UNIVERSITAT ROVIRA I VIRGILI

**Proyecto Final de Carrera**
**Ingeniería Técnica en Informática de Sistemas**
**Curso 2004-05**

# Mejora y Migración a GtkAda de la librería JEWL


Alumno: **Luis Miguel Sanz Gutiérrez.**

Dirigido por: **Sergio Gómez Jiménez .**

# 1.- ÍNDICE

# 1.- OBJETIVOS DEL PROYECTO

A día de hoy en la Universitat Rovira i Virgili cuando un informático intenta realizar proyectos y practicas en Ada que requieran una interfaz grafica, la única solución que se oye es "Librerías JEWL". Ya sea por desconocimiento de otras librerías o por su sencillez de uso, sea por lo que fuere, esta decisión tiene una implicación total con el sistema operativo a utilizar : Windows.

Esta limitación u obligación no es admisible, por lo que se pretende con este proyecto, hacer que un alumno pueda hacer sus practica de Ada independientes del sistema operativo que utilice, Windows o Linux, y que su funcionalidad y aspecto sean los mismos sin escribir diferentes códigos para cada plataforma

Estamos de acuerdo que las librerías JEWL permiten crear interfaces graficas en Ada para Windows de una manera muy fácil y rápida, dando a las practicas en este potente lenguaje un aspecto más moderno, manejable y amigable visualmente. Esto a su vez, esta obligando como se ha dicho antes al trabajo bajo Windows, cosa que no es un problema, simplemente una limitación de elección y un desagravio para el conocimiento y dominio de otros sistemas operativos del mercado.

Por otro lado, estas librerías generan un número limitado de elementos gráficos que con el tiempo se van quedando cortos para las necesidades y capacidades de creación de los alumnos y del lenguaje.

Todas estas razones no solo afectan a los alumnos, sino que también limitan a los profesores a la hora de lanzar enunciados de prácticas y de enseñar de una forma más grafica que siempre es más atractiva para ellos y para el aprendizaje de los alumnos.

Por todo ello el objetivo de este proyecto podemos decir que es básicamente doble:

- Ampliar la funcionalidad de las librerías, haciéndolas mas útiles y completas para la creación de interfaces graficas de usuario

- Llevar la base de los elementos gráficos de GDI Windows a GtkAda, de manera que la librería se convertiría en una herramienta multiplataforma, apta tanto para Windows como para Linux.

Todos estos objetivos, siempre manteniendo la política de sencillez y el mismo estilo impulsado desde las JEWL originales.

# 2.- ESPECIFICACIONES DEL PROYECTO

Basándonos en los objetivos del proyecto podemos explicar las especificaciones de este proyecto, que siempre se basaran en que el usuario de las librerías originales y las nuevas no note un cambio de estilo a la hora de programar con ellas, esto implica que las nuevas funciones deberán seguir un estándar con respecto a las funciones originales.

Con respecto a la migración de las funciones a GtkAda, esto implica cambios en el interior de las funciones y paquetes, pero no ha de implicar cambio a la hora de acceder y de pasar los parámetros, porque otro subobjetivo/especificación es que los antiguos o nuevos programas en JEWL original, puedan ser compilados con las nuevas librerías y obtengan la misma funcionalidad y resultado que los originales. Todo esto nos dice que las cabeceras serán intocables, bueno, con limitaciones ya que podremos añadir nuevas funcionalidades a las llamadas ya existentes poniendo parámetros condicionales y con valor prefijado al final de las actuales.

Por otra parte con respecto a las nuevas funcionalidades, han de complementar las ya existentes, sin afectar en su funcionamiento, pero integrándose en la estructura de objetos ya existente.

Y como se ha dicho en los objetivos, todo lo anterior independientemente del sistema operativo donde lo estemos implementando ha de tener que ofrecer las mismas funcionalidades, eso sí, la independencia del sistema operativo la debemos ofrecer y garantizar desde nuestra nueva librería, ya que no podemos garantizar que el resto de programa lo sea. Para conseguir la absoluta independencia que es lo pretendible, la solución es la no invocación de funciones de sistema.

# 3.- DISEÑO

Para hablar del nuevo diseño hemos de hablar del diseño original de las librerías JEWL.

Para comenzar, hablaremos de los paquetes de apoyo del sistema JEWL, estos son básicamente 2 (JEWL y JEWL.IO).

En el caso de JEWL es un paquete que nos aporta la lista de tipos básicos de soporte (Point_Type, Angle_Type, Font_Type...) y sus operaciones para ser utilizados ya dentro de JEWL como en nuestros programas. La parte privada para el uso dentro de la librería y la publica para todo. En esta librería de cara al diseño se cree conveniente un cambio. Este cambio se refiere al tratamiento de las fuentes, al Font_Type, ahora para referenciar una fuente y cambiar la referencia esta se tiene que copiar, pero si el discriminante de la declaración no era el mismo, complica la operación, por ello se hace un cambio que en este tipo, para que las asignaciones sean directas, la forma de hacer esto es que generamos un tipo que guarda los datos, y otro que hace la referencia a ellos, y con este es con el que trabajamos. En resumen, en lugar de trabajar con el objeto, trabajamos con un puntero a el.

En el caso de JEWL.IO es un paquete que más o menos hace una sustitución directa para el paquete estándar Ada.Text_IO y sus relaciones Ada.Integer_text_IO y Ada.Float_Text_IO, lo que significa que los programas tradicionales basados en texto pueden convertirse en programas con interfaz grafica con el mínimo esfuerzo. JEWL.IO utiliza diálogos gráficos para las entradas de datos, y todas estas entradas están relanzadas a la salida estándar. De esta forma podemos pasar la parte de entrada de cualquier programa a diálogos de entrada, mientras que la salida se mantiene escrita en código basado en texto para la salida estándar. Todo esto de que las entradas se envíen a la salida estándar significa que el 'Log' de salida se mantiene exactamente equivalente a un programa basado en texto. En este paquete no hay cambio de diseño, esta bien como esta, por lo que la decisión de diseño en este caso es no tocarlo.

Para hablar de la estructura de datos de los elementos gráficos y su almacenamiento. Los objetos gráficos están estructurados en una jerarquía de "tagged record", lo que implica que hay una herencia de campos de padres a hijos. De esta forma nos basamos en la creación de un tipo básico (en este caso el Window_Type), a partir de el, podemos generar una jerarquía de objetos que van creciendo en campos a medida que los necesitan sin tener que volver a declararlos y con la ventaja de que un tipo hijo, se puede convertir en padre, porque realmente es como el padre, pero con mas campos. De todos los tipos, tenemos algunos que son abstractos y otros que no, normalmente los no instanciables se refieren a agrupación de otros tipos, bifurcaciones del árbol y no hojas, porque no son instanciables, ni nos interesa. Los abstractos se utilizan para declarar funciones de grupo, que son permitidas para todas sus hojas.

Pero cabe destacar que en esta jerarquía no se ha utilizado en toda su potencia esta técnica, sino que esta jerarquía lo único que guarda en cada elemento es un puntero a un tipo "Controlled" de Ada, esta clase de objetos, se caracterizan porque tienen un contador interno del numero de referencias al objeto y un ciclo de vida que consta de Inicialize, Adjust y Finalize, estos 3 ciclos van acompañados de llamadas implícitas del lenguaje a funciones con el mismo nombre, que podemos sobrecargar y con ello controlar el liberado de memoria no utilizada, para así tener una correcta utilización y liberación de recursos.

Por todo ello, existen 2 jerarquías de objetos, la que marca el tipo y las operaciones de las que dispone cada objeto ("tagged record") y la jerarquía de datos. Todo esta para el diseño actual de las librerías se ha mantenido, ya que la estructura de esta forma permite un crecimiento sin limites y fácil de realizar y entender. Sobre ello, se añadirán los nuevos elementos gráficos y funciones que se generen.

En el árbol de objetos de JEWL tenemos contenedores y objetos de control, esta distribución es correcta, pero de cara a desarrollar cosas con mas contenido esta jerarquía tiene deficiencias claras en el aspecto de los controles, en este proyecto se intentara solucionar alguno de ellos, como es el caso de los menús, donde solo podemos poner elementos de submenú y de acción (el cual tiene un funcionamiento extraño) y

pretende hacer de todo, por lo que se opta por la especialización de los objetos de menú, por que porqué tener check si solo queremos que actué como botón, o si queremos que al seleccionar un elemento, deseleccione otro… De esta forma añadimos elementos de menú especializados:

- Action_Menuitem_Type : Recibe 'click' encima y envía comando
- Check_Menuitem_Type : Recibe 'click', actúa y se marca o desmarca según su estado anterior
- Radio_Menuitem_type : Recibe 'click', actúa y se marca, pero desmarcando a su vez al elemento marcado ya de su grupo de acción

Con Menuitem_Type, todas estas funcionalidades implicaban una cantidad de código y control que ahora no serán necesarias.

también se nota la falta de elementos de control de 'rango' automatizados, por lo que se genera un nuevo grupo de elementos de este tipo llamado "Range_Control_Type" y unos procedimientos asociados a ellos, todos ellos comparten que son valores que van de un mínimo a un máximo con saltos constantes o controlables por interacción del usuario o por código :

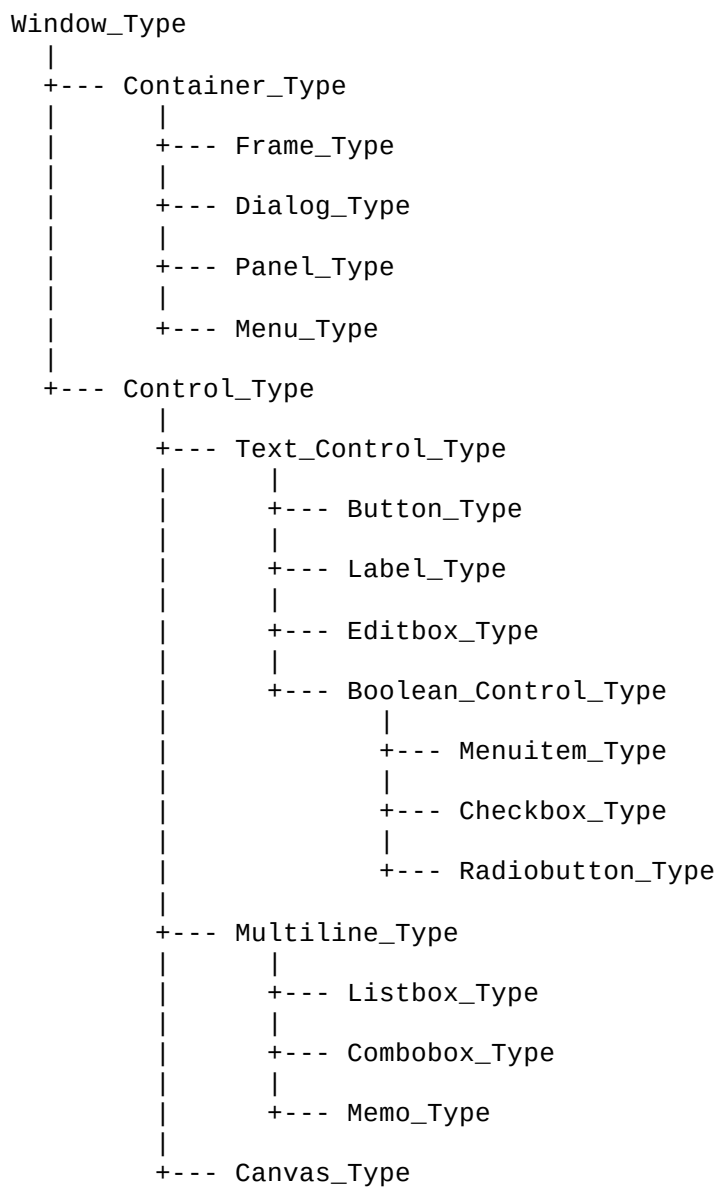
- Progressbar_Type
- Scrollbar_Type
- Scale_Type
- SpinButton_Type

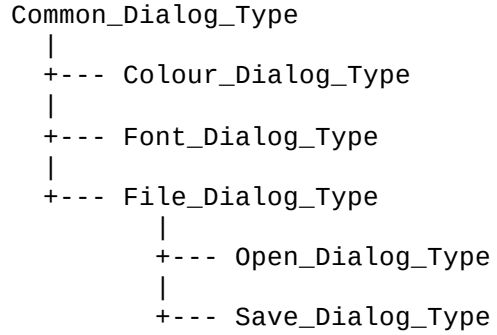
Se denota también falta de funciones sobre el texto de los contenedores, el cual es intocable, y siempre es interesante que se pueda modificar el titulo si el contenido es cambiante. En este sentido se busca ofrecer en este sentido las mismas funciones que ofrecen los Text_Control_Type.

