width.

procedure Set_Justify_Entry
```
(Sheet              : access Gtk_Sheet_Record;
 Justify_Entry      :        Boolean);
```
Set when the justification attribute for entries should be taken into account

function Justify_Entry
```
(Sheet              : access Gtk_Sheet_Record)
 return Boolean;
```
Whether the justification attribute is used for entries

procedure Set_Locked
```
(Sheet              : access Gtk_Sheet_Record;
 Locked             :        Boolean);
```
If Locked is true, the cells are no longer editable

function Locked
```
(Sheet              : access Gtk_Sheet_Record)
 return Boolean;
```
Whether cells are currently read-only

procedure Select_Range
```
(Sheet              : access Gtk_Sheet_Record;
```

```
         The_Range            : in    Gtk_Sheet_Range);
```

Select a new range of cells.

procedure Unselect_Range
```
        (Sheet                : access Gtk_Sheet_Record);
```

Unselect a specific range of cells.
If null is passed, the current selected range is used.

procedure Clip_Range
```
        (Sheet                : access Gtk_Sheet_Record;
         The_Range            : in    Gtk_Sheet_Range);
```

Create a new clip range, which is copied to the clipboard
That range is flashed on the screen.

procedure Unclip_Range
```
        (Sheet                : access Gtk_Sheet_Record);
```

Destroy the clip area.

function In_Clip
```
        (Sheet                : access Gtk_Sheet_Record)
         return Boolean;
```

Whether a range was copied to the clipboard

function Set_Active_Cell
```
        (Sheet                : access Gtk_Sheet_Record;
         Row                  : in    Gint;
         Column               : in    Gint)
         return Boolean;
```

Set active cell where the entry will be displayed.
Returns FALSE if the current cell can not be deactivated or if the requested cell can't be activated. Depending on the value passed to Set_Autoscroll, the sheet might be scrolled.

procedure Get_Active_Cell
```
        (Sheet                : access Gtk_Sheet_Record;
         Row                  : out    Gint;
         Column               : out    Gint);
```

Return the coordinates of the active cell.
This is the cell that the user is currently editing.

83.4.3 Columns

procedure Set_Column_Title
```
        (Sheet                : access Gtk_Sheet_Record;
         Column               : in    Gint;
```

```
     Title            : in   UTF8_String);
```

Modify the title of a column.
The first column on the left has the number 0. Note that this title does not appear on the screen, and can only be used internally to find a specific column.

```
function Get_Column_Title
    (Sheet            : access Gtk_Sheet_Record;
     Column           :        Gint)
     return UTF8_String;
```

Return the title of a specific column.

```
procedure Set_Column_Titles_Height
    (Sheet            : access Gtk_Sheet_Record;
     Height           :        Guint);
```

Modify the height of the row in which the column titles appear.

```
procedure Column_Button_Add_Label
    (Sheet            : access Gtk_Sheet_Record;
     Column           : in   Gint;
     Label            : in   UTF8_String);
```

Modify the label of the button that appears at the top of each column.

```
function Column_Button_Get_Label
    (Sheet            : access Gtk_Sheet_Record;
     Column           :        Gint)
     return UTF8_String;
```

Return the label for the button that appears at the top of each column

```
procedure Column_Button_Justify
    (Sheet            : access Gtk_Sheet_Record;
     Column           : in   Gint;
     Justification    : in   Gtk.Enums.Gtk_Justification);
```

Modify the justification for the label in the column button.

```
procedure Show_Column_Titles
    (Sheet            : access Gtk_Sheet_Record);
```

Show the row in which the column titles appear.

```
procedure Hide_Column_Titles
    (Sheet            : access Gtk_Sheet_Record);
```

Hide the row in which the column titles appear.

```
function Column_Titles_Visible
    (Sheet            : access Gtk_Sheet_Record)
```

```
    return Boolean;
```
Whether a special row is added at the top to show the title of the columns.

procedure Columns_Set_Sensitivity
```
   (Sheet           : access Gtk_Sheet_Record;
    Sensitive       : in     Boolean);
```
Modify the sensitivity of all the columns.
If Sensitive is False, the columns can not be resized dynamically. This also modifies the sensitivity of the button at the top of the columns.

procedure Column_Set_Sensitivity
```
   (Sheet           : access Gtk_Sheet_Record;
    Column          : in     Gint;
    Sensitive       : in     Boolean);
```
Modify the sensitivity of a specific column and its title button.
If Sensitive if False, the column can not be dynamically resized.

procedure Column_Set_Visibility
```
   (Sheet           : access Gtk_Sheet_Record;
    Column          : in     Gint;
    Visible         : in     Boolean);
```
Change the visibility of a column.

procedure Columns_Set_Resizable
```
   (Sheet           : access Gtk_Sheet_Record;
    Resizable       :        Boolean);
```
Whether columns are resizable

function Columns_Resizable
```
   (Sheet           : access Gtk_Sheet_Record)
    return Boolean;
```
Whether columns are resizable

procedure Column_Label_Set_Visibility
```
   (Sheet           : access Gtk_Sheet_Record;
    Column          : in     Gint;
    Visible         : in     Boolean := True);
```
Change the visibility of the label in a given column.

procedure Columns_Labels_Set_Visibility
```
   (Sheet           : access Gtk_Sheet_Record;
    Visible         :        Boolean := True);
```
Change the visibility for all the column labels.

procedure Set_Column_Width
```
(Sheet              : access Gtk_Sheet_Record;
 Column             : in    Gint;
 Width              : in    Guint);
```
Modify the width in pixels of a specific column.

function Get_Column_Width
```
(Sheet              : access Gtk_Sheet_Record;
 Column             : in    Gint)
 return Gint;
```
Return the width in pixels of the Column-nth in Sheet.

procedure Add_Column
```
(Sheet              : access Gtk_Sheet_Record;
 Ncols              : in    Guint);
```
Add some empty columns at the end of the sheet.

procedure Insert_Columns
```
(Sheet              : access Gtk_Sheet_Record;
 Col                : in    Guint;
 Ncols              : in    Guint);
```
Add Ncols empty columns just before the columns number Col.

procedure Delete_Columns
```
(Sheet              : access Gtk_Sheet_Record;
 Col                : in    Guint;
 Ncols              : in    Guint);
```
Delete Ncols columns starting from Col.

procedure Column_Set_Justification
```
(Sheet              : access Gtk_Sheet_Record;
 Column             : in    Gint;
 Justification      : in    Gtk.Enums.Gtk_Justification);
```
Set the default justification for the cells in the specific column.

function Get_Columns_Count
```
(Sheet              : access Gtk_Sheet_Record)
 return Guint;
```
Return the maximum column number of the displayed cells.

83.4.4 Rows

procedure Set_Row_Title
```
(Sheet              : access Gtk_Sheet_Record;
```

```
        Row                : in    Gint;
        Title              : in    UTF8_String);
```

Modify the title of a row.
The first row at the top has the number 0. Note that this title does not appear on the screen, and can only be used internally to find a specific row.

```
function Get_Row_Title
       (Sheet              : access Gtk_Sheet_Record;
        Row                :       Gint)
        return UTF8_String;
```

Return the title of a specific row.

```
procedure Set_Row_Titles_Width
       (Sheet              : access Gtk_Sheet_Record;
        Width              :       Guint);
```

Modify the width of the column that has the row titles.

```
procedure Row_Button_Add_Label
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in    Gint;
        Label              : in    UTF8_String);
```

Modify the label of the button that appears on the left of each row.

```
function Row_Button_Get_Label
       (Sheet              : access Gtk_Sheet_Record;
        Row                :       Gint)
        return UTF8_String;
```

Return the label for the button that appears on the left of each row.

```
procedure Row_Button_Justify
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in    Gint;
        Justification      : in    Gtk.Enums.Gtk_Justification);
```

Modify the justification for the label of the row button.

```
procedure Show_Row_Titles
       (Sheet              : access Gtk_Sheet_Record);
```

Show the column in which the row titles appear.

```
procedure Hide_Row_Titles
       (Sheet              : access Gtk_Sheet_Record);
```

Hide the column in which the row titles appear.

```
function Row_Titles_Visible
       (Sheet                : access Gtk_Sheet_Record)
        return Boolean;
```

Whether a special column is added to the left to show the title of the rows.

```
procedure Rows_Set_Sensitivity
       (Sheet                : access Gtk_Sheet_Record;
        Sensitive            : in    Boolean);
```

Modify the sensitivity of all the rows.
If Sensitive is False, the rows can not be resized dynamically. This also modifies the sensitivity of the button at the left of the row.

```
procedure Row_Set_Sensitivity
       (Sheet                : access Gtk_Sheet_Record;
        Row                  : in    Gint;
        Sensitive            : in    Boolean);
```

Modify the sensitivity of a specific row and its title button.
If Sensitive if False, the row can not be dynamically resized.

```
procedure Row_Set_Visibility
       (Sheet                : access Gtk_Sheet_Record;
        Row                  : in    Gint;
        Visible              : in    Boolean);
```

Modify the visibility of a specific row

```
procedure Row_Label_Set_Visibility
       (Sheet                : access Gtk_Sheet_Record;
        Row                  : in    Gint;
        Visible              : in    Boolean := True);
```

Change the visibility of the label in a given Row.

```
procedure Rows_Labels_Set_Visibility
       (Sheet                : access Gtk_Sheet_Record;
        Visible              :       Boolean := True);
```

Change the visibility for all the row labels.

```
procedure Rows_Set_Resizable
       (Sheet                : access Gtk_Sheet_Record;
        Resizable            :       Boolean);
```

Whether rows are resizable

```
function Rows_Resizable
       (Sheet                : access Gtk_Sheet_Record)
```

```
    return Boolean;
```

Whether rows are resizable

procedure Set_Row_Height

```
   (Sheet            : access Gtk_Sheet_Record;
    Row              : in    Gint;
    Height           : in    Guint);
```

Set the height in pixels of a specific row.

function Get_Row_Height

```
   (Sheet            : access Gtk_Sheet_Record;
    Row              : in    Gint)
    return Gint;
```

Return the height in pixels of the Row-th row in Sheet.

procedure Add_Row

```
   (Sheet            : access Gtk_Sheet_Record;
    Nrows            : in    Guint);
```

Append Nrows row at the end of the sheet.

procedure Insert_Rows

```
   (Sheet            : access Gtk_Sheet_Record;
    Row              : in    Guint;
    Nrows            : in    Guint);
```

Add Nrows empty rows just before the row number Row.

procedure Delete_Rows

```
   (Sheet            : access Gtk_Sheet_Record;
    Row              : in    Guint;
    Nrows            : in    Guint);
```

Delete Nrows rows starting from Row.

function Get_Rows_Count

```
   (Sheet            : access Gtk_Sheet_Record)
    return Guint;
```

Return the maximum row number of displayed cells.

