Programación de Aplicaciones Usando las Bibliotecas GNOME

George Lebl
El Proyecto GNOME

jirka@5z.com

Programación de Aplicaciones Usando las Bibliotecas GNOME
por George Lebl

En este tutorial, usted recibirá una primera impresión de las bibliotecas de GNOME. Aprenderá como desarrollar rápidamente aplicaciones usando dichas bibliotecas y cómo conseguir interfaces de usuario consistentes usando los widgets propios de GNOME. Otro punto en el que se centrará este documento es en el desarrollo de aplicaciones en lenguaje C, usando el conjunto de widgets GTK+.

Tabla de contenidos

Créditos, Copyrights y Otras Aclaraciones Adicionales

Todo el código contenido en este tutorial se acoge a la Licencia Pública General GNU (GPL) o a la Licencia Pública General para Bibliotecas GNU (LGPL). El código fue escrito por los integrantes del equipo de desarrollo de GNOME y por el autor mismo.

El autor expresa sus disculpas acerca de errores gramaticales o de ortografía que pudieran aparecer en el texto. El inglés no es su lengua nativa y espera que su código sea lo suficientemente explícito como para suplir sus carencias con el idioma.

Sobre la Traducción

La presente traducción es fruto del trabajo de un grupo de personas pertenecientes a LILO [Asociación de Informática de la Universidad de Alcalá de Henares]. Este trabajo ha sido realizado con la esperanza de acercar el software libre y más concretamente la biblioteca de desarrollo de aplicaciones gráficas GNOME, a la comunidad hispanohablante.

Durante el proceso de elaboración de esta traducción ha colaborado un grupo heterogéneo de personas, por lo tanto, y aunque no hemos escatimado esfuerzos para presentar la traducción más coherente posible, es probable que el lector se encuentre con algunos fallos. Agradeceríamos en gran medida que si detecta alguna errata o tiene alguna idea de cómo mejorar a la traducción nos la haga saber para incluir sus aportaciones en próximas revisiones.

Quisiéramos también expresar nuestro agradecimiento a todos los miembros de la lista de correo de LILO, a Gnome-es, a la Escuela Politécnica de Alcalá de Henares y a todas las personas que directa o indirectamente han colaborado con LILO, haciendo posible la presente traducción.

Los autores de esta traducción son, por riguroso orden alfabético, los siguientes:

- David Fernández Barrero <`dfb60332@alu.uah.es`>
- Javier Ballesteros Plaza <`heraclit0@wanadoo.es`>
- Jorge García González <`gar_gon@teleline.es>`>
- Jorge Rodríguez <`jorginius@interlap.com.ar`>
- Pablo Marín Tomás <`pablo.tmarin@uah.es`>
- Raúl Ocaña Rodríguez <`ror@retemail.es`>

La revisión del texto ha sido realizada por Sergio de la Hoz Pérez <`purificalo@hotmail.es`>.

Por último, si desea contactar con el grupo LILO, puede hacerlo a través de http://lilo.sourceforge.net

Capítulo 1. Vistazo a las Bibliotecas GNOME

Dónde Encajan las Bibliotecas GNOME

Antes de entrar en describir las especificaciones de las bibliotecas GNOME, es importante ver dónde encajan en el conjunto de todas las diferentes bibliotecas que son usadas en una aplicación GNOME. Las bibliotecas GNOME están en el nivel más alto. GTK+ con sus dos partes, GTK y GDK forman el siguiente nivel. GTK proporciona un modelo de objetos para C y un conjunto de herramientas de interfaz de usuario con los widgets básicos que dan la base genérica para una GUI. GTK depende de GDK, el cual es una envoltura alrededor de Xlib, la biblioteca que habla directamente con el servidor X. Todo (excepto Xlib) depende de GLib, que es una biblioteca de C muy práctica con muchas utilidades y portabilidad así como un surtido de contenedores para C fáciles de usar.