En el sentido de dar información también denoto la falta de un elemento tan utilizado como son los 'Tooltips', esas pequeñas ayudas que sin serlo informan al usuario de la funcionalidad del elemento grafico.

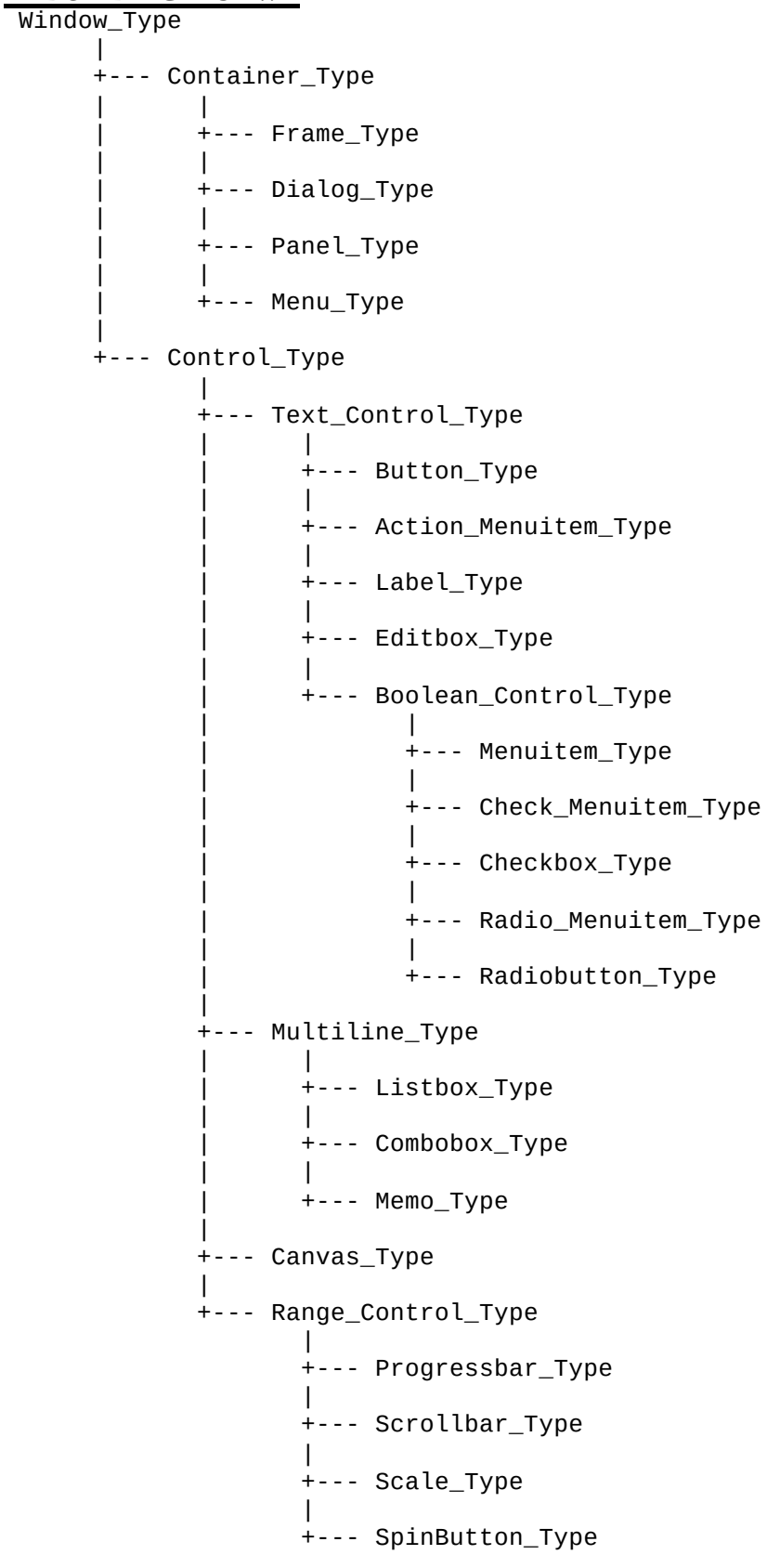
Por todo ello, pasamos del árbol original de JEWL, que es el siguiente:

***ARBOL DE JEWL***



***ARBOL DE GTKJEWL***




En árbol de Common_Dialog_Type no hay cambios

Con respecto a la creación de los objetos gráficos, desde el API de Windows todos a priori parecen del mismo tipo y tienen las mismas operaciones debido a la forma de creación. Todos ellos son creados con la misma función de sistema (CreateWindowEx), esto claramente simplifica el código de JEWL, y para hacerlo con un poco de control y generar los objetos y ciertas llamadas en orden lógico, de lo que dispone es de un "Task Type" que tiene un bucle de aceptación de llamadas, basado en la lógica de que han de existir frames, antes de poner objetos gráficos y limitaciones del estilo. En nuestro caso, en la nueva librería he prescindido de esta tarea, debido a que no era necesario para controlar lo mismo.

Con respecto a la captura de eventos, el API de Window tiene un sistema basado en mensajes de sistema y por cada manejador de contenedor que tiene quien es su hijo, este sistema en GtkAda no es así exactamente, por lo tanto para capturar los eventos que queremos no generamos unos procedimientos estándar de los contenedores, sino que generamos manejadores por cada elemento y señal que tenga que ser capturado, así no hace falta que recorra todo el árbol para llegar el evento al elemento necesario.

El tema de guardar los eventos, en JEWL se hace a través de un "Protected Type", que tiene funciones de captura, guardado y consulta comandos y sus estados, de esta forma, aseguramos la compartición de los datos y el acceso ordenado a ellos. La limitación es el buffer de un solo evento que tiene, pero aun así y todo, no he hecho cambios ya que es suficiente para un normal uso de las librerías, solo quizás en aplicaciones complejas esta forma de actuar afecte al funcionamiento.

también en referencia al tema de los comandos, no se tiene ningún control sobre ellos una vez declarado el elemento grafico y asignado su comando, pero en las aplicaciones reales se debería tener algo de control sobre ellos porque según las acciones del usuario las situaciones son cambiantes y requieren de reconfiguraciones de comandos, en este caso, no es una reconfiguración que aporte mucho, pero se pretende que si el elemento ha de dejar de emitir su comando, se pueda sin añadir código en la zona de manejador de comando, o que si ahora ha de hacer otra acción la pueda hacer, sin tener que repetir código o complicar el manejador, por ejemplo.

Como final de la exposición de las decisiones de diseño, hablaremos de unos de los elementos mas utilizados seguramente, el canvas. Este objeto que es una pizarra de dibujo y una herramienta de dibujo asociado se basa en un "Protected Type" que se encarga de recoger las acciones que se van realizando sobre el canvas y las va acumulando en una lista enlazada que sirve, para cada vez que hay que redibujar el contenido se haga un recorrido, y obtener el mismo resultado. Esta forma de actuar es discutible, debido a que provoca en general parpadeo cuando la acción grafica es constante, y el crecimiento de la lista en ciertas utilizaciones del elemento es inadecuado y puede provocar unos efectos no deseados y ralentización, a parte de un uso de memoria desproporcionado. Por ello en este elemento se decide actuar sobre su diseño.

A priori no se hacen cambios sobre la estructura de datos que por compatibilidad hacia atrás hemos de mantener la lista, lo que podemos y hacemos es modificar el funcionamiento, poniendo "double buffer", lo que evita que pintemos directamente sobre el área visible y con ello el parpadeo, por otra parte proponemos parámetros en las funciones que optimizan el funcionamiento y lo hacen mas eficiente, y hasta un nuevo estilo de canvas, el cual utilizara el algoritmo del pintor, lo ultimo que se pinta, se ve sobre lo demás (como es grafico 2D es posible), todo esto quedara explicado en la parte de desarrollo.

# 5.- DESARROLLO

Lo primero ha hacer en el desarrollo fue la parte de los paquetes de soporte, en este caso solo había un cambio a realizar y era el referente al tipo Font_Type, este tipo tenia el problema básico y que iba a dar problemas después, que era el de la asignación de fuentes, por el discriminante con respecto a la longitud del nombre de la fuente. Esto tenia una solución fácil, en lugar de trabajar con la instanciación directamente, trabajaríamos con punteros a ella, así no importaría la longitud del nombre de la fuente, ya que se había convertido en un traspaso de direcciones de memoria.

Pasamos de tener

```
type    Font_Type (Length : Natural) is
                record
                    Name   : String (1..Length);
                    Size   : Positive;
                    Bold   : Boolean := False;
                    Italic : Boolean := False;
                end record;
```

A tener :

```
type    Font_Type_Record (Length : Natural)
                  is tagged record
                    Name   : String (1..Length);
                    Size   : Positive;
                    Bold   : Boolean := False;
                    Italic : Boolean := False;
                  end record;
 type Font_Type is access Font_Type_Record'Class;
```

Con este simple cambio las actuaciones sobre las asignaciones de fuentes se simplificaban. Pero sobre ellas pesaba otro problema, y es que GtkAda no entiende este tipo de fuente, era mejor la descripción, por lo que aparecen 2 nuevas funciones sobre Font_Type, que facilitan pasar de el a una descripción y de una descripción a

Font_Type, siempre bajo un mínimo estándar de GtkAda que dice que negrita es “Bold” e itálica es “Italic”, así exactamente (Ej.: “Sans 10 Bold Italia”)

```
function Font2String (Font: Font_Type) return String;
function String2Font (Fstr: String) return Font_Type;
```

Estas funciones son de uso interno, pero por su no problemática se pone en JEWL por si el usuario querría utilizarlas.

Hasta aquí las actuaciones con respecto a los ficheros de JEWL y JEWL.IO.

Una vez definido esto paso a explicar los cambios referentes al resto de ficheros/paquetes del sistema. A priori para conseguir el objetivo del proyecto de hacer de JEWL una librería independiente de Windows, el paquete que se ha de hacer inútil y desaparecer es el de JEWL.WIN32_INTERFACE, que contiene todas las llamadas al API de Windows y todos los tipos compatibles con dicho sistema. Este fichero desaparece totalmente.

así como la interfaz con Windows y sus llamadas, también han de desaparecer los manejadores de los mensajes de Windows y sus elementos que se encuentran en el paquete JEWL.MESSAGE_HANDLING, de aquí desaparecen los mencionados manejadores de mensajes de eventos de los contenedores, así como la ‘tarea’ que se encargaba de recoger las llamadas para la creación de elementos gráficos. Dicho eso, de este paquete hemos de conservar el ‘protected type’ Window_Info, debido a que se encarga de dar la interfaz necesaria para manipular los comandos, en este caso lo movemos a otro paquete existente ( JEWL.WINDOW_IMPLEMENTATION ).

El tema de los eventos en JEWL original, podemos decir que se hace a través de los manejadores de mensajes de windows, a tiempo real. En el caso de traspasar esto a Gtk tiene su problemática. En Gtk existe el paquete Gtk.Main, dicho paquete es un listener de eventos, que se llama a través de una función que es bloqueante, y eso no nos interesa, porque el problema surge cuando como en JEWL, se quiere poder hacer cosas

mientras se comprueba si hay eventos, por lo tanto no se puede utilizar esa función. Esto nos lleva a buscar puntos en los cuales ir ejecutando eventos, para los cuales lo mejores son los puntos donde el programa pregunta si hay comando ( ejecutamos comandos pendientes sin bloquear, {Command_Ready}) o espera al siguiente (bloqueamos y ejecutamos eventos hasta que haya comando{Next_Command}), pero si no se llaman a estas funciones surge un problema (Es algo raro que no se llamen a estas funciones, pero es posible), la interfaz queda desactualizada, por lo que se genera una función que la actualiza con los eventos pendientes de ser tratados (Do_Events)

Ya que hemos introducido el paquete JEWL.WINDOW_IMPLEMENTATION, diremos que a parte de tener ahora el Window_Info, tiene en su interior las actuaciones de los Common_Dialog, y lo que es aun mas importante, los 'Controlled' que contienen los datos de los elementos de la interfaz.