83.4.5 Range

function Range_Get_Type **return** Gtk.Gtk_Type;

Return the internal value associate with a Gtk_Sheet_Range

```
procedure Range_Clear
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range);
```

Clear the content of the range.

```
procedure Range_Delete
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range);
```

Clear the content of the range and delete all the links (user_data)

```
procedure Range_Set_Background
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Color              : in     Gdk.Color.Gdk_Color);
```

Set the background color for the cells in a specific range.

```
procedure Range_Set_Foreground
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Color              : in     Gdk.Color.Gdk_Color);
```

Set the foreground color for the cells in a specific range.

```
procedure Range_Set_Justification
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Justification      : in     Gtk.Enums.Gtk_Justification);
```

Set the text justification for the cells in the range.

```
procedure Range_Set_Editable
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Editable           : in     Boolean);
```

Set whether the cells in the range are editable.

```
procedure Range_Set_Visible
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Visible            : in     Boolean);
```

Set whether the cells in the range are visible.

```
procedure Range_Set_Border
       (Sheet              : access Gtk_Sheet_Record;
        The_Range          : in     Gtk_Sheet_Range;
        Mask               : in     Gtk_Sheet_Border;
        Width              : in     Guint;
```

```
        Line_Style          : in    Gdk.GC.Gdk_Line_Style);
```

Set the style of the border for the cells in the range.

procedure Range_Set_Border_Color

```
       (Sheet               : access Gtk_Sheet_Record;
        The_Range           : in    Gtk_Sheet_Range;
        Color               : in    Gdk.Color.Gdk_Color);
```

Change the color for the borders of the cells in the range.

procedure Range_Set_Font

```
       (Sheet               : access Gtk_Sheet_Record;
        The_Range           : in    Gtk_Sheet_Range;
        Font                :       Pango.Font.Pango_Font_Description);
```

Change the font of the cells in the range.

83.4.6 Cells

procedure Set_Cell

```
       (Sheet               : access Gtk_Sheet_Record;
        Row                 : in    Gint;
        Col                 : in    Gint;
        Justification       : in    Gtk.Enums.Gtk_Justification;
        Text                : in    UTF8_String);
```

Set the cell contents.
Set Text to the empty string to delete the content of the cell.

procedure Set_Cell_Text

```
       (Sheet               : access Gtk_Sheet_Record;
        Row                 : in    Gint;
        Col                 : in    Gint;
        Text                : in    UTF8_String);
```

Set the cell contents.
The justification used is the previous one used in that cell.

function Cell_Get_Text

```
       (Sheet               : access Gtk_Sheet_Record;
        Row                 : in    Gint;
        Col                 : in    Gint)
        return UTF8_String;
```

Return the text put in a specific cell.
The empty string is returned if there is no text in that cell.

procedure Cell_Clear

```
       (Sheet               : access Gtk_Sheet_Record;
```

```
         Row                 : in    Gint;
         Col                 : in    Gint);
```

Clear the contents of the cell.

```
procedure Cell_Delete
        (Sheet               : access Gtk_Sheet_Record;
         Row                 : in    Gint;
         Col                 : in    Gint);
```

Clear the contents of the cell and remove the user data associated with it.

```
function Cell_Get_State
        (Sheet               : access Gtk_Sheet_Record;
         Row                 : in    Gint;
         Col                 : in    Gint)
         return Gtk.Enums.Gtk_State_Type;
```

Return the state of the cell (normal or selected).

```
procedure Get_Pixel_Info
        (Sheet               : access Gtk_Sheet_Record;
         X                   : in    Gint;
         Y                   : in    Gint;
         Row                 : out   Gint;
         Column              : out   Gint);
```

Return the row and column matching a given pixel on the screen. Constraint_Error is raised if no such cell exists.

```
procedure Get_Cell_Area
        (Sheet               : access Gtk_Sheet_Record;
         Row                 : in    Gint;
         Column              : in    Gint;
         Area                : out   Gdk.Rectangle.Gdk_Rectangle);
```

Get the area of the screen that a cell is mapped to. Constraint_Error is raised if no such cell exists;

83.4.7 Children

A Gtk_Sheet can contain some children, attached to some specific cells.

```
procedure Put
        (Sheet               : access Gtk_Sheet_Record;
         Widget              : access Gtk.Widget.Gtk_Widget_Record'Class;
         X                   : in    Gint;
         Y                   : in    Gint);
```

Put a new child at a specific location (in pixels) in the sheet.

```
procedure Attach
       (Sheet              : access Gtk_Sheet_Record;
        Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        Row                :      Gint;
        Col                :      Gint;
        Xoptions           :      Gtk.Enums.Gtk_Attach_Options
                            := Expand or Fill;
        Yoptions           :      Gtk.Enums.Gtk_Attach_Options
                            := Expand or Fill;
        Xpadding           :      Gint := 0;
        Ypadding           :      Gint := 0);
```

Attach a child to a specific Cell in the sheet.
X_Align and Y_Align should be between 0.0 and 1.0, indicating that the child should be aligned from the Left (resp. Top) to the Right (resp. Bottom) of the cell. If Row or Col is negative, the widget is attached to the row buttons or column buttons. Widget will not be moved if the cell is moved.

```
procedure Attach_Floating
       (Sheet              : access Gtk_Sheet_Record;
        Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        Row                :      Gint;
        Col                :      Gint);
```

Attach a child at the current location or (Row, Col).
If the cell is moved because of resizing or other reasons, Widget will be moved as well.

```
procedure Move_Child
       (Sheet              : access Gtk_Sheet_Record;
        Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        X                  : in    Gint;
        Y                  : in    Gint);
```

Move a child of the table to a specific location in pixels.
A warning is printed if Widget is not already a child of Sheet.

```
function Get_Child_At
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in    Gint;
        Col                : in    Gint)
        return Gtk_Sheet_Child;
```

Return the widget associated with the cell.

```
function Get_Widget
       (Child              :      Gtk_Sheet_Child)
        return Gtk.Widget.Gtk_Widget;
```

Return the widget in the child.

```
procedure Button_Attach
       (Sheet              : access Gtk_Sheet_Record;
        Widget             : access Gtk.Widget.Gtk_Widget_Record'Class;
        Row                :        Gint;
        Col                :        Gint);
```

Attach a new button in the row or column title.
One of Row or Col must be negative (but only one). This can be used to modify the standard buttons that appear at the top of each column, or on the left of each row.

83.4.8 Links / User_Data

You can associate any kind of data with a cell, just like you can associate user_data with all the widgets. Note that this uses a generic package, which must be instantiated at library level since it has internal clean up functions.

```
procedure Link_Cell
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in     Gint;
        Col                : in     Gint;
        Link               : in     Data_Type);
```

Associate some user specific data with a given cell.

```
function Get_Link
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in     Gint;
        Col                : in     Gint)
        return Data_Type_Access;
```

Return the user data associated with the cell.
null is returned if the cell has no user data.

```
procedure Remove_Link
       (Sheet              : access Gtk_Sheet_Record;
        Row                : in     Gint;
        Col                : in     Gint);
```

Delete the user data associated with the cell.

84 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.

84.1 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)
```

84.2 Subprograms

84.2.1 Operations on the dialog

```
procedure Gtk_New
       (File_Selection     : out   Gtk_File_Selection;
        Title              :       UTF8_String);
```

Create a new file selection dialog.
Title is the name of the dialog, as displayed in its title bar.

```
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);
```

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_Filename
       (File_Selection     : access Gtk_File_Selection_Record)
        return UTF8_String;
```

Get the selected file name.

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);
```
When this function is called, the dialog includes a series of buttons
for file operations (create directory, rename a file, delete a file).

procedure Hide_Fileop_Buttons
```
(File_Selection     : access Gtk_File_Selection_Record);
```
Hide the buttons for file operations.

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.

84.2.2 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.

85 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.

85.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Container          (Package Gtk.Container)
         \___ Gtk_Fixed           (Package Gtk.Fixed)
```

85.2 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);
```

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.

function Get_Has_Window

```
    (Fixed              : access Gtk_Fixed_Record)
     return Boolean;
```

Return whether a Gtk_Fixed widget is created with a separate
Gdk_Window for or not.

86 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.

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_Window       (Package Gtk.Window)
               \___ Gtk_Dialog    (Package Gtk.Dialog)
                  \___ Gtk_Font_Selection_Dialog
                          (Package Gtk.Font_Selection_Dialog)
```

86.2 Subprograms

86.2.1 Font_Selection functions

```
procedure Gtk_New
   (Widget            : out   Gtk_Font_Selection);
```

Create a new font selection widget.
It can be added to any existing container.

```
function Get_Type            return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Font_Selection.

```
function Get_Font_Name
   (Fontsel           : access Gtk_Font_Selection_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 in Pango.Font.

```
function Set_Font_Name
   (Fontsel          : access Gtk_Font_Selection_Record;
    Fontname         :        String)
    return Boolean;
```

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

```
function Get_Preview_Text
   (Fontsel          : access Gtk_Font_Selection_Record)
    return UTF8_String;
```

Return the string used to preview the selected font in the dialog.

```
procedure Set_Preview_Text
   (Fontsel          : access Gtk_Font_Selection_Record;
    Text             :        UTF8_String);
```

Set the string to use to preview the selected font.

86.2.2 Font_Selection_Dialog functions

```
procedure Gtk_New
   (Widget           : out   Gtk_Font_Selection_Dialog;
    Title            :        UTF8_String);
```

Create a new dialog to select a font.
The font selection widget has its own window, whose title is chosen by Title.

```
function Dialog_Get_Type      return Gtk.Gtk_Type;
```

Return the internal value associated with a Gtk_Font_Selection_Dialog.

```
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.

```
function Set_Font_Name
   (Fsd              : access Gtk_Font_Selection_Dialog_Record;
    Fontname         :        String)
    return Boolean;
```

Set the name and attributes of the selected font in Fontsel.
Fontname should have the standard format on X11 systems, that fully describe the family, weight, size, slant, etc. of the font.

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.

procedure Set_Preview_Text
```
(Fsd                 : access Gtk_Font_Selection_Dialog_Record;
 Text                :        UTF8_String);
```
Set the string to use to preview the selected font.

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.

87 Package Gtk.Font_Selection_Dialog

88 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.

88.1 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)
```

88.2 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 := "");
```
Change the label of the frame dynamically.
If Label is the empty string, the frame's label is deleted.

function Get_Label
```
(Frame          : access Gtk_Frame_Record)
 return UTF8_String;
```
Return the label associated with the frame.

procedure Set_Label_Widget
```
(Frame          : access Gtk_Frame_Record;
 Label_Widget   : access Gtk.Widget.Gtk_Widget_Record'Class);
```
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.

89 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.

89.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Editable         (Package Gtk.Editable)
         \___ Gtk_Entry         (Package Gtk.GEntry)
```

89.2 Signals

- **"activate"**
- **"copy_clipboard"**
- **"cut_clipboard"**
- **"delete_from_cursor"**
- **"insert_at_cursor"**
- **"move_cursor"**
- **"paste_clipboard"**
- **"populate_popup"**
- **"toggle_overwrite"**

89.3 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);
```
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.

function Get_Visibility
```
(The_Entry          : access Gtk_Entry_Record)
 return Boolean;
```
Return the visibility of the characters in the entry.

procedure Set_Invisible_Char
```
(The_Entry          : access Gtk_Entry_Record;
 Char               :        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.

function Get_Invisible_Char
```
(The_Entry          : access Gtk_Entry_Record)
 return Gunichar;
```
Retrieve the character displayed in place of the real characters
for entries with visisbility set to false. See Set_Invisible_Char.

procedure Set_Has_Frame
```
(The_Entry          : access Gtk_Entry_Record;
 Setting            :        Boolean := True);
```
Set whether the entry has a beveled frame around it.

function Get_Has_Frame
```
(The_Entry          : access Gtk_Entry_Record)
 return Boolean;
```
Return whether the entry has a beveled frame.

procedure Set_Max_Length
```
(The_Entry          : access Gtk_Entry_Record;
 Max                :        Gint);
```
Set the maximum length for the text.
The current text is truncated if needed.

function Get_Max_Length
```
(The_Entry          : access Gtk_Entry_Record)
 return Gint;
```
Return the maximum length for the text.

procedure Set_Activates_Default
```
(The_Entry          : access Gtk_Entry_Record;
 Setting            :        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.)

function Get_Activates_Default
```
(The_Entry          : access Gtk_Entry_Record)
 return Boolean;
```

Return whether the entry will activate the default widget.

procedure Set_Width_Chars
```
(The_Entry          : access Gtk_Entry_Record'Class;
 Width              :        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

function Get_Width_Chars
```
(The_Entry          : access Gtk_Entry_Record'Class)
 return Gint;
```

Return number of characters to leave space for in the entry.

procedure Set_Text
```
(The_Entry          : access Gtk_Entry_Record;
 Text               :        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.

function Get_Text
```
(The_Entry          : access Gtk_Entry_Record)
 return UTF8_String;
```

Return the current text written in the entry.

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.

90 Package Gtk.GLArea

This widget is derived from Gtk_Drawing_Area and provides an area where it is possible to use the openGL API.

90.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Drawing_Area       (Package Gtk.Drawing_Area)
         \___ Gtk_GLArea          (Package Gtk.GLArea)
```

90.2 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.