Figura 1-1. Jerarquía de Bibliotecas Enlazadas con una Aplicación GNOME

Estructura de las Bibliotecas GNOME

Ahora miramos a la estructura de las bibliotecas GNOME para ver qué nos pueden ofrecer. Aquí hay un listado de las diferentes bibliotecas que están presentes en el paquete gnome-libs:

libgnome
: Biblioteca de utilidades independiente del conjunto de herramientas

libgnomeui
: Biblioteca dependiente del conjunto de herramientas

libgnorba
: Biblioteca para usar la implementación de CORBA ORBit con GNOME

gtk-xmhtml
: widget xmhtml portado de GTK, usado para el visualizador de ayuda

zvt
: widget de emulación de terminal

libvfs

Una biblioteca de un sistema de archivos virtual usado en Midnight Commander

libart_lgpl

Una biblioteca usada para mostrar bonitos gráficos con suavizado (N.T: anti-aliased)

No cubriremos gtk-xmhtml, zvt, libvfs, libart_lgpl ni libgnorba porque son mayoritariamente bibliotecas especializadas y algunas, como libvfs y gtk-xmhtml probablemente serán remplazadas en un futuro cercano.

Podemos ver una clara división entre *libgnome* y *libgnomeui*. La primera es usada de forma independiente al conjunto de herramientas e incluso puede ser utilizada en programa de línea de órdenes que nunca usan X. La última es la biblioteca que proporciona los widget estándar y un armazón para aplicaciones escritas usando GTK+. Puede concebirse escribir programas con otros conjuntos de herramientas, pero nadie ha escrito todavía una *libgnomeui* con un conjunto de herramientas diferente, y dudo que esto pase alguna vez porque GTK+ es realmente un gran conjunto de herramientas.

Capítulo 2. Programando con GTK+

Introducción

GTK+ es una biblioteca escrita en C, que aglutina a un conjunto de widgets orientados hacia la programación de aplicaciones gráficas en X Window. Está altamente orientada a objetos y se acopla con los lenguajes más populares, como C++, Objective C, Perl, TOM, Guile, Python, etc... GTK+ además usa GLib, que es una biblioteca en C muy útil, incluye ayudas para portar los programas a diferentes arquitecturas y contenedores como listas enlazadas o tablas de claves (N.T: hash tables). Si ya está familiarizado con GTK+, entonces se aburrirá leyendo esta sección.

GLib

Acuerdos en la Nomenclatura

GLib es una biblioteca muy usada en GTK+ y en casi todo GNOME. Las funciones de GLib empiezan con *g_* (como *g_strdup*), la definición de tipos de GLib para la mayoría de ellos tienen simplemente como prefijo una *g* (como *gint32*), y las estructuras de GLib están en mayúsculas y empiezan con una *G* (como *GHashTable*).

Definición de Tipos

GLib proporciona algunos tipos predefinidos para facilitar la portabilidad, simplificar y clarificar el código y para mantener cierta consistencia. La siguiente tabla muestra estos tipos. Si el campo *Equivalente* está en blanco, significa que no hay representación equivalente en el C estandar.

Tabla 2-1. Tipos en GLib

Nombre	Equivalente	Descripción
gint8		Entero con signo de 8 bits
guint8		Entero sin signo de 8 bits
gint16		Entero con signo de 16 bits
guint16		Entero sin signo de 16 bits
gint32		Entero con signo de 32 bits
guint32		Entero sin signo de 32 bits
gint64		Entero con signo de 64 bits (vea nota al margen)
guint64		Entero sin signo de 64 bits (vea nota al margen)
gchar	char	Carácter
guchar	unsigned char	Carácter sin signo
gshort	short	Entero corto
gushort	unsigned short	Entero corto sin signo
glong	long	Entero largo
gulong	unsigned long	Entero largo sin signo

Nombre	Equivalente	Descripción
gint	int	Entero
guint	unsigned int	Entero sin signo
gfloat	float	Número real
gdouble	double	Número real de doble precisión
gboolean	int	Tipo para almacenar valores VERDADERO/FALSO
gpointer	void *	Tipo para almacenar punteros a distintos objetos
gconstpointer	const void *	Tipo para almacenar punteros a distintos objetos inmutables

Se debe tener en cuenta que *gint64* y *guint64* pueden no estar disponibles en todas las plataformas. Puede comprobar esto en su código viendo si la macro *G_HAVE_GINT64* está definida

Como puede ver, algunos de los tipos como *gint* parecen no tener otro sentido en la vida que tener el prefijo 'g' (Son idénticos a los ofrecidos por C estandar). La razón de su existencia es la de hacer el código mas consistente y limpio. Aunque no sea un crimen no usar estos tipos, debería emplearlos para facilitar la portabilidad de su código. Algunos de los tipos como *gboolean* están sólo para mejorar la claridad del código y podría usar también *int* para hacer exactamente lo mismo, pero el anterior método indica claramente que se esta hablando de un valor que solo puede tomar los valores TRUE/FALSE (VERDADERO/FALSO).