De este tipo de datos no hace falta que haya tantos como elementos, ya que muchos comparten información, pero también se montan en una especie de jerarquía en función del nivel donde estén y de la funcionalidad del elemento.

En este caso la existen:

```
->  Window_Internals
      |
      +--- Container_Internals
      |      |
      |      +--- Main_Window_Internals
      |
      +--- Range_Internals
      |
      +--- Canvas_Internals


->  Common_Dialog_Internals
      |
      +--- Colour_Dialog_Internals
      |
      +--- Font_Dialog_Internals
      |
      +--- File_Dialog_Internals
             |
```

```
            +--- Open_Dialog_Internals
            |
            +--- Save_Dialog_Internals

->  Image_Internals
```

Estas estructuras que todas son provenientes de le 'Controlled', disponen cada una de los campos necesarios para cada tipo de datos. El primer conjunto se utilizan en los elementos gráficos manejables, el segundo en los diálogos no manejables y el tercero en un tipo un poco especial, las imágenes. A todo ello, ya que se pierde la referencia del tipo de datos que contiene, se les añade un atributo que nos indica esto mismo, así pues, se puede diferenciar entre 2 elementos gráficos. Esto nos servirá, ya que las funciones en general son por grupos de tipos (tipos abstractos), que engloban varios tipos, y en muchas ocasiones, para ofrecer la misma acción se han de hacer cosas diferentes. De aquí surge la opción del polimorfismo o casting, cual de las 2 escoger. En este proyecto nos encontraremos ambas opciones, y con un criterio muy sencillo, en las acciones que requieren pequeñas acciones se hace casting, pero en las acciones en las que el código para cada elemento es muy diferenciado se hace polimorfismo de funciones, con lo cual creo que ganamos en comodidad y entendimiento del código, así como modificabilidad de cara al futuro, ya que encontrar y entender ciertas funciones que son cortas, pero distribuidas en todo el código afecta la visión, en cambio cuando estas acciones ya son de por si largas, el polimorfismo no afecta en esta visión global del asunto.

Volviendo al tema de las estructuras 'Controlled', una vez eliminadas las referencias de estos objetos a elementos que sean exclusivos de windows o con tipos de windows, se pasa a la evaluación de lo que necesita cada uno de estos tipos, para dar cabida a todo lo necesario para manejar cada elemento :

- **Window_Internals** ← Hereda de Controlled
  - *Tipo* Tipo de elemento al que referencia
  - *Widget* Tipo básico de GtkAda, y que se convierte en puntero de su estructura real en GtkAda
  - *Label_Widget* Puntero a la etiqueta del elemento, si esta existe separada
  - *Changed* Estado desde la ultima consulta (para editables)

- *No_List* Nuevo campo para el funcionamiento del canvas, como lista de objetos o como lienzo donde pintar
- *Parent* Puntero al PADRE del elemento
- *Next* Puntero al siguiente HERMANO
- *First* Puntero al primer HIJO
- *Last* Puntero al ultimo HIJO
- *Top* Lo que seria la Y de la posición origen
- *Left* Lo que seria la X de la posición origen
- *Height* La altura del elemento
- *Width* El ancho del elemento
- *Font* La fuente del elemento
- *Action* código de Comando principal del elemento
- *Tooltip* Información relacionada con el Tooltip del elemento
- *Undo_Text* Puntero al buffer auxiliar para hacer UNDO

Como podemos observar, entre los atributos existen campos para realizar un árbol de elementos, este árbol es el que se utilizara en el caso de que las posiciones, tamaños y fuentes sean relativos a los padres, porque de esta forma cada vez que algo varié en un padre de elementos, puede hacer que sus hijos lo perciban y modifiquen su aspecto si es necesario.

- **Container_Internals** ← Hereda de Window_Internals
  - *Fix* Contenedor para posiciones fijas (Frame, Dialog y Panel)
  - *Menu* Contenedor de menú (Menu)

Estos campos son utilizados por los contenedores de elementos, en ellos se visualizan los hijos que tienen, que pueden ser de 2 tipos, o menús o en un panel. En JEWL siempre se habla de posiciones sobre el padre, pero GtkAda tiene muchos mas contenedores, en este caso el que nos conviene y se adapta a las necesidades es el "Fixed", en el todo tiene una posición relativa con respecto al origen (margen superior-izquierdo), sobre el se calcularan las posiciones en el caso de que no sean relativas.

- **Main_Window_Internals** ← Hereda de Container_Internals
  - *Box* Contendor con sombreado para que el aspecto sea como el de las librerías JEWL originales
  - *Vbox* Divisor vertical de la pantalla en tantos huecos como precisemos, aquí se colocan menú y contenedor de elementos central
  - *MenuBar* Contenedor para los menús, el cual esta por si en la aplicación se requieren los menús

Este tipo lo utiliza únicamente el Frame, ya que es el único que tiene la posibilidad de tener menú, y es el único que necesita una división vertical. En este aspecto se podrían haber añadido más elementos para colocar 'Toolbar' o 'Statusbar'...

- **Range_Internals** ← Hereda de Window_Internals
  - *Adj* Tipo del que dispone GtkAda que nos permite unificar todos los datos necesarios para los tipos Range, ya que es un tipo que provee máximo, mínimo, incrementos…

Este tipo es nuevo (como los elementos que maneja) y se utiliza porque permite que unifiquemos todas las funciones de los Range_Control_Type sin tener que hacer polimorfismo ni casting por tipo, ya que todos los elementos de este grupo basan su posicionamiento en un elemento Gtk_Adjustment.

- **Canvas_Internals** ← Hereda de Window_Internals
  - *Monitor* Objeto protegido que nos proporciona las herramientas graficas para pintar en el canvas, así como quien mantiene la lista de objetos
  - *Buffer* Es el área de dibujo

- *Paintctx* Es el contexto de dibujo, osease la configuración de las herramientas con las que se dibuja (Pincel, color de relleno, color de borde, ancho de línea, color de fondo y fuente)
- *Keypress* Nos indica el comando a lanzar cuando aparece una tecla presionada sobre el canvas (o con el foco en canvas)

Cabe destacar que sobre este objeto se han realizado muchas actuaciones debido a las posibles deficiencias o problemas que ocasionaba el parpadeo de los gráficos y la carga sobre el sistema que pone cuando es mal utilizado. Sobre el y sus cambios se hablara mas adelante.

- **Common_Dialog_Internals** ← Hereda de Controlled
  - *Tipo* Tipo de elemento al que referencia
  - *Widget* Tipo básico de GtkAda, y que se convierte en puntero de su estructura real en GtkAda

Es el tipo base de los Common_Dialogs de Color, Fuente y Ficheros.

- **Colour_Dialog_Internals** ← Hereda de Common_Dialog_Internals
  - *Colour* Guarda el ultimo color escogido (aceptado) o el inicial que queramos asignarle. Con este color se abre el dialogo.

- **Font_Dialog_Internals** ← Hereda de Common_Dialog_Internals
  - *Font* Guarda la última fuente escogida (aceptada) o la inicial que queramos asignarle. Con esta fuente se abre el dialogo.

- **File_Dialog_Internals** ← Hereda de Common_Dialog_Internals
  - *Buffer* String que guarda la elección del usuario
  - *Directory* Directorio inicial de apertura
  - *Filter* Filtro sobre los ficheros (FUNCION DESHABILITADA)

- *Create* Boleano que modifica el comportamiento a la hora de elegir un fichero a través de Open o Save, y que se basa en pedir confirmaciones al usuario sobre las acciones que implica la elección, pero sin hacerlas.

- **Open_Dialog_Internals** ← Hereda de File_Dialog_Internals
- **Save_Dialog_Internals** ← Hereda de File_Dialog_Internals

Estos 2 tipos llevan consigo la información necesaria para seleccionar ficheros del sistema, independientemente de si es Linux o Windows.

Y aunque esta en JEWL.CANVAS_IMPLEMENTATION, explicare ahora este tipo por ser del mismo estilo que los anteriores :

- **Image_Internals** ← Hereda de Controlled
  - *Img* Buffer donde se guarda la imagen en formato GtkAda
  - *Width* Ancho de la imagen
  - *Height* Alto de la imagen

Este tipo es algo especial porque no esta en ninguna jerarquía, lo único que hace en el programa es que a través de un path del sistema de ficheros intenta guardar una imagen, en el original este tipo de imagen eran BMPs, pero Gtk dispone de unos loaders que mediante el mismo código permite reconocer y traspasar a formato Gtk muchos formatos de imagen, por lo que esto se gana con respecto al original de windows, y así quitamos una restricción. Por lo tanto gracias a esto, en el canvas podemos poner imágenes en muchos más formatos que antes.

En el tema de los mensajes de sistema, disponemos de 4 funciones, originalmente 3, que nos aportan la forma de dar a conocer información al usuario interrumpiendo su actividad normal con el programa, esto son las “Message boxes”, que a las ya existentes de Mensaje (Show_Message), Error (Show_Error) y Consulta (Show_Query), se les une

las de Aviso (Show_Warning). De las cuales cada una tiene su icono identificativo, y se dividen en 2 : las que solo dan información y esperan confirmación de leído (Mensaje, Error y Aviso) y las que obtienen un 'feedback' (Consulta).

Para comentar el desarrollo de todo el resto, que corresponde a los elementos gráficos que están en los paquetes JEWL.CANVAS_IMPLEMENTATION y JEWL.WINDOWS comentare parte por parte la jerarquía y sus funciones.

**1. Window_Type**

Empezamos como es normal por el tipo básico de la jerarquía : Window_Type, que aunque es "abstract" y no puede generarse un objeto de este tipo, engloba a todos los demás descendientes y todas las funciones que son comunes a todos ellos como son :

Estas acciones están pensadas para que actúen sobre el principal atributo de los datos, el "Widget", el tipo básico que soporta los tipos de Gtk

```
procedure Show      (Window  : in Window_Type;
                     Visible : in Boolean := True);
procedure Hide      (Window  : in Window_Type);
procedure Focus     (Window  : in Window_Type);
function  Visible   (Window  : Window_Type) return Boolean;
function  Get_Origin (Window  : Window_Type) return Point_Type;
function  Get_Width  (Window  : Window_Type) return Natural;
function  Get_Height (Window  : Window_Type) return Natural;
```

En el caso de las 2 siguientes funciones, actuar sobre el principal atributo no es suficiente, debido a que puede ser que trabajemos con datos relativos al padre. Estas 2 funciones por ello se complementan de otras internas, que hacen 2 cosas, primera, calculan si el objeto debe ser modificado y calculan como, y segundo lo hace recorriendo el árbol descendentemente por los hijos, para garantizar que si existen exigencias relativas al padre, estas se cumplirán.

```
procedure Set_Origin (Window  : in Window_Type;
                      Origin  : in Point_Type);

procedure Set_Size   (Window  : in Window_Type;
                      Width   : in Natural := 0;
                      Height  : in Natural := 0);
```

Estas funciones de fuente, también utilizan el mismo principio de la relatividad al padre, si esta dependencia existe, se recorre el árbol, pero en este caso, ascendentemente hasta dar con la fuente del padre o en su falta la fuente por defecto.

```
function  Get_Font   (Window  : Window_Type) return Font_Type;
procedure Set_Font   (Window  : in Window_Type;
                      Font    : in Font_Type);
```

Las nuevas funciones sobre los comandos están aquí, ya que pueden afectar a cualquier elemento de esta jerarquía, eso si, pueden llamarse sobre cualquiera, pero solo se notaran los efectos sobre aquellos que realmente lancen algún comando cuando el usuario hace alguna acción. están puestas en global porque hay elementos afectados de todos los ámbitos y porque de cara a futuras ampliaciones, esta función seria aplicable. En este caso lo que hacen estas funciones es tocar el atributo de Window_Internals que guarda la acción a devolver (Action).

```
procedure Set_Action(Window : in Window_Type;
                     Command : in   Command_Type);
procedure Disable_Action(Window : in Window_Type);
function Have_Action(Window : in Window_Type) return Boolean;
```