90.3 Subprograms

procedure Gtk_New
```
(Widget              : out    Gtk_GLArea;
 Attr_List           : in     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           : in     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 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.

91 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.

91.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Widget                 (Package Gtk.Widget)
      \___ Gtk_Range               (Package Gtk.Grange)
```

91.2 Types

subtype Gtk_GRange **is** Gtk_Range;

91.3 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.

92 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.

92.1 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)
```

92.2 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.

93 Package Gtk.Grange

94 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.

94.1 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)
```

94.2 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.

94.3 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);
```

Set the type of shadow to be drawn around the border of the Handle_Box.

```
function Get_Shadow_Type
       (Handle_Box          : access Gtk_Handle_Box_Record)
        return Enums.Gtk_Shadow_Type;
```

Return the shadow type of the handle box.

```
procedure Set_Handle_Position
       (Handle_Box          : access Gtk_Handle_Box_Record;
        Position            :        Enums.Gtk_Position_Type);
```

Set the side of the Handle_Box where the handle is drawn.

```
function Get_Handle_Position
       (Handle_Box          : access Gtk_Handle_Box_Record)
        return Enums.Gtk_Position_Type;
```

Return 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);
```

Set 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.

```
function Get_Snap_Edge
       (Handle_Box          : access Gtk_Handle_Box_Record)
        return Enums.Gtk_Position_Type;
```

Return the snap edge of a Handle_Box.

95 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.

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

95.1 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));
```

96 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.

96.1 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)
```

96.2 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.

97 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.

97.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Misc            (Package Gtk.Misc)
         \___ Gtk_Image        (Package Gtk.Image)
```

97.2 Types

type Gtk_Image_Type **is**
```
        (Image_Empty,
         Image_Pixmap,
         Image_Image,
         Image_Pixbuf,
         Image_Stock,
         Image_Icon_Set,
         Image_Animation);
```

97.3 Subprograms

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);
```

Create a GtkImage displaying the given animation.

```
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);
```

Set the value 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);
```

Set 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);
```

```
procedure Set
       (Image              : access Gtk_Image_Record;
        Stock_Id           :      String;
        Size               :      Gtk.Enums.Gtk_Icon_Size);
```

```
procedure Set
       (Image              : access Gtk_Image_Record;
        Icon_Set           :      Gtk.Icon_Factory.Gtk_Icon_Set;
        Size               :      Gtk.Enums.Gtk_Icon_Size);
```

```
procedure Set
       (Image              : access Gtk_Image_Record;
        Animation          :      Gdk.Pixbuf.Gdk_Pixbuf_Animation);
```

Cause the Image to display the given Animation.

```
function Get_Storage_Type
       (Image              : access Gtk_Image_Record)
        return Gtk_Image_Type;
```

```
procedure Get
       (Image              : access Gtk_Image_Record;
        Pixmap             : out   Gdk.Pixmap.Gdk_Pixmap;
        Mask               : out   Gdk.Bitmap.Gdk_Bitmap);
```

Get the values of a Gtk_Image.
Mask indicates which parts of the image should be transparent.

```
procedure Get
       (Image              : access Gtk_Image_Record;
        Val                : out   Gdk.Image.Gdk_Image;
        Mask               : out   Gdk.Bitmap.Gdk_Bitmap);
```

Get the values of a Gtk_Image.
Mask indicates which parts of the image should be transparent.

```
function Get
       (Image              : access Gtk_Image_Record)
        return Gdk.Pixbuf.Gdk_Pixbuf;
```

```
function Get
       (Image              : access Gtk_Image_Record;
        Size               : access Gtk.Enums.Gtk_Icon_Size)
        return String;
```

```
procedure Get
       (Image              : access Gtk_Image_Record;
        Icon_Set           : out    Gtk.Icon_Factory.Gtk_Icon_Set;
        Size               : out    Gtk.Enums.Gtk_Icon_Size);
```

```
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.

98 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).

98.1 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)
```

98.2 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.

procedure Initialize_With_Mnemonic
```
    (Widget             : access Gtk_Image_Menu_Item_Record'Class;
     Label              :        UTF8_String);
```
ditto.

procedure Initialize_From_Stock
```
    (Widget             : access Gtk_Image_Menu_Item_Record'Class;
     Stock_Id           :        String);
```
ditto.

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;
```

99 Package Gtk.Item

100 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)

100.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Misc                 (Package Gtk.Misc)
         \___ Gtk_Label             (Package Gtk.Label)
```

100.2 Subprograms

procedure Gtk_New
```
(Label              : out   Gtk_Label;
 Str                :       UTF8_String := "");
```
Create a new label.
Str is the string to be displayed.

procedure Gtk_New_With_Mnemonic
```
(Label              : out   Gtk_Label;
 Str                :       UTF8_String);
```
Create a new label containing the text in Str.
If characters in Str are preceded by an underscore, they are underlined indicating that they represent a keyboard accelerator called a mnemonic. The mnemonic key can be used to activate another widget, chosen automatically or explicitely using Set_Mnemonic_Widget.

procedure Initialize_With_Mnemonic
```
(Label              : access Gtk_Label_Record'Class;
 Str                :       UTF8_String);
```
Internal initialization function.

function Get_Justify
```
(Label              : access Gtk_Label_Record)
 return Enums.Gtk_Justification;
```
Returns the justification of the label. See Set_Justify.

function Get_Line_Wrap
```
(Label              : access Gtk_Label_Record)
 return Boolean;
```

Returns True if the lines of the label are automatically wrapped.

```
function Get_Selectable
       (Label                : access Gtk_Label_Record)
        return Boolean;
```

Returns True if the user can copy text from the label.

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

Return the internal value associated with a Gtk_Label.

```
function Get_Use_Markup
       (Label                : access Gtk_Label_Record)
        return Boolean;
```

Return True if the label's text will be parsed for markup.

```
function Get_Use_Underline
       (Label                : access Gtk_Label_Record)
        return Boolean;
```

Return True whether an embedded underline in the label indicates
the mnemonic accelerator keys.

```
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_VoidSymb

```
procedure Set_Text
       (Label                : access Gtk_Label_Record;
        Str                  :        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_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.

```
function Get_Text
       (Label              : access Gtk_Label_Record)
        return UTF8_String;
```

Get the current value of the text displayed in the label.

```
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);
```

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 Set_Justify
       (Label              : access Gtk_Label_Record;
        Jtype              :        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_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".

procedure Set_Selectable
```
(Label              : access Gtk_Label_Record;
 Selectable         :        Boolean);
```
Selectable labels allow the user to select text from the label,
for copy-and-paste.

procedure Set_Line_Wrap
```
(Label              : access Gtk_Label_Record;
 Wrap               :        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_Use_Markup
```
(Label              : access Gtk_Label_Record;
 Markup             :        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);
```
Sets whether the text of the label should be underlined.
If Underline is True, then Label will be underlined.

101 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.

101.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Layout        (Package Gtk.Layout)
```

101.2 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.

101.3 Subprograms

procedure Gtk_New

```
(Layout          : out   Gtk_Layout;
 Hadjustment     :       Adjustment.Gtk_Adjustment
                   := Adjustment.Null_Adjustment;
 Vadjustment     :       Adjustment.Gtk_Adjustment
                   := Adjustment.Null_Adjustment);
```

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);
```

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 Get_Size
```
(Layout            : access Gtk_Layout_Record;
 Width             : out    Guint;
 Height            : out    Guint);
```

Get the current size in pixels of the layout, as set with Set_Size

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.

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.

procedure Set_Hadjustment
```
   (Layout              : access Gtk_Layout_Record;
    Adjustment          :        Gtk.Adjustment.Gtk_Adjustment);
```
Specify a new adjustment to use for the horizontal visual area.

procedure Set_Vadjustment
```
   (Layout              : access Gtk_Layout_Record;
    Adjustment          :        Gtk.Adjustment.Gtk_Adjustment);
```
Specify a new adjustment to use for the vertical visual area.

102 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.

102.1 Types

type Idle_Callback **is access function return** Boolean;

Function that can be called automatically whenever GtkAda is not processing events. It should return True if the function should be called again as soon as possible, False if it should be unregistered.

type Idle_Handler_Id **is new** Guint;
Id for Idle handlers.

type Idle_Priority **is new** Guint;
Priorities that can be set for idle handlers. The higher the priority, the less urgent the task. Handlers whose priority is lower will be called before others.

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

Function called just before starting the main loop. This can be registered with Init_Add below.

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.

type Timeout_Callback **is access function return** Boolean;

Function that can be called automatically at precise time intervals. It should return True if the function should be called again as soon as possible, False if it should be unregistered.

type Timeout_Handler_Id **is new** Guint;
Id for Timeout handlers.

102.2 Subprograms

102.2.1 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, ...

102.2.2 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.

102.2.3 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 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.

```
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.

102.2.4 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.

```
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.

102.2.5 Idle

GtkAda gives the possibility to register Idle functions.
These are called every time there is no more event to process in the queue, and are used for instance internally to redraw widgets (so that the application keeps reacting to user input even if there is a heavy redrawing to do). The Idle function returns a boolean, which should be True if the idle remains active and should be called again, or False if the idle should be unregistered. The priority of these idle callbacks can also be modified, so that the scheduling calls one callback before another.

Two versions are given, either with a user data or with none.

```
function Idle_Add
   (Cb               : in   Idle_Callback;
    Priority         : in   Idle_Priority
```

```
                           := Priority_Default_Idle)
        return Idle_Handler_Id;
```

Register an idle callback with no user data.