Portabilidad y Funciones de Utilidad

Hay funciones que o bien no se comportan exactamente igual en todos los sistemas, o bien son inseguras o bien no existen en alguna plataforma (o en ninguna), así que GLib proporciona sus propias implementaciones o recubrimientos que tienen un comportamiento constante y que normalmente comprueban los argumentos.

Aquí tiene algunas de las funciones más útiles que encontrará en esta categoría. Tenga en cuenta que el encabezado "Prototipo" es a título informativo, ya que algunas de estas funciones pueden ser realmente macros.

Tabla 2-2. Algunas Funciones Portables de GLib

Prototipo	Descripción
gchar * g_strdup (const gchar *)	Devuelve la localización de una nueva cadena que es una copia del argumento, si el argumento es NULL, se devuelve NULL
gpointer g_malloc (int tam)	Devuelve una nueva región de memoria de 'tam' bytes, si no se puede reservar la memoria solicitada, el programa abortará invocando *g_error*
void g_free (gpointer p)	Libera la memoria apuntada por 'p', si 'p' es NULL no realiza ninguna operación

Prototipo	Descripción
gint g_snprintf (gchar *cadena, gulong n, gchar const *formato, ...)	Funciona simplemente como sprintf escribiendo los argumentos de acuerdo con ’formato’ en ’cadena’, sin embargo solo usará ’n’ bytes de la ’cadena’, así que truncará el resultado si éste necesitase más. Devuelve el numero de bytes que son en realidad escritos en ’cadena’.
void g_usleep (gulong cont)	Suspende la ejecución durante al menos ’cont’ microsegundos
gint g_strcasecmp (const gchar *c1, const gchar *c2)	Compara la cadena ’c1’ y ’c2’ sin tener en cuenta las mayúsculas
gint g_strncasecmp (const gchar *c1, const gchar *c2, guint n)	Compara los ’n’ primeros caracteres de la cadena ’c1’ y ’c2’ sin tener en cuenta las mayúsculas

También hay algunas funciones de utilidad y macros que realmente no se encuentran en las bibliotecas normales de C. Aquí hay una pequeña lista de algunas de las más útiles e importantes.

Tabla 2-3. Algunas Funciones Útiles de GLib

Prototipo	Descripción
g_new (tipo, cont)	Una macro que asignará memoria nueva para ’cont’ elementos del tipo ’tipo’ y devuelve el resultado como ’tipo’. Es equivalente a ’(tipo) g_malloc(cont * sizeof(tipo))’
g_new0 (tipo,cont)	La misma semántica que g_new, excepto que la memoria devuelta estará a cero. Tenga en cuenta que no se debería asumir que poner la memoria a cero pondrá a cero los números reales (Del tipo ’float’)
gchar * g_strconcat (const gchar *cad, ...)	Cuando se le pasa cualquier numero de argumentos del tipo (const char *) y un NULL despues del último argumento, devolverá una nueva cadena que proviene de la concatenación de todos los argumentos.
gchar * g_strdup_printf (const gchar *formato, ...)	Una función como printf que devolverá una nueva cadena con el resultado de la operación printf
gchar * g_strstrip (gchar *cadena)	Eliminará los espacios en blanco del principio y final de una cadena. No reserva memoria nueva, pero modifica la cadena original y devuelve un puntero a ella. Si desea reservar memoria nueva use una construcción como esta: ’cadena2 = g_strstrip(g_strdup(cadena1));’

Hay otros utilísimos métodos en GLib, y le insto a estudiar la documentación de GLib y el archivo de cabecera de GLib (*glib.h*): si lo hace ahorrará gran cantidad de tiempo no reimplementando funcionalidades básicas.

Contenedores

Probablemente la mejor parte de GLib sean sus contenedores. Aquí tenemos una lista de los contenedores de GLib.