La siguiente funcionalidad es la obtención del ancho y alto de la pantalla principal (escritorio) con las funciones “Screen_Width” y “Screen_Height”, esta se obtiene en gtk como si fuera una ventana mas, lo que pasa que es la ventana root (Null_Window).

también en esta línea se han de cubrir la obtención del ancho y alto de un frame y del dialog. Esto como no es una constante siempre y Gtk no permite obtenerlo, se ha implementado como unas variables del paquete, las cuales se inicializan al instanciar el paquete, calculando sobre una ventana auxiliar estos parámetros:

*Frame_Width_Constant: Natural :=VAR_CALC;*

*Frame_Height_Constant: Natural :=VAR_CALC;*

*Frame_Border_Constant: Natural :=Frame_Width_Constant/2;*

*Frame_Title_Constant: Natural :=Frame_Height_Constant - (Frame_Width_Constant/2);*

La altura del menú también es modificable, pero lo que hacemos es ponerle el mismo alto que una ventana sin contenido:

*Menu_Height_Constant: Natural :=Frame_Width_Constant;*

### 1.1. Container_Type

Ahora empezamos con los container, tenemos de 2 tipos, las que permiten colocar otros elementos en su interior en una posición determinada y los menús. Los menús no dejan de ser una jerarquía de elementos de un cierto tipo, pero marcamos el orden y la posición en función del momento de la instanciación. Con respecto a los otros, ya que JEWL se basa en posiciones de pantalla y no en colocación, lo que son básicamente, es un contenedor “Fixed” de Gtk, donde con una posición y un tamaño se colocan los elementos. Pero existe la excepción, y esta es el Frame_Type, el cual además de tener esta zona fija, nos permite contener a los menús.

Siguiendo con los containers, estos no tenían ninguna función en especial sobre ellos, pero sobre la nueva versión se ha añadido unas funciones que deshacen la

limitación inicial de mantener el mismo texto durante toda la vida útil del container. Debido a que son elementos cambiantes, también lo ha de ser su titulo o descripción, por lo que aparecen asociadas a estos elementos las funciones siguientes, que acceden a los títulos de cada parte :

```
function  Get_Length (Container : Container_Type) return Natural;
function  Get_Text   (Container : Container_Type) return String;
procedure Get_Text   (Container : in  Container_Type;
                      Text      : out String;
                      Length    : out Natural);
procedure Set_Text   (Container : in  Container_Type;
                      Text      : in  String);
```

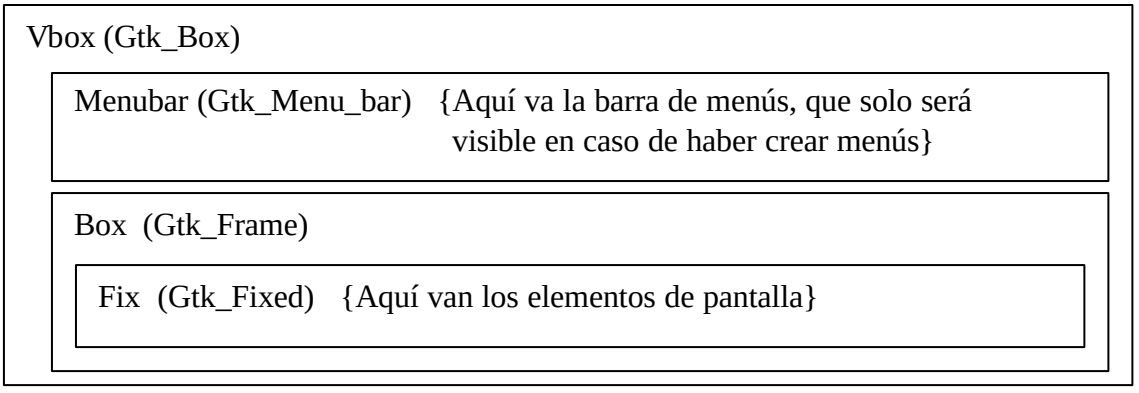
### 1.1.1. Frame_Type

El Frame es la ventana principal del programa. Tiene 2 constructores, uno con posición y otro sin ella, en este caso, la posición es que salga centrada en medio del escritorio.

Es el único contenedor que puede tener menús, además es el único redimensionable vía la interacción del usuario. Puede tener en su interior (dentro del área de cliente) cualquier otro Window_Type.

Este elemento tiene una estructura Gtk más compleja que los demás por los elementos que ha de soportar :

Widget (Gtk_Window)



Toda esta estructura se complementa con la captura de diversas señales que nos harán cumplir con las especificaciones del frame :

*'delete_event'* → Capturar este evento nos permite poder enviar el comando al usuario y a su vez conseguir que no se cierre la ventana

*'focus_in_event'* → Capturar este evento, no permite saber en cada momento que ventana es la que esta activa, de cara a que los diálogos tengan la dependencia desde la ventana que salio la llamada

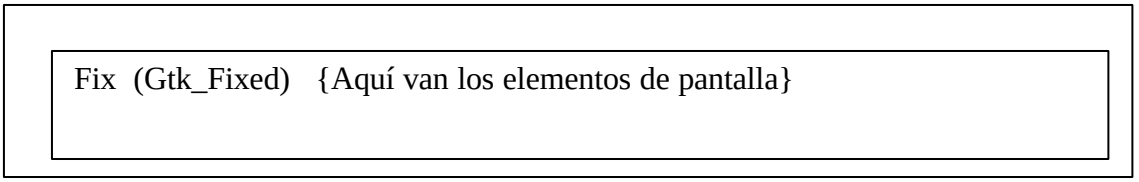
*'configure_event'* → Este evento se produce cada vez que la ventana sufre un cambio en su posición o en su tamaño, lo que nos permite tener control sobre las posiciones y tamaños relativos de los elementos

### 1.1.2. Dialog_Type

Este elemento es parecido al Frame, pero este no es redimensionable y no puede tener menús, además cada vez que aparece en pantalla bloquea a su ejecutor, hasta que haya algún evento.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Window)




Toda esta estructura se complementa con la captura de diversas señales que nos harán cumplir con las especificaciones del frame :

*'delete_event'* → Capturar este evento nos permite poder enviar el comando al usuario y a su vez conseguir que no se cierre la ventana

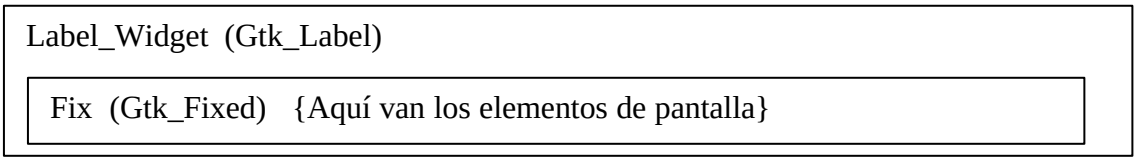
*{otros eventos}* → Cada vez que llamamos a "Execute" de un dialog, hemos de esperar a que haya algún evento para salir

### 1.1.3. Panel_Type

Este elemento es parecido al Dialog, pero este es redimensionable.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Frame)



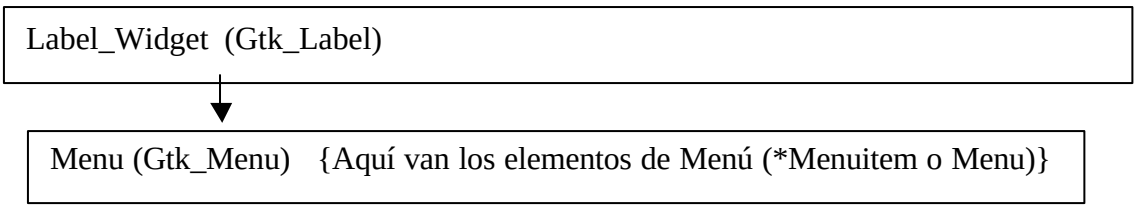

En este caso no tenemos eventos asociados.

### 1.1.4. Menu_Type

Este elemento es contenedor de menús, y puede colgar de un Frame (menú de barra de menús) o colgar de otro menú (submenú). En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&".

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Menu_item)



En este caso no tenemos eventos asociados. Y aquí se supone que la fuente no se puede cambiar, pero yo he puesto que si se pueda, el problema es para los menús que hay en la barra de menú, ya que esta depende del tamaño de las ventanas y si ponemos un tamaño muy grande esta no se vera bien.

### 1.2. Control_Type

Estos elementos son los de interacción con el usuario, ya sea desde programa o a través del teclado y ratón. Todos muestran información, pero algunos son capaces de generar eventos.

En este nivel ofrecen 3 funciones a todos los que pertenecen a esta rama:

```
procedure Enable  (Control : in Control_Type;
                   Enabled : in Boolean := True);
procedure Disable (Control : in Control_Type);
function  Enabled (Control : Control_Type) return Boolean;
```

Hasta aquí llegaban las declaraciones de la versión original, pero se decide que todos los controles puedan lucir también un “Tooltip”, por lo que es en este nivel donde se declaran las siguiente funciones de creación de “Tooltips”:

```
Procedure Set_Tooltip (Parent : Control_Type'Class; Control : in Control_Type;
                       Text : in String; PrivateText: in String:="");
Procedure Tooltip_On (Control : in Control_Type);
procedure Tooltip_Off (Control : in Control_Type);
function Get_Tooltip (Control : in Control_Type) return String;
function Get_Tooltip_Private (Control : in Control_Type) return String;
```

Estos Tooltips constan de 2 textos, uno que se vera si están activados y otro privado que nos permitirá guardar información extra consultable a través de código a la discreción del programador.

Este elemento nuevo en JEWL, podría funcionar de 2 formas :

- Tooltips individuales : cada elemento tiene un texto publico y otro privado, y se activan y desactivan individualmente
- Tooltips agrupados : los elementos tienen un texto propio, y se activan y desactivan en grupo, todas las del grupo se ven o no

Pero en el desarrollo vi que cuartar una de las 2 posibilidades era limitar, por lo tanto cree una función de asignación de tooltip que nos pide el Parent y el window, de forma que si coinciden, se mantiene en el mismo grupo o es un grupo unitario (estilo primero) y si ponemos 2 diferentes, hacemos que ambos se activen y desactiven juntos, por lo que tenemos las 2 soluciones implementadas.

### 1.2.1. Text_Control_Type

Llegamos a los controles de texto, todos ellos muestran un texto que puede ser asignado y consultado desde programa, por lo que aparecen asociadas a estos elementos las funciones siguientes, que acceden a los contenidos de cada parte :

```
function  Get_Length (Control : Text_Control_Type) return Natural;
function  Get_Text   (Control : Text_Control_Type) return String;
procedure Get_Text   (Control : in  Text_Control_Type;
                      Text    : out String;
                      Length  : out Natural);
procedure Set_Text   (Control : in  Text_Control_Type;
                      Text    : in  String);
```

#### 1.2.1.1. Button_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&".

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Button)

| Label_Widget (Gtk_Label) |
|---|

Este elemento devuelve un comando, capturando un evento :

*'clicked'* → Capturar este evento nos permite poder enviar el comando al usuario

En este objeto además se añade como ultimo parámetro y por lo tanto opcional, poner el path de una imagen, la cual aparecería a la izquierda del elemento junto al texto del botón, la imagen ha de ser del tamaño proporcional al del botón, ya que sino solo se vera una parte. En el caso de que no exista la imagen, la imagen será una X que indicara error al buscar la imagen, he optado por dejar que aparezca así somos conscientes del error.