```
procedure Idle_Remove
       (Id                    : in    Idle_Handler_Id);
```

Remove an idle callback, when its Id is known.

102.2.6 Timeout

A timeout is a function that is called after a specified amount of time. You can of course use Ada tasks for the same role, however this might provide an easier way of doing things.

In case your timeout function takes longer to execute than the specific delay (for instance it takes 200ms for an internal of 100ms), then no invocation is queued, and they are simply discarded. There is no queue of expired timers. On the other hand, standard events are still processed, but slowly (since they have the same priority as timeouts).

The redrawing and resizing of widgets, which are being done as idles with lower priority will not take place.

```
function Timeout_Add
       (Interval              : in    Guint32;
        Func                  :       Timeout_Callback)
        return Timeout_Handler_Id;
```

Add a new timeout. Func will be called after Interval milliseconds.
The function will be called as long as it returns True.

```
procedure Timeout_Remove
       (Id                    :       Timeout_Handler_Id);
```

Unregister a timeout function.

103 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.

104 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 Chapter 112 [Package_Gtk.Option_Menu], page 443 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.

104.1 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)
```

104.2 Types

type Gtk_Menu_Detach_Func **is access procedure**

type Gtk_Menu_Position_Func **is access procedure**

104.3 Subprograms

104.3.1 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);
```
Select a specified item in the menu.
You will almost never need this function, it is used internally by Gtk_Option_Menu. Note that the item is not considered as being pressed by the user, and thus no callback is called as a result.

function Get_Active
```
(Menu               : access Gtk_Menu_Record)
 return Gtk.Menu_Item.Gtk_Menu_Item;
```
Get the active menu item.
In a Gtk_Option_Menu, this is the item that is currently shown in the button.

procedure Set_Tearoff_State
```
(Menu               : access Gtk_Menu_Record;
 Torn_Off           :        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.

function Get_Tearoff_State
```
(Menu               : access Gtk_Menu_Record)
 return Boolean;
```
Return the tearoff status of the menu.

procedure Set_Title
```
(Menu               : access Gtk_Menu_Record;
 Title              :        UTF8_String);
```
Set the title of the menu.
Title is displayed when the menu is displayed as a tearoff menu in an independent window.

function Get_Title

```
(Menu                : access Gtk_Menu_Record)
 return UTF8_String;
```

Return the tiel of the menu.

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.

104.3.2 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;
 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.

104.3.3 Modifying the accelerators

procedure Set_Accel_Group
```
    (Menu              : access Gtk_Menu_Record;
     Accel             :        Accel_Group.Gtk_Accel_Group);
```
Set the Accel_Group that holds the global accelerators and key bindings
for the menu.

function Get_Accel_Group
```
    (Menu              : access Gtk_Menu_Record)
     return Accel_Group.Gtk_Accel_Group;
```
Get the accelerator group used to set the key bindings in 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.

104.3.4 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 Detached 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.

104.4 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;
```

105 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.

105.1 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)
```

105.2 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.

106 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.

106.1 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)
```

106.2 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);
```
Create a new Gtk_Menu_Item containing a label.
The label is created using Gtk.Label.Gtk_New_With_Mnemonic, so underscores in Label indicate the mnemonic for the menu item. Warning: the menu_item will not be properly destroyed when you remove it from its parent menu, if you created it with a non-empty Label. In this case, you first need to destroy the child of the Menu_Item, and then remove it from its parent menu.

procedure Initialize_With_Mnemonic
```
(Menu_Item          : access Gtk_Menu_Item_Record'Class;
 Label              :      UTF8_String);
```
Internal initialization procedure.

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);
```

Set the submenu underneath Menu_Item.

```
function Get_Submenu
       (Menu_Item          : access Gtk_Menu_Item_Record)
        return Gtk.Widget.Gtk_Widget;
```

Get the submenu underneath this menu item, if any, null otherwise.

```
procedure Remove_Submenu
       (Menu_Item          : access Gtk_Menu_Item_Record);
```

```
procedure Gtk_Select
       (Menu_Item          : access Gtk_Menu_Item_Record);
```

```
procedure Deselect
       (Menu_Item          : access Gtk_Menu_Item_Record);
```

```
procedure Activate
       (Menu_Item          : access Gtk_Menu_Item_Record);
```

```
procedure Right_Justify
       (Menu_Item          : access Gtk_Menu_Item_Record);
```

Deprecated. Use Set_Right_Justified with Justify = True instead.

```
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;
```

```
procedure Set_Right_Justify
       (Menu_Item          : access Gtk_Menu_Item_Record;
        Justify            :        Boolean);
```

This procedure is needed by Gate to automate the code generation.

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).

107 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.

107.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Container          (Package Gtk.Container)
         \___ Gtk_Menu_Shell      (Package Gtk.Menu_Shell)
```

107.2 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.

- **"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);

  An action signal which selects another menu item (given by direction). 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.

107.3 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.

107.3.1 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.

108 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.

108.1 Widget Hierarchy

```
Gtk_Object                     (Package Gtk.Object)
   \___ Gtk_Widget             (Package Gtk.Widget)
      \___ Gtk_Misc            (Package Gtk.Misc)
```

108.2 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);
```
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 Get_Alignment
```
       (Misc                : access Gtk_Misc_Record;
        Xalign              : out    Gfloat;
        Yalign              : out    Gfloat);
```
Return the alignment of the widget.

procedure Set_Padding
```
       (Misc                : access Gtk_Misc_Record;
        Xpad                :        Gint;
        Ypad                :        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.

procedure Get_Padding
```
       (Misc                : access Gtk_Misc_Record;
        Xpad                : out    Gint;
        Ypad                : out    Gint);
```

Return the padding of the widget.

109 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).

109.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Container        (Package Gtk.Container)
         \___ Gtk_Notebook      (Package Gtk.Notebook)
```

109.2 Signals

- **"focus_tab"**

  ???

- **"select_page"**

  ???

- **"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, ...

109.3 Types

subtype Gtk_Notebook_Page **is** Gtk.Gtk_Notebook_Page;

type Gtk_Notebook_Tab **is**

```
(Notebook_Tab_First,
 Notebook_Tab_Last);
```

109.4 Subprograms

109.4.1 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.

109.4.2 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 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            : in     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            : in     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.

109.4.3 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        : in     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.

109.4.4 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.

109.4.5 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.

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.

109.4.6 List of pages

```
function Get_Children
       (Widget             : access Gtk_Notebook_Record)
        return Page_List.Glist;
```

Return the list of all pages in the notebook.

109.4.7 GValue support

```
function Get_Notebook_Page
       (Value              :        Glib.Values.GValue)
        return Gtk_Notebook_Page;
```

Convert a Value into a notebook page.

110 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:

- 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.
- 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.
- 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.

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.

110.1 Widget Hierarchy

```
Gtk_Object                       (Package Gtk.Object)
```

110.2 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.

110.3 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.

```
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. ???

110.3.1 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:

- "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).
- "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).

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.

111 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.

111.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Old_Editable       (Package Gtk.Old_Editable)
```

111.2 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.

111.3 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.

112 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.

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_Button       (Package Gtk.Button)
               \___ Gtk_Option_Menu (Package Gtk.Option_Menu)
```

112.2 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.

```
function Get_Menu
       (Option_Menu         : access Gtk_Option_Menu_Record)
        return Gtk.Menu.Gtk_Menu;
```

Return the Gtk_Menu associated with the Gtk_Option_Menu.

```
procedure Set_Menu
       (Option_Menu         : access Gtk_Option_Menu_Record;
        Menu                : access Widget.Gtk_Widget_Record'Class);
```

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.

```
function Get_History
       (Option_Menu         : access Gtk_Option_Menu_Record)
        return Gint;
```

Return the index corresponding to the menu item selected.

procedure Set_History
```
    (Option_Menu        : access Gtk_Option_Menu_Record;
     Index              :        Gint);
```
Select the menu item specified by index making it the newly selected value for the option menu.

113 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.

113.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Container          (Package Gtk.Container)
         \___ Gtk_Paned           (Package Gtk.Paned)
```

113.2 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.

procedure Initialize_Vpaned
```
    (Widget             : access Gtk_Paned_Record'Class);
```

Internal initialization function.
See the section "Creating your own widgets" in the documentation.

procedure Initialize_Hpaned
```
   (Widget          : access Gtk_Paned_Record'Class);
```
Internal initialization function.
See the section "Creating your own widgets" in the documentation.

function Get_Type **return** Glib.GType;

Return the internal value associated with a Gtk_Paned.

function Get_Type_Vpaned **return** Glib.GType;

Return the internal value associated with a vertical Gtk_Paned.

function Get_Type_Hpaned **return** Glib.GType;

Return the internal value associated with a horizontal 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.

function Get_Position
```
(Paned          : access Gtk_Paned_Record)
 return Gint;
```
Return the position of the separator.

procedure Set_Position
```
(Paned          : access Gtk_Paned_Record;
 Position       :        Gint);
```
Change 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.

114 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.

114.1 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)
```

114.2 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.

115 Package Gtk.Progress

116 Package Gtk.Progress_Bar

The progress bar provides a convenient way of displaying a state of completion for typically lengthy tasks.

116.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Progress         (Package Gtk.Progress)
         \___ Gtk_Progress_Bar (Package Gtk.Progress_Bar)
```

116.2 Types

type Gtk_Progress_Bar_Orientation **is**
```
(Progress_Left_To_Right,
 Progress_Right_To_Left,
 Progress_Bottom_To_Top,
 Progress_Top_To_Bottom);
```

116.3 Subprograms

procedure Gtk_New
```
(Progress_Bar        : out   Gtk_Progress_Bar);
```
Create a new 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);
```
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);
```

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);
```

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);
```

Cause the progress bar to switch to a different orientation
(left-to-right, right-to-left, top-to-bottom, or bottom-to-top).

function Get_Text
```
(Progress_Bar       : access Gtk_Progress_Bar_Record)
 return UTF8_String;
```

Retrieve the text displayed superimposed on the progress bar.

function Get_Fraction
```
(Progress_Bar       : access Gtk_Progress_Bar_Record)
 return Gdouble;
```

Return the current fraction of the task that's been completed.

function Get_Pulse_Step
```
(Progress_Bar       : access Gtk_Progress_Bar_Record)
 return Gdouble;
```

Return the pulse step set with Set_Pulse_Step.

function Get_Orientation
```
(Progress_Bar       : access Gtk_Progress_Bar_Record)
 return Gtk_Progress_Bar_Orientation;
```

Return the current progress bar orientation.

117 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

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_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)
```

117.2 Subprograms

procedure Gtk_New

```
(Radio_Button      : out   Gtk_Radio_Button;
 Group             :      Widget_SList.GSlist
                    := Widget_SList.Null_List;
 Label             :      UTF8_String := "");
```

Create a new radio button, belonging to Group.
If Label is left as the empty string, then the button will not have any child and you are free to put any thing you want in it, including a pixmap. To initialize the group (when creating the first button), leave Group to the Null_List. You can later get the new group that is created with a call to the Group subprogram below.

procedure Gtk_New

```
(Radio_Button      : out   Gtk_Radio_Button;
 Group             :      Gtk_Radio_Button;
 Label             :      UTF8_String := "");
```

Create a new radio button in the same group as Group.
If Label is left as the empty string, Radio_Button is created without any child and you can put whatever you want in it, including a pixmap.