Tabla 2-4. Contenedores de GLib Comunes

Nombre	Descripción
GList	Lista doblemente enlazada
GSList	Lista enlazada simple
GHashTable	Tabla de claves (N.T: hash)
GCache	Cache
GTree	Árbol binario balanceado
GNode	Árbol n-simo
GString	Cadena de tamaño dinámico
GArray	Vector (N.T: Array) de tamaño dinámico
GPtrArray	Vector de punteros de tamaño dinámico
GByteArray	Vector de bytes (guint8) de tamaño dinámico

GList, Lista Doblemente Enlazada

La forma más fácil de usar listas o pilas en su código es con *GList*. La estructura básica de *GList* representa por un lado una lista doblemente enlazada y por otro un único nodo de la lista. Podrá añadir sus datos dentro del puntero de datos de *GList* (El miembro 'data' perteneciente a la estructura). Aquí tenemos una enumeración de las funciones que actúan sobre GList. Las funciones normalmente recogen un puntero y devuelven la lista modificada (Un puntero a ella). Tenga en cuenta que el primer nodo puede ser ahora diferente.

Tabla 2-5. Funciones Más Importantes de GList

Prototipo	Descripción
GList* g_list_append (GList *lista, gpointer datos)	Añade al final de la lista un nuevo nodo que contiene el puntero 'datos'. Si 'lista' es NULL creará una lista nueva.
GList* g_list_prepend (GList *lista, gpointer datos)	Añade en el inicio de la lista un nodo que contiene 'datos', si 'lista' es NULL creará una lista nueva.
GList* g_list_remove (GList *lista, gpointer datos)	Elimina el nodo que contiene el parametro 'datos' de la lista. Tenga en cuenta que sólo se comparan punteros (Y no contenidos).

Prototipo	Descripción
GList* g_list_find (GList *lista, gpointer datos)	Encuentra el nodo de GList que contiene el parametro 'datos'. De nuevo, téngase en cuenta que se comparan únicamente los punteros.
GList* g_list_next (GList *lista)	Una macro que devuelve un puntero al nodo posterior.
GList* g_list_previous (GList *list)	Una macro que devuelve un puntero al nodo anterior.
void g_list_free(GList *list)	Libera la lista entera. *NO libera la memoria asignada a sus datos, esto debe hacerlo usted mismo antes antes de invocar a esta función.*

Para acceder a los datos de un nodo particular de *GList* lea el miembro *data* en la estructura *GList*. Así, el código que crearía una lista doblemente enlazada de dos elementos con dos cadenas duplicadas, y después liberara esa lista y las cadenas, quedaría así:

```
GList *list = NULL; /*el actual puntero a la lista*/
GList *li; /*simplemente un puntero temporal a un nodo us-
ado para interaccionar
             con la lista*/
...
/*aquí añadimos dos cadenas a la lista*/
list = g_list_append(list,g_strdup("Cadena 1"));
list = g_list_append(list,g_strdup("Cadena 2"));
...
/*aquí nos movemos por la lista, liberando todas las cade-
nas y al final
/* liberamos la lista en sí*/
for(li = list; li!= NULL; li = g_list_next(li)) {
        char *string = li->data;
        g_free(string);
}
g_list_free(list);
```

GString, Cadenas de Tamaño Dinámico

Otro contenedor simple de usar y muy útil es el contenedor *GString*. Es un contenedor para cadenas de tamaño dinámico, ideales en las ocasiones en las que no sabe que longitud tendrá la cadena que va a necesitar. Aquí tenemos una lista de las funciones más importantes.

Tabla 2-6. Funciones Más Importantes de GString

Prototipo	Descripción
GString* g_string_new (const gchar *cad)	Crea una GString nueva con el valor 'cad'
void g_string_free (GString *cadena, int borra_contenido)	Libera la estructura GString y opcionalmente también el segmento de datos de la cadena

Prototipo	Descripción
GString* g_string_append(GString *cadena, const gchar *val)	Añade al final 'val' a 'cadena'
GString* g_string_prepend (GString *cadena, const gchar *val)	Añade al principio 'val' a 'cadena'
void g_string_sprintf (GString *cadena, const gchar *formato, ...)	Una función como sprintf para GString
void g_string_sprintfa (GString *cadena, const gchar *formato, ...)	Una función como sprintf para GString, pero añade la cadena en lugar de sobreescribirla

Para acceder a los datos de la cadena para usarlos como un *char **, simplemente acceda al elemento *str* de la estructura *GString*. En realidad es posible liberar la estructura *GString* sin liberar el segmento de datos. Esto es útil si quiere crear una cadena en C normal. El siguiente ejemplo es una función que coge un vector de enteros y los escribe dentro de una cadena y devuelve un *char **.

```
char *
create_number_list(int array[], int array_len)
{
        int i;           /* el contador del vector */
        GString *string; /* la variable GString */
        char *ret;       /* al valor de retorno */

        /* crear una nueva GString vacía */
        string = g_string_new("");

        /* recorrer el vector de enteros */
        for(i=0; i<array_len; i++) {
                /* añadir el número a la cadena en paréntesis */
                g_string_sprintfa(string, "(%d)", array[i]);
        }

        /* poner el valor de retorno */
        ret = string->str;

        /* liberar la estructura GString, pero no los datos */
        g_string_free(string,FALSE);

        /* devolver la cadena */
        return ret;
}
```