#### 1.2.1.2. Action_Menuitem_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el

"_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Y este elemento se pone en un menú.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Menu_item)

| Label_Widget (Gtk_Label) |
|---|

Este elemento devuelve un comando, capturando un evento :

*'activate'* → Capturar este evento nos permite poder enviar el comando al usuario

#### 1.2.1.3. Label_Type

Es un elemento de muestre de información que se compone de una simple etiqueta.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Label)

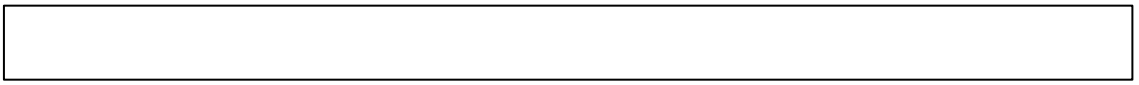
| |
|---|

#### 1.2.1.4. Editbox_Type

Es un elemento de acción y muestra información que permite editar

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Gentry)



Este elemento devuelve no devuelve ningún comando, pero para cumplir las especificaciones capturamos un evento :

*'changed'* → Capturar este evento nos permite poder informar al usuario de si ha sufrido alguna modificación desde la ultima vez que se consulto

#### 1.2.1.5. Boolean_Control_Type

Llegamos a los controles de estado, todos ellos muestran un texto que puede ser asignado y consultado desde programa, y un estado de activado o no, por lo que aparecen asociadas a estos elementos las funciones siguientes, que acceden a los contenidos de cada parte :

```
function  Get_State (Control : Boolean_Control_Type) return Boolean;
procedure Set_State (Control : in Boolean_Control_Type;
                     State   : in Boolean);
```

#### 1.2.1.5.1. Menuitem_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el

"_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Y este elemento se pone en un menú. Es un elemento muy especial con respecto a su funcionamiento y con respecto a Gtk, ya que es una mezcla de Action_Menuitem_Type y Check_Menuitem_Type,

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Check_Menu_item)

| Label_Widget (Gtk_Label) |
|---|

Este elemento devuelve un comando, pero hemos de capturar varios para que su comportamiento sea como en el original :

*'button_press_event'* → Capturar este evento nos permite poder enviar el comando al usuario sin que se marque la casilla con el check

*'activate'* → Capturar este evento nos permite poder enviar el comando al usuario cuando el usuario trabaja con las teclas

#### 1.2.1.5.2. Check_Menuitem_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Y este elemento se pone en un menú.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Check_Menu_item)

| Label_Widget (Gtk_Label) |
|---|

Este elemento devuelve un comando,:

*'activate'* → Capturar este evento nos permite poder enviar el comando al usuario cuando el usuario trabaja con las teclas

En este caso con respecto al anterior, queremos que el funcionamiento sea normal con respecto a que cada vez que varié la marca envié comando.

#### 1.2.1.5.3. Radio_Menuitem_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Y este elemento se pone en un menú.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Radio_Menu_item)

| Label_Widget (Gtk_Label) |
|---|

Este elemento devuelve un comando,:

*'activate'* → Capturar este evento nos permite poder enviar el comando al usuario cuando el usuario trabaja con las teclas

En este caso, para crear el grupo de Radio_Menuitem's hemos de generarlos seguidos, ya que de esta forma coge el grupo del anterior Radio_Menuitem debido a que se guarda el ultimo objeto generado a través de una variable del paquete "LastObject".

**Separator**

Elemento de menú que pone una línea de división, este elemento es visual, pero no esta integrado en el árbol de objetos, ya que no tiene ninguna información relevante a guardar.

#### 1.2.1.5.4. Checkbox_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Tiene el mismo funcionamiento que Check_Menuitem, pero en este caso no lanzamos ningún comando

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Check_button)

| Label_Widget (Gtk_Label) |
|---|

#### 1.2.1.5.5. Radiobutton_Type

Es un elemento de acción. En este caso los elementos tienen teclas de acceso rápido mediante "&", el problema existe ya que en Gtk, el símbolo para esa utilidad es el "_", lo que he hecho, y esto vale para los demás que tienen lo mismo, es sustituir del texto el "_" por el "-", así después puedo con toda tranquilidad poner el "_" en lugar del "&". Tiene el mismo funcionamiento que Radio_Menuitem, pero en este caso no lanza ningún comando.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Radio_button)

| Label_Widget (Gtk_Label) |
|---|

Como en el caso de los Radio_Menuitem, para crear el grupo de RadioButton's hemos de generarlos seguidos, ya que de esta forma coge el grupo del anterior radiobutton debido a que se guarda el ultimo objeto generado a través de una variable del paquete "LastObject".

### 1.2.2. Multiline_Control_Type

Estos controles contienen múltiples líneas de texto que van numeradas desde la 1 en adelante. Cada línea individual puede es accesible especificando el numero de línea, pero también puede ser seleccionable con el ratón, a través de un clic del usuario. Internamente todos estos controles tienen un índice desde 0, pero no es utilizable, ya que el 0 nos indica que queremos la línea seleccionada por el usuario. además, una excepción 'Constraint_Error' es lanzada cada vez que pedimos una línea que esta fuera del rango real de líneas disponibles.

Este conjunto de controles ofrece el siguiente juego de funciones :

```
function  Get_Count   (Control : Multiline_Type) return Natural is abstract;
function  Get_Line    (Control : Multiline_Type) return Natural is abstract;
function  Get_Length  (Control : Multiline_Type;
                       Line    : Natural := 0)   return Natural is abstract;
function  Get_Text    (Control : Multiline_Type;
                       Line    : Natural := 0)   return String  is abstract;

procedure Get_Text    (Control : in  Multiline_Type;
                       Line    : in  Natural := 0;
                       Text    : out String;
                       Length  : out Natural);
procedure Set_Text    (Control : in  Multiline_Type;
                       Text    : in  String;
                       Line    : in  Natural := 0)          is abstract;

procedure Select_Line (Control : in Multiline_Type;
                       Line    : in Natural := 0)          is abstract;
procedure Append_Line (Control : in Multiline_Type;
                       Text    : in String)                is abstract;
procedure Insert_Line (Control : in Multiline_Type;
                       Text    : in String;
                       Line    : in Natural := 0)          is abstract;
procedure Delete_Line (Control : in Multiline_Type;
                       Line    : in Natural := 0)          is abstract;
procedure Delete_All  (Control : in Multiline_Type)        is abstract;
```

Estas funciones son declaradas abstractas, porque realmente ninguna existe, pero cada control de este grupo implementa las suyas propias siguiendo el patrón marcado

por las anteriores. En este caso se opta por el polimorfismo porque el comportamiento de un control y otro es bastante diferenciado.

#### 1.2.2.1. Listbox_Type

Es un elemento de edición basado en líneas que consiste en una lista de líneas no modificables directamente por parte del usuario, que se convierte en un item de selección.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Scrolled_Window)

| Label_Widget (Gtk_List) |
|---|

En este caso hemos de complementar el elemento de lista (Gtk_List) con un elemento de scroll, ya que sino en cuanto nos pasásemos del área visible los ítems introducidos no seria visibles, de esta forma hacemos que el objeto de scroll nos garantice acceder y visualizar todas las líneas.

#### 1.2.2.2. Combobox_Type

Es un elemento de edición basado en líneas que consiste en una lista de líneas no modificables directamente por parte del usuario, que se convierte en un item de selección, pero en este caso tenemos el elemento seleccionado en un editbox, el cual si podemos cambiar, además de que el resto de líneas se esconde.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Combo)

| Label_Widget (Gtk_Gentry) |
|---|

En este caso, el objeto gtk (Gtk_Combo) cubre todas las necesidades, lo que hacemos es capturar su editable con el puntero a Widget_Label, para así facilitar el trabajo de coger el texto de la selección y a la hora de cambiar la fuente del objeto.

#### 1.2.2.3. Memo_Type

Es un elemento de edición basado en líneas que consiste en una lista de líneas modificables directamente por parte del usuario.

Este elemento tiene una estructura Gtk tal que:

Widget (Gtk_Frame)

| Gtk_Scrolled_Window |
|---|
| Label_Widget (Gtk_Text_View) |

En este caso hemos de complementar el elemento de edición (Gtk_Text_View) con un elemento de scroll, ya que sino en cuanto nos pasásemos del área visible los ítems introducidos no seria visibles, de esta forma hacemos que el objeto de scroll nos garantice acceder y visualizar todas las líneas. El caso de introducir todo esto en un Gtk_Frame tiene como respuesta, que el aspecto sea parecido al del JEWL original.

En este caso también, comento que para cumplir con el requerimiento del "Undo" era necesario conocer cuando se hacían cambios. Este objeto, tiene un funcionamiento por transacciones, las cuales empiezan por un "Begin_User_Action" y acaban con un "End_User_Action", esto nos permitía poder obtener un punto de salvado del contenido sobre el atributo de Window_Internals, Undo_Text. De esta forma cada vez que se

produce un cambio este se guarda, pudiendo hacer el undo. Pero esto solo se aplica a acciones de usuario, por lo que cuando hacemos acciones via programa, esto no funcionaba, por lo que en cada procedimiento teníamos que hace que se llamaran a las funciones de "Begin_User_Action" y "End_User_Action", de esta forma los cambios de usuario y los cambios por programa se pueden deshacer.

### 1.2.3. Canvas_Type

Llegamos al canvas, el área de pintura. Para empezar comentar los problemas que se apreciaban sobre el objeto original. Para comenzar el parpadeo de los gráficos y los modos de trabajo del mismo.

Pero empecemos explicando la estructura Gtk:

Widget (Gtk_Drawing_Area)

Esta drawing_area, es solo la parte visible, ya que si simplemente usáramos esto, los problemas seguirían siendo los mismos, por lo que esta estructura se apoya en un buffer grafico en 'background' (Gtk_Pixmap) que es donde realmente pintamos y es el cual nos ayuda a implementar una especie de doble buffer de canvas. Con ello se evita parte del parpadeo, ya que no pintamos directamente sobre el área visible, pero no se resuelve del todo. Otra peculiaridad de este doble buffer, es que siempre se guarda el área mas grande necesitada por el canvas, de esta forma no hemos de estar generando cada vez que hay un repintado un segundo buffer, mas grande o pequeño, además esta peculiaridad nos servirá de cara a otras modalidades del canvas. Este doble buffer se encuentra en un paquete separado (GtkJEWL.Double_Buffer)

Una vez explicado el doble buffer del canvas, hemos dicho que esto no soluciona el problema, debido a que cada grafico se pinta a través de una lista de gráficos, la cual, cada vez que añadimos un elemento se repinta, cosa que sigue produciendo algo de parpadeo, debido a que repintamos sobre el buffer principal prácticamente, porque el cambio de un buffer al otro es instantáneo entre pintar 2 pequeñas cosas. Una solución que propongo es el retardo del repintado hasta haber añadido a la lista todos los objetos que queremos que aparezcan en ese momento y al final del proceso, hacer el cambio de buffer, eso lo que evita es repintado inútiles y por supuesto el parpadeo que ello provoca, además la CPU también lo nota, cargándola menos. Todo esto se consigue con las mismas funciones graficas, a las cuales se les ha añadido un parámetro al final de las mismas llamado 'Paint' que nos indica si hemos de hacer el repintado, el traslado de la lista al área de dibujo. Este parámetro si no es utilizado esta por defecto a 'True', esto es debido a que hemos de conseguir la compatibilidad hacia atrás, lo que conlleva que ha de ser repintado cada vez si no le decimos nada, y con ello, según lo que hagamos habrá parpadeo, como en el original.

El funcionamiento de este modo se basara en que en todas las llamadas graficas deben llevar a al final el "Paint=>False);", necesitando al final del proceso forzar un repintado, lo cual se puede hacer de 2 formas, dejando en el ultimo elemento grafico el 'Paint' a Trae o con una nueva función que fuerza esta situación sobre el canvas :

Force_Draw ( Canvas : in Canvas_Type)

De esta forma podemos seguir trabajando como en el original, pero cambiando pequeñas cosas que aportan una visualización mejor, pero aun seguimos teniendo el problema del posible crecimiento desmesurado de la lista, lo cual podemos solucionar de 2 formas :

- Poniendo limite : No es una buena solución, porque nunca sabemos encontrar un máximo que en todos los sistemas y maquinas ofrezca el mejor resultado, además todo lo que son restricciones, no son la solución.
- La otra posibilidad es "eliminar" la lista.