To initialize a new group (when creating the first button), you should pass it null or a button that has not been created with Gtk_New, as in the example below.

procedure Gtk_New_With_Mnemonic

```
(Radio_Button      : out   Gtk_Radio_Button;
 Group             :      Widget_SList.GSlist
```

```
                           := Widget_SList.Null_List;
        Label              :      UTF8_String);
```

Create a new Gtk_Radio_Button containing a Label. The Label is created using Gtk.Label.New_With_Mnemonic, so underscores in Label indicate the mnemonic for the button.

To initialize a new group (when creating the first button), you should pass it null or a button that has not been created with Gtk_New, as in the example below.

procedure Gtk_New_With_Mnemonic

```
       (Radio_Button       : out   Gtk_Radio_Button;
        Group              :      Gtk_Radio_Button;
        Label              :      UTF8_String);
```

Create a new radio button in the same group as Group. The label is created using Gtk.Label.New_With_Mnemonic, so underscores in Label indicate the mnemonic for the button.

To initialize a new group (when creating the first button), you should pass it null or a button that has not been created with Gtk_New, as in the example below.

procedure Initialize_With_Mnemonic

```
       (Radio_Button       : access Gtk_Radio_Button_Record'Class;
        Group              :      Widget_SList.GSlist;
        Label              :      UTF8_String);
```

procedure Initialize_With_Mnemonic

```
       (Radio_Button       : access Gtk_Radio_Button_Record'Class;
        Group              :      Gtk_Radio_Button;
        Label              :      UTF8_String);
```

function Get_Type **return** Gtk.Gtk_Type;

Return the internal value associated with a Gtk_Radio_Button.

function Get_Group

```
       (Radio_Button       : access Gtk_Radio_Button_Record)
        return Widget_SList.GSlist;
```

Return the group to which Radio_Button belongs.
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.

function Group

```
       (Radio_Button       : access Gtk_Radio_Button_Record)
        return Widget_SList.GSlist;
```

This function is deprecated.

```
procedure Set_Group
   (Radio_Button       : access Gtk_Radio_Button_Record;
    Group              :        Widget_SList.GSlist);
```

Modify the group to which the button belongs.
This will not change anything visually.

117.3 Example

```
-- This creates a group of two buttons. Note how the group is initialized.

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;
```

118 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.

118.1 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)
```

118.2 Subprograms

```
procedure Gtk_New
   (Scrolled_Window    : out   Gtk_Scrolled_Window;
    Hadjustment        :       Gtk_Adjustment
                        := Null_Adjustment;
    Vadjustment        :       Gtk_Adjustment
                        := Null_Adjustment);
```

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);
```
Set the Gtk_Adjustment for the horizontal scrollbar.

procedure Set_Vadjustment
```
(Scrolled_Window    : access Gtk_Scrolled_Window_Record;
 Vadjustment        :        Gtk_Adjustment);
```
Set the Gtk_Adjustment for the vertical scrollbar.

function Get_Hadjustment
```
(Scrolled_Window    : access Gtk_Scrolled_Window_Record)
 return Gtk_Adjustment;
```
Return the horizontal scrollbar's adjustment.
This adjustment is used to connect the horizontal scrollbar to the child widget's horizontal scroll functionality.

function Get_Vadjustment
```
(Scrolled_Window    : access Gtk_Scrolled_Window_Record)
 return Gtk_Adjustment;
```
Return the vertical scrollbar's adjustment.
This adjustment is used to connect the vertical scrollbar to the child widget's vertical scroll functionality.

procedure Set_Policy
```
(Scrolled_Window    : access Gtk_Scrolled_Window_Record;
 H_Scrollbar_Policy :        Enums.Gtk_Policy_Type;
 V_Scrollbar_Policy :        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 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);
```
Return the scrollbar policy for the horizontal and vertical scrollbars.

procedure Set_Placement
```
(Scrolled_Window    : access Gtk_Scrolled_Window_Record;
 Window_Placement   :        Gtk.Enums.Gtk_Corner_Type);
```

Determine the location of the widget with respect to the scrollbars.
The default is Corner_Top_Left.

```
function Get_Placement
       (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
        return Gtk.Enums.Gtk_Corner_Type;
```

Return the location of the widget with respect to the scrollbars.

```
procedure Set_Shadow_Type
       (Scrolled_Window    : access Gtk_Scrolled_Window_Record;
        Shadow_Type        :        Gtk.Enums.Gtk_Shadow_Type);
```

Change the type of shadow drawn around the contents of Scrolled_Window.

```
function Get_Shadow_Type
       (Scrolled_Window    : access Gtk_Scrolled_Window_Record)
        return Gtk.Enums.Gtk_Shadow_Type;
```

Return 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.

119 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://www.freedesktop.org/standards/clipboards.txt` for more information on the way selection works on X-Window systems.

119.1 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.

119.2 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.

119.3 Subprograms

119.3.1 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_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.

119.3.2 Selection_Data

```
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        : in   Selection_Data)
    return Gint;
```

Return the length of the data.

```
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.

119.3.3 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.

119.3.4 Signals

```
function Make_Atom
       (Num                :        Gulong)
        return Gdk.Types.Gdk_Atom;
```

120 Package Gtk.Separator_Menu_Item

This widget serves as separator between menu items.

120.1 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)
```

120.2 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.

121 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.

121.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Size_Group            (Package Gtk.Size_Group)
```

121.2 Types

type Property_Size_Group_Mode **is new** Size_Group_Mode_Properties.Property;

type Size_Group_Mode **is**
```
        (None, Horizontal, Vertical, Both);
```

121.3 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);
```
Change the way the group effects the size of its component widgets.

function Get_Mode

```
(Size_Group          : access Gtk_Size_Group_Record)
 return Size_Group_Mode;
```

Indicate 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.

122 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).

122.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
      \___ Gtk_Container      (Package Gtk.Container)
         \___ Gtk_Socket      (Package Gtk.Socket)
```

122.2 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.

122.3 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)));
```

123 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 Chapter 89 [Package_Gtk.GEntry], page 373 for a text editing widget without spin buttons.

123.1 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)
```

123.2 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);

123.3 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);
```

Set the adjustment settings of the spin button.

```
function Get_Adjustment
       (Spin_Button        : access Gtk_Spin_Button_Record)
        return Gtk.Adjustment.Gtk_Adjustment;
```

Return the adjustment settings of the spin button.

```
procedure Set_Digits
       (Spin_Button        : access Gtk_Spin_Button_Record;
        The_Digits         :      Guint);
```

Set number of decimals of the spin button.

function Get_Digits
```
(Spin_Button        : access Gtk_Spin_Button_Record)
 return Guint;
```
Return the number of decimals of the spin button.

procedure Set_Increments
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Step               :        Gdouble;
 Page               :        Gdouble);
```
Set increments for a single step and a page move.

procedure Get_Increments
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Step               : out    Gdouble;
 Page               : out    Gdouble);
```
Return increments for a single step and a page move.

procedure Set_Range
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Min                :        Gdouble;
 Max                :        Gdouble);
```
Set range of the spin button.

procedure Get_Range
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Min                : out    Gdouble;
 Max                : out    Gdouble);
```
Return range of the spin button.

function Get_Value
```
(Spin_Button        : access Gtk_Spin_Button_Record)
 return Gdouble;
```
Return the current value of the spin button in a float.

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_Value
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Value              :        Gdouble);
```
Set the current value of the spin button.

procedure Set_Update_Policy
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Policy             :        Gtk_Spin_Button_Update_Policy);
```
Set the update policy of the spin button.
See Gtk_Spin_Button_Update_Policy for the meaning of Policy.

function Get_Update_Policy
```
(Spin_Button        : access Gtk_Spin_Button_Record)
 return Gtk_Spin_Button_Update_Policy;
```
Return the update policy of the spin button.

procedure Set_Numeric
```
(Spin_Button        : access Gtk_Spin_Button_Record;
 Numeric            :        Boolean);
```
If Numeric is True, then only a numeric value can be typed in the text entry, otherwise also nonnumeric text.

function Get_Numeric
```
(Spin_Button        : access Gtk_Spin_Button_Record)
 return Boolean;
```
Return whether only numeric values can be typedin the text entry.

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);
```
Set whether the spin button should "wrap around" when exceeding the upper and lower limits.

function Get_Wrap
```
(Spin_Button        : access Gtk_Spin_Button_Record)
 return Boolean;
```
Return whether the spin button will "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);
```

Set the spin button to round the value to the nearest step value which is set within its adjustment settings.

function Get_Snap_To_Ticks
```
(Spin_Button         : access Gtk_Spin_Button_Record)
 return Boolean;
```
Return whether the spin button rounds 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.

124 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.

124.1 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)
```

124.2 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.

124.3 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.

124.4 Subprograms

function Convert

```
(Msg                 :      Status_Bar_Msg)
 return System.Address;
```

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);
```
Set the value of the resize_grip attribute for a given status bar.

function Get_Has_Resize_Grip
```
(Statusbar          : access Gtk_Status_Bar_Record)
 return Boolean;
```
Return the value of the resize_grip attribute for a given status bar.

125 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.

125.1 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;
```

125.2 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.

126 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.

126.1 Widget Hierarchy

```
Gtk_Object                          (Package Gtk.Object)
   \___ Gtk_Widget                  (Package Gtk.Widget)
      \___ Gtk_Container            (Package Gtk.Container)
         \___ Gtk_Table             (Package Gtk.Table)
```

126.2 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.

127 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.

127.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Widget              (Package Gtk.Widget)
      \___ Gtk_Old_Editable     (Package Gtk.Old_Editable)
         \___ Gtk_Text          (Package Gtk.Text)
```

127.2 Subprograms

```
procedure Gtk_New
       (Text                 : out   Gtk_Text;
        Hadj                 : in    Adjustment.Gtk_Adjustment
                              := Adjustment.Null_Adjustment;
        Vadj                 : in    Adjustment.Gtk_Adjustment
                              := Adjustment.Null_Adjustment);
```

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.

128 Package Gtk.Text_Attributes

This package defines the Gtk_Text_Attributes type.

128.1 Types

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

128.2 Subprograms

procedure Gtk_New
```
(Text_Attr          : out   Gtk_Text_Attributes);
```
Create a new Gtk_Text_Attributes structure.

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.

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_Font

```
       (Text_Attr          :       Gtk_Text_Attributes;
        Font               :       Pango.Font.Pango_Font_Description);
```

Set the Pango_Font_Description associated to the given
Gtk_Text_Attributes.

129 Package Gtk.Text_Buffer

This is the public representation of a text buffer to be used in cunjunction with Gtk.Text_View.

129.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Text_Buffer           (Package Gtk.Text_Buffer)
```

129.2 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);

129.3 Subprograms

```
procedure Gtk_New
   (Buffer          : out  Gtk_Text_Buffer;
    Table           :      Gtk.Text_Tag_Table.Gtk_Text_Tag_Table
                     := null);
```

Create a new text buffer.
Create a new table if Table is null. The buffer is created with a reference count of 1, and therefore should be manually unreferenced (Glib.Object.Unref). It is recommended to do that as soon as the buffer has been used to create a Gtk.Text_View.Gtk_Text_View widget.

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.

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 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);
```

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_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 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 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.

```
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 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.

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.

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));

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.

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.

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.