GHashTable

Aunque usado con menor frecuencia que GList y GString, el contenedor para tablas de claves es muy útil también. Una tabla de claves se podría entender como un conjunto de objetos (en GLib el término objeto se refiere a un *gpointer*) y sendas llaves, estás últimas nos permitirán acceder a cada objeto posteriormente. GLib lleva este hecho más allá, haciendo que la llave sea un *gpointer* también, y dandonos la posibilidad de que le proporcionemos una función de comparación y de acceso (N.T: hashing). Aunque esto hace a *GHashTable* mucho más flexible, puede llevarnos a alguna confusión con respecto a la asignación de la memoria para las llaves.

Vamos a enunciar algunas funciones importantes y nos ocuparemos de los detalles posteriormente:

Tabla 2-7. Funciones Más Importantes de GHashTable

Prototipo	Descripción
GHashTable* g_hash_table_new (GHashFunc func_hash, GCompareFunc func_clave_cmp)	Crea una tabla de claves nueva usando la función de generación de claves y la de comparación especificadas
void g_hash_table_destroy (GHashTable *tabla_hash)	Destruye la tabla de claves y libera la memoria. Esto no implica que libere ningún dato o las claves, esto lo tendrá que hacer usted mismo
void g_hash_table_insert (GHashTable *tabla_hash, gpointer clave, gpointer valor)	Introduce un nuevo 'valor' con una clave 'clave'
void g_hash_table_remove (GHashTable *tabla_hash, gconstpointer clave)	Elimina el valor con la clave 'clave' de la tabla. No libera ni el valor ni la clave.
gpointer g_hash_table_lookup (GHashTable *tabla_hash, gconstpointer clave)	Busca el valor del puntero con clave 'clave'. Devuelve NULL si no lo encuentra
gboolean g_hash_table_lookup_extended (GHashTable *tabla_hash, gconstpointer lookup_key, gpointer *orig_key, gpointer *value)	Consulta si existe el dato con clave 'clave_valor', almacena la clave del puntero original en 'clave_orig' y el valor en 'valor'. Devuelve TRUE si la consulta tuvo éxito, FALSE en caso contrario. Debe usar esta función al eliminar un elemento de la memoria para deshacerse de la clave original.
void g_hash_table_foreach (GHashTable *tabla_hash, GHFunc func, gpointer datos)	Ejecuta 'func' sobre cada dato almacenado en la tabla de claves. El parámetro 'datos' se le pasará a la función como último argumento. El prototipo de GHFunc es el siguiente.
void (*GHFunc) (gpointer clave, gpointer valor, gpointer datos)	Esta es la función prototipo que usará con la función que se pasa a g_hash_table_foreach. Coge la 'clave', el 'valor' y los 'datos', especificado este último campo en la llamada a g_hash_table_foreach.
guint g_str_hash (gconstpointer v)	Una función para generar claves a partir de cadenas
gint g_str_equal (gconstpointer v, gconstpointer v2)	Una función de comparación de cadenas para tablas de claves
guint g_int_hash (gconstpointer v)	Una función para generar claves a partir de enteros
gint g_int_equal (gconstpointer v, gconstpointer v2)	Una función de comparación de enteros para tablas de claves

Para crear una tabla de claves, pase las funciones de acceso y de comparación de claves a *g_hash_table_new*. Hay funciones genéricas definidas para cadenas (*g_str_hash* y *g_str_equal*) y otras. Sin embargo si le pasa NULL como funciones de acceso y de comparación, obtendrá un puntero directo: los punteros serán en realidad utilizados como claves.