Lo de eliminar la lista no es literal, lo único que hacemos es ofrecer una nueva modalidad de canvas, el cual se basa en que la lista una vez se pinta sobre el canvas, esta se borra, podríamos decir que aplicamos el algoritmo del pintor, las capas se van superponiendo, para borrar algo no hacemos un 'restore', sino que pintamos encima. La ventaja de este método es que nos permite también utilizar las ventajas de parámetro 'Paint' ya que podemos acumular unos cuantos elementos en la lista antes de forzar el repintado. Todo este lo podemos obtener al instanciar el canvas, mediante un parámetro al final de la llamada llamado 'No_List' que por defecto esta a false, por compatibilidad hacia atrás.

En este punto retomare la funcionalidad del doble buffer, que como he dicho se mantenía el buffer de mayor tamaño instanciado aunque no fuera visible, esto porque los dibujos en este ultimo método no se redibujan por lo tanto si lo hemos pintado querremos conservarlo aunque no este visible, y la única forma de ello es conservar el buffer donde lo pintamos, por ello si hemos tenido un buffer grande, lo conservamos. En los otros modos esto no es importante, ya que al repintar, se repintaría lo que esta fuera del área visible también.

En este último modo, solo cabe un problema o limitación, perdemos las funciones de 'Save', 'Clear', 'Restore' y 'Set_Colour', por lo que el color de fondo no es cambiable.

también cabe destacar que para reducir los cálculos gráficos todos los puntos de From y To de los gráficos son ordenados de tal forma que el From tiene la X e Y mínimas y el To las X e Y máximas.

Una vez explicados los modos de salida del canvas enunciaremos los eventos que son necesarios para mantener este montaje:

*'expose_event'* → Repinta las zonas que por un momento han quedado tapadas y vuelven a ser visibles, sin tener que repintar todo

'*configure_event*' → Para este evento, tenemos 2 niveles : nivel de double_buffer y nivel del canvas. En el primer nivel, garantiza que el buffer de background tenga el tamaño máximo de todas las llamadas y que el contenido se traspase. En el nivel de canvas hace que repintemos la pantalla, ya que esta puede haber crecido y habrá objetos que antes no se veían y ahora si, para garantizar que todo es correcto.

'*button_press_event*' → Captura cuando un botón del ratón es presionado sobre el área del canvas, lo que pasa es que solo tratamos los del botón_1, y solo los tratamos cuando la acción del ratón esta activada.

'*button_release_event*' → Captura cuando un botón del ratón es soltado sobre el área del canvas.

'*button_motion_event*' → Captura cuando el ratón se mueve con el botón presionado sobre el área del canvas, lo que pasa es que solo tratamos los del botón_1, y solo los tratamos cuando la acción del ratón esta activada.

'*key_press_event*' → Captura cuando una tecla imprimible es presionada sobre el área del canvas solo los tratamos cuando la acción del teclado esta activada.

### 1.2.4. Range_Control_Type

Este grupo de elementos es nuevo en GtkJewl. Nos provee de objetos que marcan un valor sobre un rango. Y tienen unas funciones comunes que nos permiten manejar su funcionamiento :

*procedure Set_Value ( R : Range_Control_Type; Val : in Float);*
*procedure Set_Min ( R : Range_Control_Type; Min : in Float);*
*procedure Set_Max ( R : Range_Control_Type; Max : in Float);*

*function Get_Value ( R : Range_Control_Type) return Float;*
*function Get_Min ( R : Range_Control_Type) return Float;*
*function Get_Max ( R : Range_Control_Type) return Float;*

#### 1.2.4.1. Progressbar_Type

Elemento que nos permite marcar un progreso sobre un rango de valores entre un mínimo y un máximo, pero gráficamente expresado en un porcentaje de relleno. Este elemento, tiene también la posibilidad de que decidamos no solo la dirección del objeto sino el sentido de crecimiento en 4 posibilidades (de derecha a izquierda, de izquierda a derecha, de arriba hacia abajo, de abajo hacia arriba ) mediante el enumerado Range_Orientation.

#### 1.2.4.2. Scrollbar_Type

Elemento que nos permite marcar un valor sobre un rango de valores entre un mínimo y un máximo, pero gráficamente expresado en la posición del marcador. Nos permite indicarle el salto por 'click' para cuando este marcador no es arrastrado. también nos permite elegir entre vertical u horizontal, ya que aquí el sentido no es importante, se indica con el mismo enumerado 'Range_Orientation'.

#### 1.2.4.3. Scale_Type

Elemento que nos permite marcar un valor sobre un rango de valores entre un mínimo y un máximo, pero gráficamente expresado en la posición del marcador y en el valor en texto puesto al lado. Nos permite indicarle el salto por 'click' para cuando este

marcador no es arrastrado y el número de dígitos de precisión del valor mostrado. también nos permite elegir entre vertical u horizontal, ya que aquí el sentido no es importante, se indica con el mismo enumerado 'Range_Orientation'.

#### 1.2.4.4. Spinbutton_Type

Elemento que nos permite marcar un valor sobre un rango de valores entre un mínimo y un máximo, pero gráficamente expresado en el valor en texto puesto al lado. Nos permite indicarle el salto por 'click' y el número de dígitos de precisión del valor mostrado.

### 2. Common_Dialog_Type

Proveen al usuario de ventanas de ayuda para la obtención de parámetros de sistema como son, fuentes, colores y acceso al sistema de ficheros.

Para lanzarlos hemos de ejecutar a través de esta función que nos devuelve si ha confirmado la elección el usuario o no:

*function Execute (Dialog : Common_Dialog_Type) return Boolean;*

### 2.1. Colour_Dialog_Type

Provee al usuario una interfaz para elegir un color, en este caso con GtkAda no podemos coger el selector nativo para asegurar la independencia, cogemos el dialogo que tiene Gtk, de esta forma se vera igual en Windows y Linux, sin perder funcionalidad. Para apoyar este tipo tenemos las funciones de acceso:

```
procedure Set_Colour    (Dialog : in Colour_Dialog_Type;
                         Colour : in Colour_Type);
function  Get_Colour    (Dialog : in Colour_Dialog_Type) return Colour_Type;
```

### 2.2. Font_Dialog_Type

Provee al usuario una interfaz para elegir una fuente, en este caso con GtkAda no podemos coger el selector nativo para asegurar la independencia, cogemos el dialogo que tiene Gtk, de esta forma se vera igual en Windows y Linux, sin perder funcionalidad. Para apoyar este tipo tenemos las funciones de acceso:

```
procedure Set_Font    (Dialog : in Font_Dialog_Type;
                       Font   : in Font_Type);
function  Get_Font    (Dialog : in Font_Dialog_Type) return Font_Type;
```

### 2.3. File_Dialog_Type

Provee al usuario una interfaz para elegir archivos del sistema de ficheros, en este caso con GtkAda no podemos coger el selector nativo para asegurar la independencia, cogemos el dialogo que tiene Gtk, de esta forma se vera igual en Windows y Linux, pero para este caso hemos de perder funcionalidad. Los filtros no están implementados, pero por compatibilidad se mantienen las llamadas relacionadas. Para apoyar este tipo tenemos las funciones de acceso:

```
procedure Set_Name      (Dialog : in File_Dialog_Type;
                         Name   : in String);
function  Get_Name      (Dialog : in File_Dialog_Type) return String;
procedure Add_Filter    (Dialog : in File_Dialog_Type;
                         Text   : in String;
                         Filter : in String);
procedure Set_Directory (Dialog : in File_Dialog_Type;
                         Name   : in String);
```

### 2.3.1. Open_Dialog_Type

En este caso, el archivo que escojamos del sistema de ficheros es para lectura.

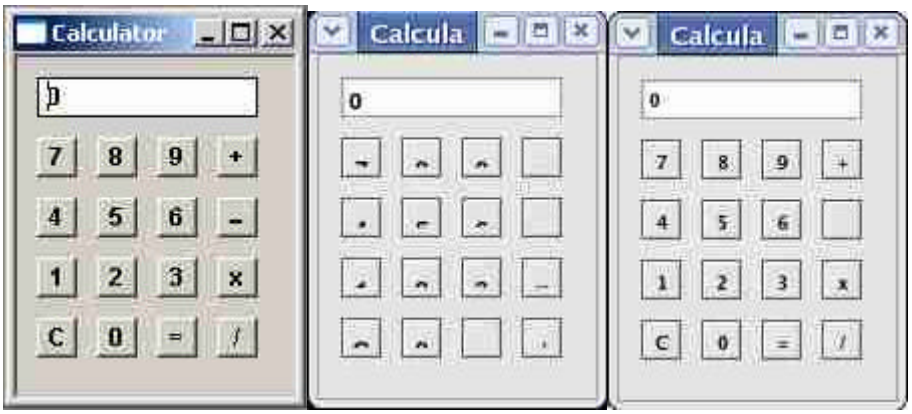
### 2.3.2Open_Dialog_Type

En este caso, el archivo que escojamos del sistema de ficheros es para escritura.

# 6.- EVALUACIÓN

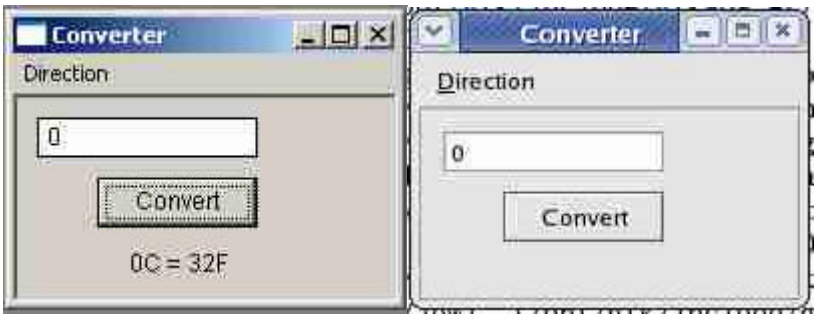
Para la evaluación del correcto funcionamiento se evalúan los resultados con los programas de ejemplos que vienen con las librerías JEWL originales, que por especificación han de obtener un aspecto más o menos acorde y una funcionalidad totalmente igual. Evaluamos por lo tanto estos casos.

**Calculador**



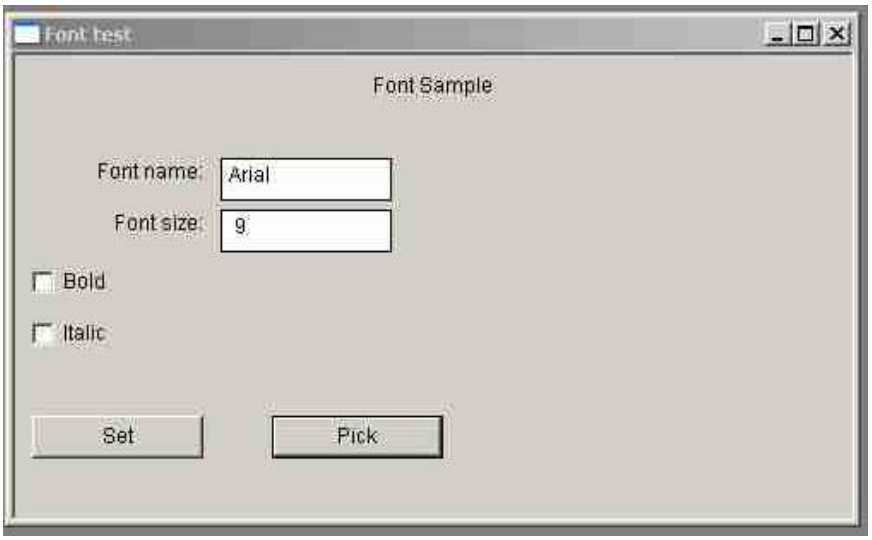

En este ejemplo podemos observar en primer lugar el original, el segundo es bajo linux con las GtkJewl, existe el problema de que las fuentes en Windows y Linux no son las mismas a priori, me explico, el original esta en Arial 9, pero linux no tiene Arial, por lo que coge Sans, y el aspecto a priori no es el mismo, hasta que tocamos el código y vemos que funciona correctamente.