```
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);
```

???

```
function Get_Modified
    (Buffer              : access Gtk_Text_Buffer_Record)
     return Boolean;
```

Whether the buffer has been modified since the last call to Set_Modified set the modification flag to False. Used for example to enable a "save" function in a text editor.

```
procedure Set_Modified
    (Buffer              : access Gtk_Text_Buffer_Record;
     Setting             :        Boolean := True);
```

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 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
said text if it's 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 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.

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.

function Get_Buffer
```
    (Iter               :      Gtk.Text_Iter.Gtk_Text_Iter)
     return Gtk_Text_Buffer;
```
Return the buffer associated to the given Gtk_Text_Iterator.

130 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.

130.1 Types

type Gtk_Text_Iter **is limited private**;

type Gtk_Text_Iter_Access **is access all** Gtk_Text_Iter;

130.2 Subprograms

procedure Copy
```
   (Source           :      Gtk_Text_Iter;
    Dest             : out  Gtk_Text_Iter);
```
Create a copy of Source.

130.2.1 Convert to different kinds of index

function Get_Offset
```
   (Iter             :      Gtk_Text_Iter)
    return Gint;
```
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.

function Get_Line
```
   (Iter             :      Gtk_Text_Iter)
    return Gint;
```
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.

function Get_Line_Offset
```
   (Iter             :      Gtk_Text_Iter)
    return Gint;
```
Return the character offset of the iterator, counting from the start of
a newline-terminated line. The first character on the line has offset 0.

function Get_Line_Index
```
(Iter : Gtk_Text_Iter)
 return Gint;
```

Return the byte index of the iterator, counting from the start of a newline-terminated line. Remember that Gtk_Text_Buffer encodes text in UTF-8, and that characters can require a variable number of bytes to represent.

function Get_Visible_Line_Offset
```
(Iter : Gtk_Text_Iter)
 return Gint;
```

function Get_Visible_Line_Index
```
(Iter : Gtk_Text_Iter)
 return Gint;
```

130.2.2 Dereference operators

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.

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 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 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 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).

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).

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 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.

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.

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.

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.

130.2.3 Moving around the buffer

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 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 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 Forward_Lines
```
   (Iter               : in out Gtk_Text_Iter;
    Count              :        Gint := 1;
    Result             : out    Boolean);
```
Call Forward_Line, up to Count times.

procedure Backward_Lines
```
   (Iter               : in out Gtk_Text_Iter;
    Count              :        Gint := 1;
    Result             : out    Boolean);
```
Call Backward_Line, up to Count times.

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 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 Forward_Word_Ends

```
(Iter                : in out Gtk_Text_Iter;
 Count               :        Gint := 1;
 Result              : out   Boolean);
```

Call Forward_Word_End up to Count times.

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 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.

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 Backward_Cursor_Positions
```
(Iter                 : in out Gtk_Text_Iter;
 Count                : Gint := 1;
 Result               : out Boolean);
```

Call Backward_Cursor_Position up to Count times.

procedure Set_Offset
```
(Iter                 : in out Gtk_Text_Iter;
 Char_Offset          : Gint);
```

Set Iter to point to Char_Offset.
Char_Offset counts from the start of the entire text buffer, starting with 0.

procedure Set_Line
```
(Iter                 : in out Gtk_Text_Iter;
 Line_Number          : Gint);
```

Move iterator Iter to the start of the line Line_Number.

```
procedure Set_Line_Offset
       (Iter                : in out Gtk_Text_Iter;
        Char_On_Line        :        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.

```
procedure Set_Line_Index
       (Iter                : in out Gtk_Text_Iter;
        Byte_On_Line        :        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.

```
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.

```
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 Set_Visible_Line_Offset
       (Iter                : in out Gtk_Text_Iter;
        Char_On_Line        :        Gint);
```

Like Set_Line_Index, but the index is in visible bytes.
I.e. text with a tag making it invisible is not counted in the index.

```
procedure Set_Visible_Line_Index
       (Iter                : in out Gtk_Text_Iter;
        Byte_On_Line        :        Gint);
```

Like Set_Line_Offset, but the offset is in visible characters.
I.e. text with a tag making it invisible is not counted in the offset.

```
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.

```
procedure Forward_Search
       (Iter               :        Gtk_Text_Iter;
        Str                :        UTF8_String;
        Visible_Only       :        Boolean := False;
        Slice              :        Boolean;
        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 Visible_Only: if True, search only visible text Slice: if True, Str contains 16#FFFC# when we want to match widgets, pixbufs 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;
        Visible_Only       :        Boolean := False;
        Slice              :        Boolean;
        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.

130.2.4 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.

130.2.5 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.

```
function Get_Text_Iter
   (Val                 :      Glib.Values.GValue)
    return Gtk_Text_Iter;
```

This is the function equivalent of procedure Get_Text_Iter above. It is sometimes more convenient that then procedure, but its use is limited by the fact that the result returned can not be stored in a variable, since Gtk_Text_Iter is limited.

```
function C_Gtk_Text_Iter_Size  return Gint;
```

131 Package Gtk.Text_Mark

The Gtk_Text_Mark data type.

131.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Text_Mark           (Package Gtk.Text_Mark)
```

131.2 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);
```

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_Visible
       (Mark                : access Gtk_Text_Mark_Record)
        return Boolean;
```

Return True if the mark is visible.
i.e. a cursor is displayed for it.

```
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.

131.2.1 Converting to/from GValue

procedure Set_Text_Mark
```
(Val                : in out Glib.Values.GValue;
 Mark               : access Gtk_Text_Mark_Record);
```
Set the value of the given GValue to Mark.

function Get_Text_Mark
```
(Val                :      Glib.Values.GValue)
 return Gtk_Text_Mark;
```
Extract the iterator from the given GValue.

132 Package Gtk.Text_Tag

The Gtk_Text_Tag data type.

132.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Text_Tag              (Package Gtk.Text_Tag)
```

132.2 Signals

- **"event"**

  function Handler
  (Widget : access Gtk_Text_Tag_Record'Class;
  Event_Object : out G_Object;
  Event : Gdk.Event.Gdk_Event;
  Iter : access Gtk.Text_Iter.Gtk_Text_Iter_Record'Class)
  return Gint;

132.3 Subprograms

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);

function Get_Type **return** Glib.GType;

Return the internal value associated with this widget.

function Get_Priority
```
(Tag                 : access Gtk_Text_Tag_Record)
 return Gint;
```
Return the tag priority.

procedure Set_Priority
```
(Tag                 : access Gtk_Text_Tag_Record;
 Priority            :      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.

133 Package Gtk.Text_Tag_Table

A table is a collection of tags where you can Add, Remove, Lookup or traverse (Foreach) a tag.

133.1 Widget Hierarchy

```
Gtk_Object                         (Package Gtk.Object)
   \___ Gtk_Text_Tag_Table         (Package Gtk.Text_Tag_Table)
```

133.2 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);

133.3 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_Table_Record'Class;
         Data : Data_Type_Access);
```

133.4 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.

```
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.

134 Package Gtk.Text_View

This is the View of the Model/View/Controller text capabilities of GtkAda.

134.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Container          (Package Gtk.Container)
         \___ Gtk_Text_View       (Package Gtk.Text_View)
```

134.2 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);

134.3 Subprograms

```
procedure Gtk_New
   (Widget              : out   Gtk_Text_View;
    Buffer              :      Gtk.Text_Buffer.Gtk_Text_Buffer
                         := null);
```

Create a new Gtk_Text_View.
If Buffer is null, an empty default buffer will be created for you. Get the buffer with Get_Buffer. Otherwise, create a new text view widget displaying Buffer. One buffer can be shared among many widgets. The text view adds its own reference count to the buffer; it does not take over an existing reference.

```
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 Set_Cursor_Visible
```
       (Text_View          : access Gtk_Text_View_Record;
        Setting            :      Boolean := True);
```
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.

function Get_Cursor_Visible
```
       (Text_View          : access Gtk_Text_View_Record)
        return Boolean;
```
Whether the cursor is being displayed.

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_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);
```

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.

```
procedure Forward_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
       (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);
```

???

```
procedure Backward_Display_Line_Start
       (Text_View          : access Gtk_Text_View_Record;
        Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
        Result             : out    Boolean);
```

???

```
function Starts_Display_Line
       (Text_View          : access Gtk_Text_View_Record;
        Iter               :        Gtk.Text_Iter.Gtk_Text_Iter)
        return Boolean;
```

???

procedure Move_Visually
```
       (Text_View          : access Gtk_Text_View_Record;
        Iter               : in out Gtk.Text_Iter.Gtk_Text_Iter;
        Count              :      Gint;
        Result             : out   Boolean);
```
???

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);
```
???

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);
```
???

procedure Move_Child
```
       (Text_View          : access Gtk_Text_View_Record;
        Child              : access Gtk.Widget.Gtk_Widget_Record'Class;
        Xpos               :      Gint;
        Ypos               :      Gint);
```
???

procedure Set_Wrap_Mode
```
       (Text_View          : access Gtk_Text_View_Record;
        Wrap_Mode          :      Gtk.Enums.Gtk_Wrap_Mode);
```
Set the line wrapping for the view.

function Get_Wrap_Mode
```
       (Text_View          : access Gtk_Text_View_Record)
        return Gtk.Enums.Gtk_Wrap_Mode;
```
Get the line wrapping for the view.

procedure Set_Editable
```
       (Text_View          : access Gtk_Text_View_Record;
        Setting            :      Boolean := True);
```
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.

function Get_Editable
```
(Text_View          : access Gtk_Text_View_Record)
 return Boolean;
```
Return the default editability of the Gtk_Text_View.
Tags in the buffer may override this setting for some ranges of text.

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;
```
???

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;
```
???

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;
```
???

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;
```
???

```
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;
```
???

```
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;
```
???

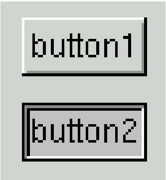
```
procedure Set_Indent
       (Text_View          : access Gtk_Text_View_Record;
        Indent             :        Gint);
```
???

```
function Get_Indent
       (Text_View          : access Gtk_Text_View_Record)
        return Gint;
```
???

135 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.



135.1 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)
```

135.2 Signals

- **"toggled"**
  procedure Handler (Toggle : access Gtk_Toggle_Button_Record'Class);

  This signal is emitted every time the state of the button is modified.

135.3 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.

procedure Initialize_With_Mnemonic
```
(Toggle_Button      : access Gtk_Toggle_Button_Record'Class;
 Label              :        UTF8_String);
```
Internal initialization function.

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);
```
Change the mode of the button.
If Draw_Indicator is False, then the button is hidden.

function Get_Mode
```
(Toggle_Button      : access Gtk_Toggle_Button_Record)
 return Boolean;
```
Return the mode of the button.

procedure Set_Active
```
(Toggle_Button      : access Gtk_Toggle_Button_Record;
 Is_Active          :        Boolean);
```
Change the state of the button.
When Is_Active is True, the button is drawn as a pressed button.

function Get_Active
```
(Toggle_Button      : access Gtk_Toggle_Button_Record)
 return Boolean;
```
Return true if the button is in its active state, i.e is pressed.

procedure Set_Inconsistent
```
(Toggle_Button      : access Gtk_Toggle_Button_Record;
 Setting            :        Boolean := True);
```
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.

function Get_Inconsistent
```
(Toggle_Button      : access Gtk_Toggle_Button_Record)
 return Boolean;
```
Get the value set by Set_Inconsistent.

135.3.1 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.

135.4 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;
```

136 Package Gtk.Toolbar

137 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.