El problema de la reserva de memoria se vuelve aparente cuando empezamos a usar cadenas como claves.*GHashTable* no almacena la cadena, todo lo que almacena es un puntero. Por tanto, cuando insertamos un valor dentro de la tabla, tenemos que crear una copia nueva de la clave para ese valor (un otro caso, habrá elementos que compartan la clave). Esto es importante recordarlo pues sino las cosas no funcionarán como esperamos. El otro problema es como liberar entonces la clave. Si usa *g_hash_table_remove*, dará como clave una cadena con el mismo contenido que la clave original pero no en la misma zona de memoria (un puntero distinto). Entonces, el puntero a la clave original se habrá perdido y a menos que haya almacenado la dirección en algún sitio se habrá creado una laguna en la memoria. Lo que tiene que hacer en su lugar es usar *g_hash_table_lookup_extended* para obtener primero los punteros a la clave y al valor y luego hacer uso de *g_hash_table_remove*.

El siguiente ejemplo creará una nueva tabla de claves, insertará una pareja de cadenas, las recuperaremos, y entonces destruiremos la tabla y los valores almacenados en ella:

```
/* función que usaremos para lib-
erar las claves y los datos en la tabla antes
de que la destruyamos */
static void
free_key_value(gpointer key, gpointer value, gpointer user_data)
{
        g_free(key);
        g_free(value);
}

...

/* en algún lugar del código   */

GHashTable *ht;

/* crea una nueva tabla de claves cadenas como claves*/
ht = g_hash_table_new(g_str_hash, g_str_equal);

/* introduce una pareja de cadenas (es decir, colores marca-
dos por la forma) */
g_hash_table_insert(ht, g_strdup("triangulo"), g_strdup("verde"));
g_hash_table_insert(ht, g_strdup("cuadrado"), g_strdup("rojo"));
g_hash_table_insert(ht, g_strdup("círculo"), g_strdup("azul"));

/* de nuevo, en algún lugar del código */
...
/* ahora aquí deseamos imprimir el color de un cuadrado */
char *color;

/* coger el color del cuadrado*/
color = g_hash_table_lookup(ht, "cuadrado");

printf("El color del cuadrado es: %s\n",color);

/* de nuevo en algún lugar del código */
...
/* Ahora simplemente queremos destruir la tabla de claves y lib-
erar toda la
 * memoria asociada a ella. Usaremos la fun-
ción free_key_value y la usaremos
```

```
 * sobre todos los valores de la tabla. */
g_hash_foreach(ht, free_key_value, NULL);

/* ahora podemos destruir la tabla de claves actual */
g_hash_table_destroy(ht);
```

Más Información Sobre GLib

Para mas información mirad el archivo de cabecera *glib.h* y la documentación el sitio web www.gtk.org[1].

GTK+

GUIs Básicos

Escribir un GUI (Graphic User Interface: Interfáz gráfico de Usuario) basado en GTK+ es en esencia extremadamente simple, y nosotros usaremos el ejemplo de Hola Mundo del excelente tutorial de GTK+ de Ian Main, el cual es una utilísima guía para escribir aplicaciones para GNOME. Pero primero hablaremos sobre la filosofía oculta de GTK+.

GTK+ es una herramienta basada en contenedores, queriendo esto decir que no especifica dónde está el widget, sino en que contenedor está. Algunos widget, como una ventana, un marco o un botón, son contenedores que sólo pueden almacenar otro único widget. Por ejemplo un botón con una etiqueta es realmente un botón dentro del cual hemos añadido un widget etiqueta. Si necesita poner más widgets dentro de ese contenedor, necesitará añadir otro contenedor dentro de él.

De hecho la mayoría de capas de una ventana normalmente están formadas con contenedores como cajas horizontales, cajas verticales y tablas, estos son los más importantes de aprender. Una caja horizontal es un widget al que puede añadir varios widgets dentro y estos serán añadidos en una fila horizontal. La altura de la caja horizontal es la altura del widget añadido más alto y su longitud es la suma de la longitud de todos los widgets. Las cajas verticales se comportan exactamente igual, excepto que es vertical en lugar de horizontal. Una tabla puede incluir widget en diferentes filas y columnas.

Figura 2-1. Ejemplo de la Jerarquía de la Ventana

Modelo de Objetos en GTK+

El modelo de objetos de Gtk es un marco de trabajo orientado a objetos para C. Incluye una singular herencia de objetos, metodos virtuales, señales, modificación de objetos en tiempo de ejecución, comprobación de tipos en tiempo de ejecución y otras bondades. Aunque escribir un objeto GTK+ es más enrevesado que escribirlo en algo como Java, tiene muchas ventajas. GTK+ es un modelo de objetos que no requiere herencia para la mayoría de las cosas que se hacen con objetos. Por ejemplo, como los métodos son simplemente funciones que cogen un puntero al objeto como primer argumento, es muy fácil escribir más metodos en su propio código, que no existían en el objeto original.