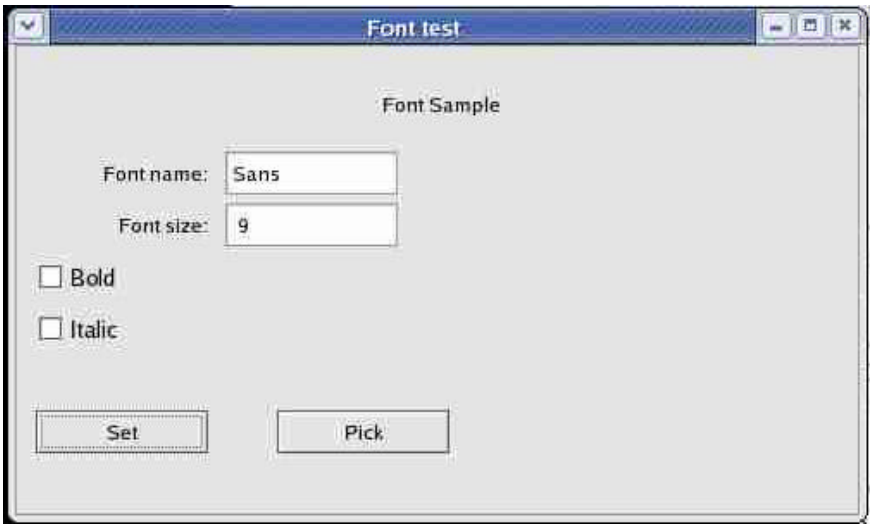
**Converter**



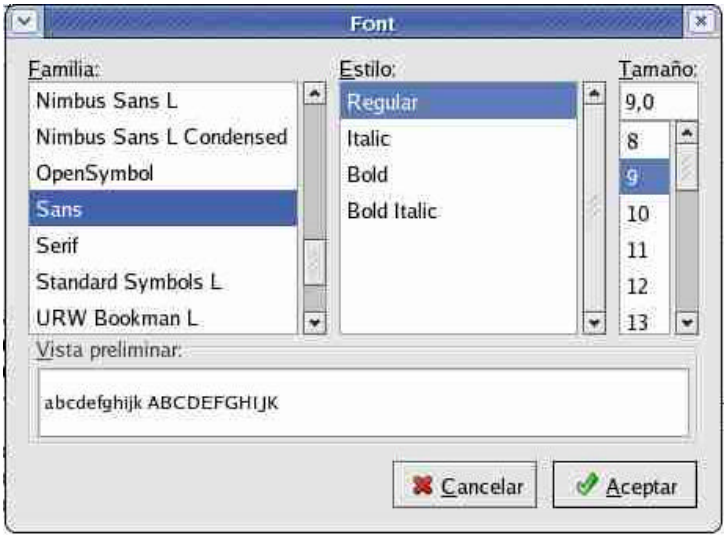

En este caso no existe ningún problema, la funcionalidad y el aspecto son los mismos. menús, editboxes, botones y una etiqueta que recibe un Set_Text, todo correcto

**Font_test**



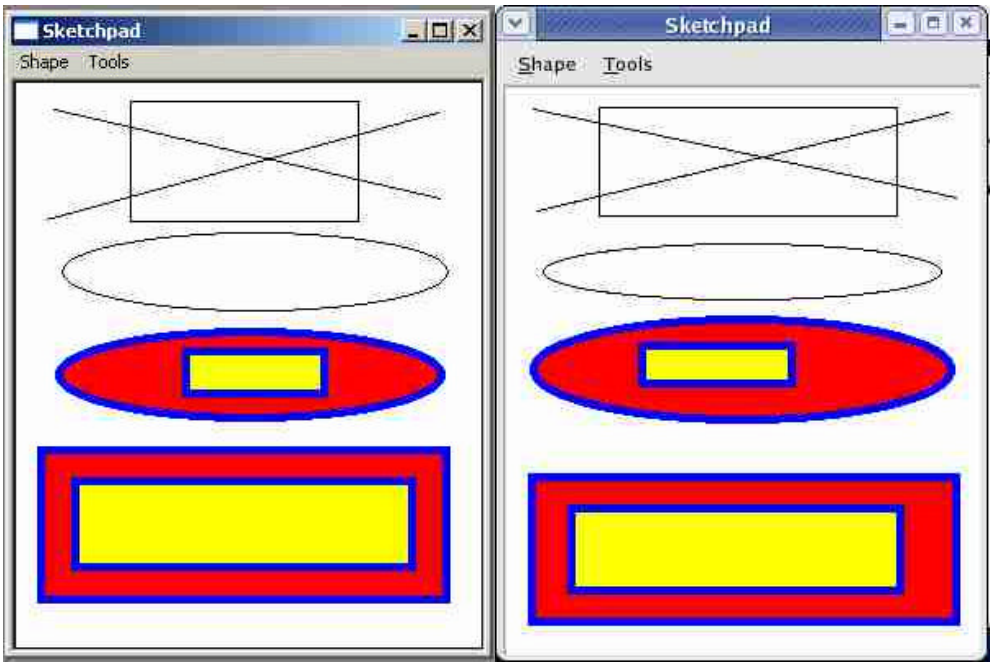




En este ejemplo, también podemos ver que el funcionamiento y la funcionalidad son las mismas, aquí tenemos nuevos elementos como son los checkbox y modificamos la fuente de una etiqueta. La gran diferencia es la que tienen los selectotes de fuente, diferentes en aspecto, pero idénticos en funcionalidad. también vemos el nuevo selector



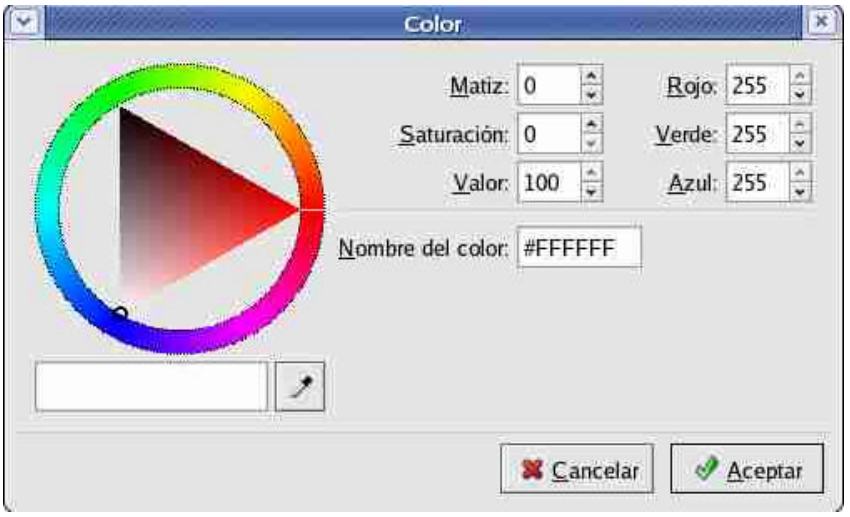
de fuentes de la librería, igual funcionalidad, diferente aspecto.

**Skecth**

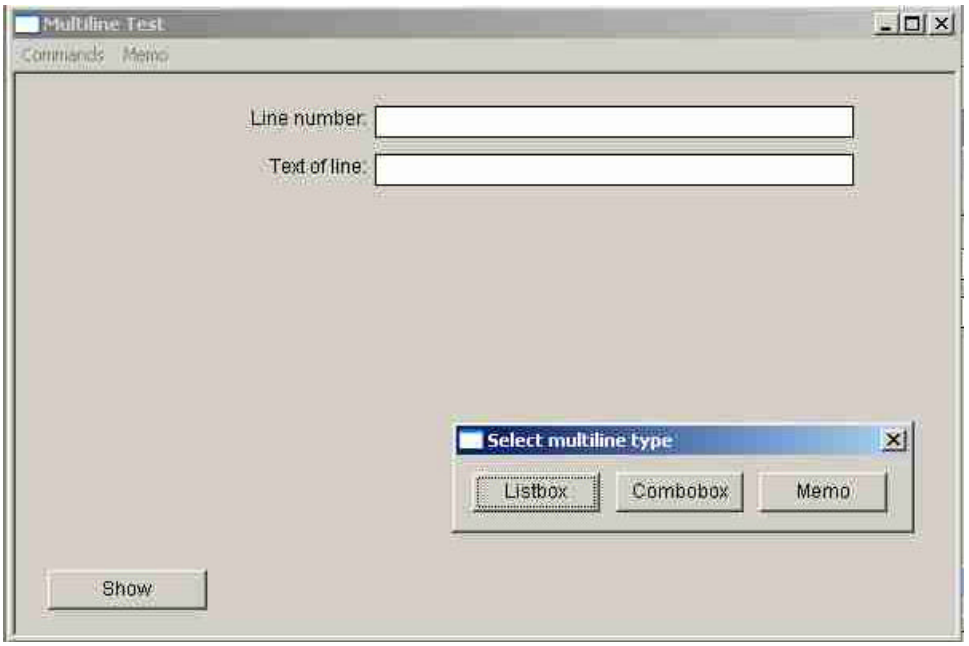


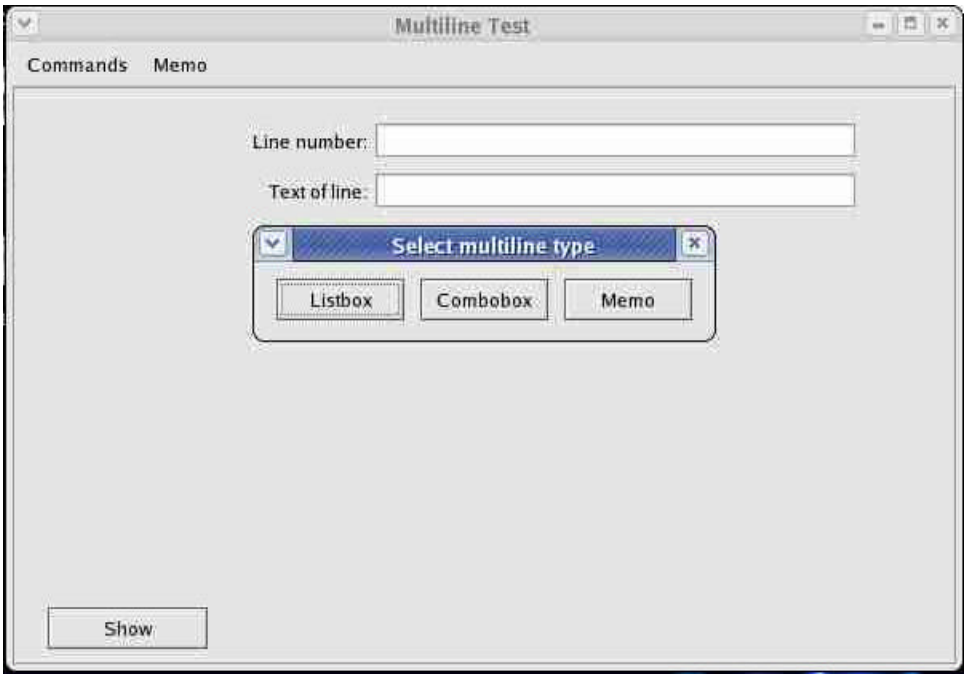

En este caso probamos el canvas, en ambos casos el dibujo parpadea, en el original todo cada vez que añadimos algo, en el segundo solo el nuevo elemento parpadea, porque el redibujar anterior solo se hace una vez y con doble buffer, el gran cambio se nota cuando añadimos el parámetro 'Paint=>False' a todos los Draw y Restore , poniendo al final un Force_Draw, en esta situación el parpadeo desaparece.

también en este ejemplo probamos la funcionalidad del nuevo selector de color, que aunque diferente, nos permite los mismos colores (a excepción de los predefinidos o personalizados)