137.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Tooltips              (Package Gtk.Tooltips)
```

137.2 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.

137.3 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 following.

--  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;
```

138 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.

138.1 Widget Hierarchy

```
Gtk_Object                      (Package Gtk.Object)
   \___ Gtk_Tree_Model          (Package Gtk.Tree_Model)
```

138.2 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

138.3 Types

type Gtk_Tree_Iter **is private**;

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;

138.4 Subprograms

138.4.1 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.

138.4.2 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 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.

138.4.3 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 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.

function Valid
```
(Reference           :       Gtk_Tree_Row_Reference)
 return Boolean;
```
Return True if Reference is non null and is still valid.

procedure Row_Reference_Free
```
(Reference           :       Gtk_Tree_Row_Reference);
```
Free the memory occupied by Reference.

138.4.4 Iterators

??? Need to be able to access the user_data fields, so that new models can define their own iterators

procedure Iter_Copy
```
   (Source          :     Gtk_Tree_Iter;
    Dest            : out  Gtk_Tree_Iter);
```
Create a copy of Source.

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_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_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.

138.4.5 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

139 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.

139.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Tree_Model            (Package Gtk.Tree_Model)
      \___ Gtk_Tree_Store         (Package Gtk.Tree_Store)
```

139.2 Types

type Data_Type **is private**;

type Data_Type_Access **is access all** Data_Type;

139.3 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.

procedure Set
```
(Tree_Store           : access Gtk_Tree_Store_Record;
 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.

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;
        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.

```
procedure Clear
   (Tree_Store      : access Gtk_Tree_Store_Record);
```

Removes all rows from Tree_Store

139.3.1 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.

139.4 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 Internal
      (Tree : System.Address;
       Iter : Gtk.Tree_Model.Gtk_Tree_Iter;
       Column1 : Gint; Value1 : UTF8_String;
       Column2 : Gint; Value2 : Gint;
       Final : Gint := -1);
    pragma Import (C, Internal, "gtk_tree_store_set");
 begin
    Internal
      (Get_Object (Tree_Store), Iter,
       Column1, Value1 & ASCII.NUL, Column2, Boolean'Pos (Value2));
 end Set;
```

140 Package Gtk.Tree_View

See extended documentation in Gtk.Tree_View_Column and Gtk.Tree_Store.

140.1 Widget Hierarchy

```
Gtk_Object                        (Package Gtk.Object)
   \___ Gtk_Widget                (Package Gtk.Widget)
      \___ Gtk_Container          (Package Gtk.Container)
         \___ Gtk_Tree_View       (Package Gtk.Tree_View)
```

140.2 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);

140.3 Types

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);
```

140.4 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);
```

```
function Get_Model
       (Tree_View           : access Gtk_Tree_View_Record)
        return Gtk.Tree_Model.Gtk_Tree_Model;
```

Returns the model the the Gtk_Tree_View is based on. Returns Null if
the model is unset.

```
procedure Set_Model
       (Tree_View           : access Gtk_Tree_View_Record;
        Model               :        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.

```
function Get_Hadjustment
       (Tree_View           : access Gtk_Tree_View_Record)
        return Gtk.Adjustment.Gtk_Adjustment;
```

Sets the Gtk_Adjustment for the current horizontal aspect.

```
procedure Set_Hadjustment
       (Tree_View           : access Gtk_Tree_View_Record;
        Adjustment          : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
```

Sets the Gtk_Adjustment for the current horizontal aspect.

```
function Get_Vadjustment
       (Tree_View           : access Gtk_Tree_View_Record)
```

```
         return Gtk.Adjustment.Gtk_Adjustment;
```
Gets the Gtk_Adjustment currently being used for the vertical aspect.

```
procedure Set_Vadjustment
        (Tree_View           : access Gtk_Tree_View_Record;
         Adjustment          : access Gtk.Adjustment.Gtk_Adjustment_Record'Class);
```
Sets the Gtk_Adjustment for the current vertical aspect.

140.4.1 Column and header operations

```
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 Set_Headers_Visible
        (Tree_View           : access Gtk_Tree_View_Record;
         Headers_Visible     :        Boolean);
```
Sets the the visibility state of the headers.

```
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);
```
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).

```
function Get_Rules_Hint
        (Tree_View           : access Gtk_Tree_View_Record)
         return Boolean;
```

Get the setting set by Set_Rules_Hint.

140.4.2 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 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 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

```
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_Reorderable
   (Tree_View          : access Gtk_Tree_View_Record;
    Reorderable        :        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.

```
function Get_Reorderable
   (Tree_View          : access Gtk_Tree_View_Record)
    return Boolean;
```

Retrieve whether the user can reorder the tree via drag-and-drop.
See Set_Reorderable.

```
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.

```
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 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             :        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.

procedure Unset_Rows_Drag_Source

```
        (Tree_View          : access Gtk_Tree_View_Record);
```

procedure Unset_Rows_Drag_Dest

```
        (Tree_View          : access Gtk_Tree_View_Record);
```

function Create_Row_Drag_Icon

```
        (Tree_View          : access Gtk_Tree_View_Record;
         Path               :        Gtk.Tree_Model.Gtk_Tree_Path)
         return Gdk.Pixmap.Gdk_Pixmap;
```

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;
```

function Get_Search_Column

```
        (Tree_View          : access Gtk_Tree_View_Record)
         return Gint;
```

procedure Set_Search_Column

```
        (Tree_View          : access Gtk_Tree_View_Record;
         Column             :        Gint);
```

141 Package Gtk.Tree_View_Column

General organization of the tree_view widgets:

____________Tree_View_______________________________ | ______________________ ______________________|
| |____Tree_View_Column1__| |___Tree_View_Column2 __|| | | | | | || | | ——— ——
—|| | || | | |Renderer1| |render2 || | || | | | | | || | || | | | | | || | || | | |
| | || | || | | |———| |———|| | || | |_____________________| |_____________________||
|____|

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).

141.1 Widget Hierarchy

```
Gtk_Object                       (Package Gtk.Object)
   \___ Gtk_Tree_View_Column     (Package Gtk.Tree_View_Column)
```

141.2 Signals

- **"clicked"**
  procedure Handler (Widget : access Gtk_Tree_View_Column_Record'Class);

141.3 Types

type Cell_Data_Func **is access procedure**

type Data_Type **is**
```
        (<>);
```

type Gtk_Tree_View_Column_Sizing **is**
```
        (Tree_View_Column_Grow_Only,
         Tree_View_Column_Autosize,
         Tree_View_Column_Fixed);
```

141.4 Subprograms

procedure Convert;

procedure Gtk_New
```
(Widget              : out   Gtk_Tree_View_Column);
```
Create a new Gtk_Tree_View_Column.

function Get_Type **return** Glib.GType;
Return the internal value associated with this widget.

141.4.1 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

141.4.2 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;
    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.

141.4.3 Options for manipulating the columns

```
procedure Set_Spacing
   (Tree_Column      : access Gtk_Tree_View_Column_Record;
    Spacing          :        Gint);
```

Set the spacing field of Tree_Column.
The spacing field is the number of pixels to place between cell renderers packed into it.

```
function Get_Spacing
   (Tree_Column      : access Gtk_Tree_View_Column_Record)
    return Gint;
```

Return the spacing of Tree_Column.

```
procedure Set_Visible
   (Tree_Column      : access Gtk_Tree_View_Column_Record;
    Visible          :        Boolean);
```

Set the visibility of Tree_Column.

```
function Get_Visible
       (Tree_Column         : access Gtk_Tree_View_Column_Record)
        return Boolean;
```

Return True if Tree_Column is visible.

```
procedure Set_Resizable
       (Tree_Column         : access Gtk_Tree_View_Column_Record;
        Resizable           :      Boolean);
```

Set whether the Tree_Column is resizable.

```
function Get_Resizable
       (Tree_Column         : access Gtk_Tree_View_Column_Record)
        return Boolean;
```

Return whether Tree_Column is resizable.

```
procedure Set_Sizing
       (Tree_Column         : access Gtk_Tree_View_Column_Record;
        The_Type            :      Gtk_Tree_View_Column_Sizing);
```

Set the growth behavior of Tree_Column to The_Type.

```
function Get_Sizing
       (Tree_Column         : access Gtk_Tree_View_Column_Record)
        return Gtk_Tree_View_Column_Sizing;
```

Return the current type of Tree_Column.

```
function Get_Width
       (Tree_Column         : access Gtk_Tree_View_Column_Record)
        return Gint;
```

Return the current size of the Tree_Column in pixels.

```
function Get_Fixed_Width
       (Tree_Column         : access Gtk_Tree_View_Column_Record)
        return Gint;
```

Get the fixed width of the column.
This value may not be the actual width of the column on the screen, just what is requested.

```
procedure Set_Fixed_Width
       (Tree_Column         : access Gtk_Tree_View_Column_Record;
        Fixed_Width         :      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.

procedure Set_Min_Width
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Min_Width          :        Gint);
```
Set the minimum width of the Tree_Column.
If Min_Width is -1, then the minimum width is unset.

function Get_Min_Width
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return Gint;
```
Return the minimum width in pixels of the Tree_Column, or -1 if no minimum width is set.

procedure Set_Max_Width
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Max_Width          :        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.

function Get_Max_Width
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return Gint;
```
Return the maximum width in pixels of the Tree_Column, or -1 if no maximum width is set.

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_Title
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Title              :        UTF8_String);
```
Set the title of the Tree_Column.
If a custom widget has been set, then this value is ignored.

function Get_Title
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return UTF8_String;
```
Return the title of the Tree_Column.

procedure Set_Clickable
```
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
```

```
         Clickable          :      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.

```
function Get_Clickable
        (Tree_Column        : access Gtk_Tree_View_Column_Record)
         return Boolean;
```

Return True if the user can click on the header for the column.

```
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);
```

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.

```
function Get_Alignment
        (Tree_Column        : access Gtk_Tree_View_Column_Record)
         return Gfloat;
```

Return the current x alignment of Tree_Column.
This value can range between 0.0 and 1.0.

```
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;
```

```
procedure Set_Sort_Column_Id
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Sort_Column_Id     :        Gint);
```

Set the logical sort_column_id that this column sorts on when this column is selected for sorting. Doing so makes the column header clickable.

```
function Get_Sort_Column_Id
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return Gint;
```

Get the logical sort_column_id that the model sorts on when this column is selected for sorting. See Gtk.Tree_View_Column.Set_Sort_Column_Id. Return value: the current sort_column_id for this column, or -1 if this column can't be used for sorting.

```
procedure Set_Sort_Indicator
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Setting            :        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.

```
function Get_Sort_Indicator
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return Boolean;
```

Get the value set by Set_Sort_Indicator.

```
procedure Set_Sort_Order
   (Tree_Column        : access Gtk_Tree_View_Column_Record;
    Order              :        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.

```
function Get_Sort_Order
   (Tree_Column        : access Gtk_Tree_View_Column_Record)
    return Gtk_Sort_Type;
```

Get the value set by Set_Sort_Order.

```
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;
```

142 Package Gtk.Type_Conversion

Provides full dynamic typing within GtkAda.

142.1 Subprograms

procedure Init;

This function has to be called to enable the full capacity for type
conversions in GtkAda. If this function is not called, then converting a C widget to an Ada type will not be as exact (for instance, most C widgets will get converted to a Gtk.Object, instead of the matching Ada widget. One major exception is Gtk_Label which will always be recognized by GtkAda). On the other hand, if you call this function (or with this package), then your application will 'with' all the GtkAda packages, involving bigger statically linked executables and a longer elaboration.

143 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.

143.1 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)
```

143.2 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.

144 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.

144.1 Widget Hierarchy

```
Gtk_Object                    (Package Gtk.Object)
   \___ Gtk_Widget            (Package Gtk.Widget)
```

144.2 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).
- **"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"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_data_delete"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_data_get"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_data_received"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_drop"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_end"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_leave"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"drag_motion"**

  Event related to drag-and-drop support. See the Gtk.Dnd documentation.

- **"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).

- **"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"**

  Related to the selection mechanism, see Gtk.Selection

- **"selection_notify_event"**

  function Handler (Widget : access Gtk_Widget_Record'Class;
  Event : Gdk.Event.Gdk_Event_Selection)
  return Boolean;

  ???

- **"selection_received"**

  Related to the selection mechanism, see Gtk.Selection

- **"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).

144.3 Types

```
type Expose_Event_Handler is access function
        (Widget : System.Address; Event : Gdk.Event.Gdk_Event) return Boolean;
```

```
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 Size_Allocate_Handler is access procedure
```

type Widget_Type **is new** Gtk_Widget_Record **with private**;

144.4 Subprograms

function Get_Type **return** Glib.GType;

Return the internal value associated with a Gtk_Widget.

144.4.1 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 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);
```
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 Get_Child_Visible
```
(Widget             : access Gtk_Widget_Record)
 return Boolean;
```
Return the value set by Set_Child_Visible

144.4.2 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 Draw
   (Widget              : access Gtk_Widget_Record;
    Area                :        Gdk.Rectangle.Gdk_Rectangle
                         := Gdk.Rectangle.Full_Area);
```

Emit a "draw" signal for a specific area of the widget.
The visual aspect might be different whether the widget has the focus or not. pragma Deprecated (Draw);

144.4.3 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);
```

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.

procedure Set_UPosition
```
(Widget             : access Gtk_Widget_Record;
 X, Y               :        Gint);
```

Modify the position of the widget.
This should be used only for toplevel widgets (windows and dialogs), since other widgets' positions are handled by their parent.

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.

144.4.4 Accelerators

procedure Add_Accelerator
```
(Widget              : access Gtk_Widget_Record;
 Accel_Signal        :      String;
 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.

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).

144.4.5 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 propagating 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 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);
```

Sets 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.

function Get_Events
```
(Widget              : access Gtk_Widget_Record)
 return Gdk.Event.Gdk_Event_Mask;
```

Get the event mask for the widget.
This indicates the list of events that the widget receives.

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);
```

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 Get_Extension_Events
```
(Widget              : access Gtk_Widget_Record)
 return Gdk.Types.Gdk_Extension_Mode;
```

Return the current extension events mask.

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.

144.4.6 Colors and colormaps

function Get_Colormap
```
(Widget              : access Gtk_Widget_Record)
 return Gdk.Color.Gdk_Colormap;
```

Return the colormap used for the widget. This will generally be the
same one for all widgets, but might be different if for instance a 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 Set_Colormap
```
(Widget              : access Gtk_Widget_Record;
 Cmap                :        Gdk.Color.Gdk_Colormap);
```

Modify the colormap of the widget.
The widget must not have been realized.

procedure Push_Colormap
```
(Cmap                :        Gdk.Color.Gdk_Colormap);
```

Modify temporarily the default colormap set for newly created widgets.
You should use this in pair with Pop_Colormap below (Push the new value, create the widget, and pop the value).

procedure Pop_Colormap;

See Push_Colormap for more information.

function Get_Default_Colormap **return** Gdk.Color.Gdk_Colormap;

Return the default colormap used when a widget is created.

procedure Set_Default_Colormap
```
        (Cmap                :       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.

function Get_Default_Visual **return** Gdk.Visual.Gdk_Visual;

Return the default visual used when a new widget is created.

144.4.7 Styles

procedure Push_Style
```
        (Style               :       Gtk.Style.Gtk_Style);
```

Change the default values for styles.
This is generally used just before creating a widget. You should use this procedure in pair with Pop_Style (Push the new value, create the widget then pop the value)

procedure Pop_Style;

Restore the default values for styles.
This is generally used just after creating a widget. You should use this procedure in pair with Push_Style (Push the new value, create the widget then pop the value)

procedure Set_Style
```
        (Widget              : access Gtk_Widget_Record;
         Style               :       Gtk.Style.Gtk_Style);
```

Set the style for a given widget.

function Get_Style
```
        (Widget              : access Gtk_Widget_Record)
         return Gtk.Style.Gtk_Style;
```

Return the style of a given widget.

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.

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.

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.

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.

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

144.4.8 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.

```
procedure Set_Parent
        (Widget              : access Gtk_Widget_Record;
         Parent              : access Gtk_Widget_Record'Class);
```

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);
```

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_Parent
        (Widget              : access Gtk_Widget_Record)
         return Gtk_Widget;
```

Return the parent of the widget, or null if Widget is a toplevel
widget.

```
function Get_Parent_Window
        (Widget              : access Gtk_Widget_Record)
         return Gdk.Window.Gdk_Window;
```

Return the widget's parent window.

```
function Get_Toplevel
        (Widget              : access Gtk_Widget_Record)
         return Gtk_Widget;
```

Return the toplevel ancestor of the widget.
This is the window or dialog in which the widget is included. The widget returned does not have any parent.

```
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.

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.

144.4.9 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.

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);
```

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.

function Get_State

```
(Widget              : access Gtk_Widget_Record)
 return Enums.Gtk_State_Type;
```

Return the state of the widget.

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);
```
Set the Gdk window associated with the widget.

function Get_Window
```
(Widget             : access Gtk_Widget_Record)
 return Gdk.Window.Gdk_Window;
```
Get 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.

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.

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.

144.4.10 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.

144.4.11 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.

- "Toplevel": Set if the widget is a toplevel widget, ie has no parent. This is mostly true for windows and dialogs.
- "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.
- "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
- "Mapped": Set if the widget is visible on the screen. This is only possible if the Visible flag is also set.
- "Visible": Set if the widget will be displayed on the screen when mapped (see the functions Show and Hide in this package).
- "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)
- "Parent_Sensitive": Set if the parent is sensitive. A widget is sensitive only if both the Sensitive and Parent_Sensitive are set.
- "Can_Focus": Set if the widget can have the focus, ie get keyboard events. Most widgets can not have the focus.
- "Has_Focus": Set if the widget currently has the focus. See the function Grab_Focus in this package.
- "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.
- "Has_Default": Set if the widget is currently the default widget. See the function Grab_Default in this package.
- "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.
- "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.
- "Composite_Child": This indicates whether the widget is composed of other widgets
- "No_Reparent": This flags is never used in gtk+.
- "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.
- "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.

```
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.

144.4.12 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.

145 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.

145.1 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)
```

145.2 Signals

- **"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.

- **"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.

145.3 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);
```

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.

```
function Get_Title
       (Window            : access Gtk_Window_Record)
        return UTF8_String;
```

Return the title of the window, or "" if there is none

```
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);
```

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

procedure Add_Accel_Group
```
(Window            : access Gtk_Window_Record;
 Accel_Group       :        Gtk.Accel_Group.Gtk_Accel_Group);
```
Specify an accelerator group for the window.

procedure Remove_Accel_Group
```
(Window            : access Gtk_Window_Record;
 Accel_Group       :        Gtk.Accel_Group.Gtk_Accel_Group);
```
Remove the specified accelerator group for the window.

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 as been called, it is respected. This is the default case.

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);
```

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);
```

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_Destroy_With_Parent

```
(Window              : access Gtk_Window_Record;
 Setting             :        Boolean := True);
```

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_Resizable

```
(Window              : access Gtk_Window_Record;
 Resizable           :        Boolean := True);
```

Set whether the user can resize a window.
Windows are user resizable by default.

function Get_Resizable

```
(Window              : access Gtk_Window_Record)
 return Boolean;
```

Whether the user can resize a window.

function Get_Resizeable

```
(Window              : access Gtk_Window_Record)
 return Boolean;
```

This function is deprecated.

procedure Set_Gravity

```
(Window              : access Gtk_Window_Record;
 Gravity             :        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.

```
function Get_Gravity
        (Window              : access Gtk_Window_Record)
         return Gdk.Window.Gdk_Gravity;
```

Return the value set by Set_Gravity.

```
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_Has_Frame
        (Window              : access Gtk_Window_Record);
```

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            :        Gint;
 Top             :        Gint;
 Right           :        Gint;
 Bottom          :        Gint);
```
Change the size of the frame border.
This has only an effect for windows with frames (see Set_Has_Frame).

procedure Set_Decorated
```
(Window          : access Gtk_Window_Record;
 Setting         :        Boolean := True);
```
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);
```
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.

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.

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 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.

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.

```
function Get_Transient_For
   (Window            : access Gtk_Window_Record)
    return Gtk_Window;
```

Return the window for which this one is a temporary window.
See Set_Transient_For below for more information on transient windows. null is returned if there is no such window.

```
procedure Set_Default_Size
   (Window            : access Gtk_Window_Record;
    Width             :        Gint;
    Height            :        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 Add_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.

145.3.1 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.

```
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.

145.4 Example

```
-- This example shows how you can display a banner while your application 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;
```

146 Package Pango

This is the top level package of the Pango hierarchy.

147 Package Pango.Attributes

This package provides a set of types and subprograms to manipulate the attributes of text displayed in a pango_layout

147.1 Types

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

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

147.2 Subprograms

147.2.1 Attributes

function Attr_Underline_New
```
(Underline          :      Pango.Enums.Underline)
 return Pango_Attribute;
```
Create a new underline attribute

147.2.2 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

148 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.

148.1 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;

148.2 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. This is

procedure Set_Family

```
(Desc                 :      Pango_Font_Description;
 Name                 :      String);
```

Set the Family_Name 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)

148.2.1 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_Type**return** 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.

148.2.2 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);
```

```
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

148.2.3 Properties

See the package Glib.Properties for more information on how to use properties

function Internal_Copy
```
(F                  :      Pango_Font_Description)
 return System.Address;
```

149 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 multiline 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.

149.1 Types

type Pango_Alignment **is**
```
(Pango_Align_Left,
 Pango_Align_Center,
 Pango_Align_Right);
```

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

type Pango_Wrap_Mode **is**
```
(Pango_Wrap_Word,
 Pango_Wrap_Char);
```

149.2 Subprograms

149.2.1 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.

149.2.2 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.

149.2.3 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.

149.2.4 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.

149.2.5 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.

149.2.6 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

149.2.7 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

150 Package Pango_Layout

Index

C

G

P

Table of Contents