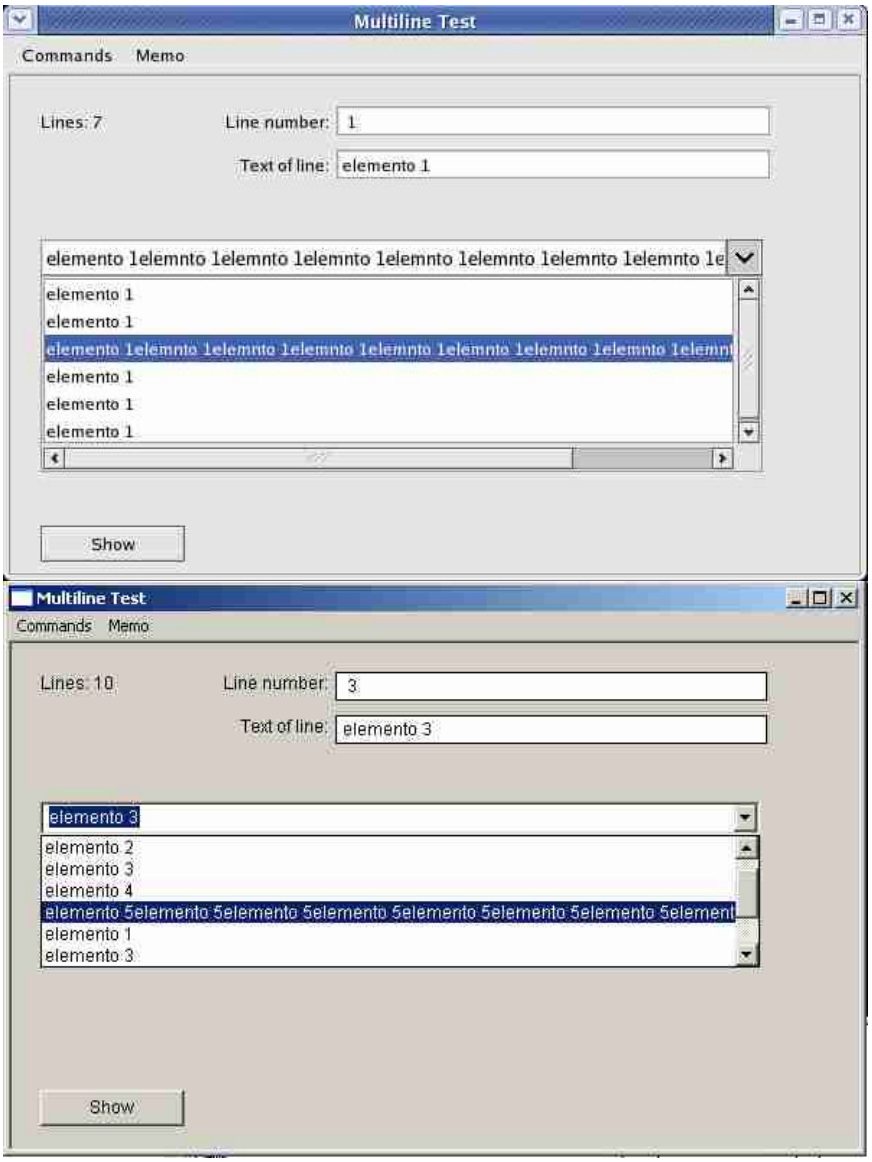


**Multicine_test**




En este ejemplo, podemos observas las diferentes posiciones originales de los diálogos. Probamos que la ventana padre no tenga el focus, en el original no puede no tener el foco, en cambio en GtkJewl el Frame no puede actuar nada excepto que puede obtener el foco (eso si, siempre permaneciendo detrás) y puede ser maximizado y minimizado, pero eso son limitaciones de GtkAda. En este ejemplo podemos probar el funcionamiento de todos los "Multiline", viendo que funcione añadir, insertar, borrar, seleccionar… Todo ello comprobando también que las medidas de scroll automático funcionen para así dejar que el usuario pueda ver toda la información de cada línea.




En este caso de la "Combobox" probamos el buen funcionamiento de las principales funciones que son las de seleccionar, insertar, añadir , borrar y borrar todo. El aspecto es parecido, la única diferencia que tenemos es que en las nuevas librerías Jewl tenemos scroll en ambas direcciones, eso no es un problema, al contrario diría yo, ya que permite navegar por la línea que se sale del grafico.

*NOTA :* Como ejemplo de prueba de correcto funcionamiento se que no son suficientes estas pruebas, pero una validación completa de todas las funcionalidades y funciones de las librerías se han ido probando a medida que se generaban, así creo que la validación se ha realizado (y no hace falta hacer un informe mas grande). El problema es que no esta documentada, pero espero que si algún día tienen utilidad estas librerías se abra un periodo de 'Test', para corregir los pormenores que no se ven.

# 7.- CONCLUSIONES

Para comenzar con las conclusiones cabe destacar que los objetivos inicialmente previstos se han cumplido, no se si con creces o no, porque en informática todo siempre es mejorable, pero aun así podemos decir que esta versión de las librerías ofrece mas que la anterior y es capaz de crecer aun mas.

Por esto empezaré a decir que esta mayor funcionalidad aportada a las librerías permitirá ampliar el abanico de posibilidades de exigencias en los enunciados así como las posibilidades de los mismos, esto a su vez pone menos limites para continuar con la enseñanza en Ada que parecía que perdía fuerza debido a que tenia la gran deficiencia del aspecto grafico, y aunque existen herramientas en Ada para este propósito, esta pretende ser una mas cercana a los alumnos dada la simplicidad de uso y aprendizaje.

Por otro lado, elimina límites a la hora de elegir el sistema de desarrollo que en estos momentos será claramente a favor de Windows, y eso hoy día que se pretende promover el software libre, es un obstáculo. Por lo tanto estas librerías pretenden fomentar también el uso de Linux, que para un informático creo que es interesante saber manejar.

Sobre el rendimiento de estas librerías, es bueno, mejoran muchos aspectos importantes de las JEWL originales, pero de cara al rendimiento de comunicación con el sistema operativo, nunca se puede competir con librerías y funcionamientos que han sido pensados para un Sistema en concreto.

Para finalizar las conclusiones, decir que habiendo aprendido GtkAda, no es la mejor forma de utilizarlo, ya que no aprovechamos todo su potencial, solo una pequeña parte, esto da 2 ventajas a las GtkJewl :

- Posibilidad de crecimiento alta
- Para niveles de exigencia bajos, no compensa aprender todo sobre GtkAda

# 8.- RECURSOS UTILIZADOS

La información sobre GtkAda aun de momento es pobre, tanto en la red como fuera de ella, lo que si que existe abundante es para Gtk, lo cual no es del todo una ayuda ya que GtkAda va muy por detrás en las implementaciones y si se pretende utilizar la ultima versión de Gtk, esto es complicado ya que GtkAda esta limitado hasta donde se haya llegado. GtkAda actualmente esta por la versión 2.4, la cual es compatible totalmente con Gtk 2.0, mientras que Gtk trabaja en la 2.8.

A continuación detallo mis fuentes de información para la realización de este proyecto :

- *Web oficial de John English (JEWL) (http://www.it.bton.ac.uk/staff/je/jewl/)*

  Extraemos distribución y manual disponible, para comenzar a saber que es lo que tenemos que conseguir.

- *Web oficial de GtkAda en Europa (*http://libre.act-europe.fr/GtkAda/*)*

  Esta Web nos aporta acceso a las últimas distribuciones de las fuentes de GtkAda, en su versión Windows y versión Linux.

- *Documentación de GtkAda*

  Me refiero a la que viene dentro de los ficheros de instalación de GtkAda, esta en formato HTML, el problema es que no tiene información de todos los paquetes, aunque estos ya estén disponibles

- *Web de Gtk+ (http://www.gtk.org)*

  De esta Web sacamos los runtimes de Gtk, así como podemos consultar funcionalidades de paquetes de Gtk, aunque se ha de ser consciente que no todo esta implementado en GtkAda

- *GTK+/GNOME Programming (Meter Wright) {Disponible en la Biblioteca de la ETSE}*

Libro que hace una introducción a la programación GTK, no es una ayuda total en Ada, pero sirvió para que supiera que era Gtk y de que disponía, así como su filosofía de programación.

- *Buscador Google (http://www.google.es)*

útil casi para cualquier tema, pero sobre todo en este caso para buscar foros en los cuales alguien tuviera problemas con GtkAda y ver sus contenidos. Ya lo digo de antemano, tema GtkAda no muy divulgado, mas dudas que soluciones.

# 9.- MANUAL

## REQUISITOS

Los requisitos iniciales y mínimos que necesitamos para hacer la instalación de GtkJewl son :

- Tener instalador el compilador de ADA (pruebas realizadas con GNAT-3.15p)
- Tener instaladas las librerías de Gtk (pruebas realizadas con gtk+ 2.6.8)
- Tener instaladas las librerías de GtkAda (pruebas realizadas con GtkAda 2.4)
- Sistemas Windows o Linux (pruebas con Win2000, WinXP y Linux Fedora )

## INSTALACIÓN EN WINDOWS

- Partimos de que se cumple la exigencia de tener el compilador de GNAT instalado y configurado en el sistema (yo acostumbro a instalar "gnat-3.15p-nt.exe", "gnatwin-3.15p.exe" y "adaguide")

- Continuamos con la instalación de Gtk ( en las pruebas "gtk-runtime-2.6.8-rev-a.exe")
  - Instalamos sobre <gtkpath>
  - Hemos de poner el en PATH de sistema el <gtkpath>\bin

- Continuamos con la instalación de GtkAda ("GtkAda-2.4.exe" en las pruebas)
  - Ejecutamos el fichero de instalación sobre <gtkada_path>
  - Garantizamos que en <gtkada_path>\include\gtkada\gtk.ads exista la línea de código "pragma Linker_Options ("-luser32");"
  - Abrimos ventana de comando y vamos a "cd <gtkada_path>"
  - Ejecutamos "gnatmake –Pinclude/gtkada/gtkada"
  - Añadimos a las variables de entorno ADA_INCLUDE_PATH y ADA_OBJECTS_PATH el siguiente camino "<gtkada_path>\include\gtkada"

- Realizamos la instalación de GtkJewl
  - Copiamos la carpeta GtkJewl (que contiene doc, source y examples) en el lugar de destino (instalación). Hasta GtkJewl lo llamaremos <gtkjewl_path>
  - Abrimos ventana de comando y vamos a "cd <gtkjewl_path>"
  - Ejecutamos "gnatmake GtkJewl-io.adb"
  - Añadimos a las variables de entorno ADA_INCLUDE_PATH y ADA_OBJECTS_PATH el siguiente camino "<gtkJewl_path>\source"

## INSTALACIÓN EN LINUX

- En este caso partimos de que se cumple la exigencia de tener el compilador de GNAT instalado y configurado en el sistema, así como las librerías de Gtk, normalmente en el CD de instalaciones (mínimo gtk+2.2) o en el apartado de programación.

- Continuamos con la instalación de GtkAda ("GtkAda-2.4.0.tgz" en las pruebas)
  - Copiar Gtk-2.4.0.tgz en /tmp
  - Acceder a tmp con "cd /tmp" y descomprimir con "tar xfz GtkAda-2.4.0.tgz"
  - Accedemos a la carpeta descomprimida con "cd GtkAda-2.4.0"
  - Ejecutamos el fichero de instalación sobre <gtkada_path> esto se hace mediante el comando "./configure –prefix=<gtkada_path>
  - Si todo esta OK, podemos instalar y ejecutamos "make install", una vez acabado esto tenemos GtkAda en el sistema en <gtkada_path>
  - Modificamos las variables de entorno PATH y LD_LIBRARY_PATH, esto se puede hacer de 2 formas :
    - Cada vez que accedemos al terminal lo hemos de hacer para que sean visibles:

      PATH=<gtkada_path>/bin:$PATH
      LD_LIBRARY_PATH=<gtkada_path>/lib:$LD_LIBRARY_PATH
      LANG=es_ES
      export PATH LD_LIBRARY_PATH LANG
    - Configuración de nuestro usuario

      Añadimos las mismas líneas, pero en el fichero de configuración '.bash_profile'

- Realizamos la instalación de GtkJewl
  - Copiamos la carpeta GtkJewl (que contiene doc, source y examples) en el lugar de destino (instalación). Hasta GtkJewl lo llamaremos <gtkjewl_path>
  - "cd <gtkjewl_path>"
  - Ejecutamos "gnatmake GtkJewl-io.adb"

## COMPILACION EN WINDOWS

Compilación de 2 formas:

1.- A través de Adaguide, apretando al botón de "Build"
2.- En línea de comando utilizar: "gnatmake <file>.adb"

## COMPILACION EN LINUX

Compilaremos con la siguiente línea de comando:

→gnatmake –I<gtkJewl_path> <file>.adb `gtkada-config` ←

## MANUAL DE USO

En carpeta /doc de la carpeta de las librerías GtkJewl