Los objetos en GTK+ son simplemente estructuras en C y la herencia se logra simplemente incluyendo la estructura padre como primer elemento en la estructura hija. Esto significa que podemos hacer conversiones explícitas entre tipos sin problemas. Además de la estructura del objeto que es una instancia, hay también una estructura de clase para cada clase que contenga cosas como punteros a funciones virtuales y manejadores de señales.

Tipos de Metodos en GTK+

El modelo de objetos en GTK+ usa 3 tipos de métodos, un método puede ser una simple función C que espera un puntero a un objeto como primer argumento. Este tipo de método es el más rápido en ejecutarse, pero el programador no dispone de ningún mecanismo para modificar su funcionamiento o para reemplazarlo por su propio método.

El segundo tipo es el método virtual. Usted puede reemplazar estos métodos por los suyos propios en clases derivadas (N.T: lo que comúnmente se llama "polimorfismo" en los términos de la programación orientada a objetos), pero hasta aquí llegan sus ventajas. Un método virtual se implementa como un puntero a una función en la estructura de la clase, normalmente con una función de recubrimiento que se encarga de llamar al método virtual.

El tercer tipo de método y el más flexible (y también el de llamada más lenta) es el método de señal. El tipo señal es parecido al virtual, el usuario puede modificar el funcionamiento del método pero esto lo consigue conectando manejadores que se pueden ejecutar antes o despues del método por defecto. Algunas veces los objetos tienen una señal con el único propósito de tener manejadores del programador conectados a ella. Estas señales tienen su cuerpo de método vacío (No aportan método por defecto alguno).

Datos en Objetos

Hay una forma de almacenar datos arbitrarios en los objetos para extender el objeto. Esto se hace con el método, *gtk_object_set_data* (o *gtk_object_set_user_data* para un puntero simple sin nombre). Para escoger el dato, usamos *gtk_object_get_data*. Ejemplo:

```
GtkObject *obj;
void *some_pointer;
...
/* aquí ponemos el dato "some_data" en obj para que apunte a
   some_pointer */
gtk_object_set_data(obj,"some_data",some_pointer);
...
/* recuperamos el puntero de some_data desde obj y lo guardamos en
```

```
  some_pointer */
some_pointer = gtk_object_get_data(obj,"some_data");
```

El puntero puede ser de un tipo desconocido (Un (void *)) puesto que es manipulado como tal.

Convención de Nombres en GTK+/GNOME

GTK+ y GNOME usan la misma norma para los nombres de objetos y funciones. GTK+ usa el prefijo *gtk_* para las funciones, y *Gtk* para los objetos, y GNOME usa *gnome_* y *Gnome*. Cuando una función es un método de un objeto, el nombre (en minúsculas) se añade al prefijo. Por ejemplo, el objeto botón se llama *GtkButton* (que es el nombre de la estructura en C que sustenta los datos del objeto), y llama al método "*new*" (El constructor) para *GtkButton* como *gtk_button_new*. Las macros asociadas con objetos usan la misma norma que las funciones, pero todas están en mayúsculas. Por ejemplo una macro que convierte explícitamente un objeto a *GtkButton* se llama *GTK_BUTTON*. Hay excepciones importantes, la macro que comprueba el tipo para *GtkButton*, se llama *GTK_IS_BUTTON*.

Usando Métodos en GTK+

Puesto que GTK+ es orientado a objetos, usa herencia para los widgets. Por ejemplo *GtkHBox* y *GtkVBox* derivan de *GtkBox*. Y así se puede usar cualquier método *GtkBox* sobre *GtkVBox* o *GtkHBox*. Sin embargo necesitamos convertir explícitamente el objeto *GtkVBox* a *GtkBox* antes de que poder llamar a la función. Esto se podría hacer en C estándar de esta forma:

```
GtkVBox *vbox;
...
gtk_box_pack_start((GtkBox *)vbox, ...);
...
```

Aunque esto funcionaría, es inseguro (No verifica en ningún momento que vbox sea un objeto perteneciente a una clase derivada de GtkBox). GTK+ proporciona un mecanismo de comprobación de tipos, de forma que pueda prevenirle si esta haciendo un conversión a un objeto que no deriva del objeto al que estas convirtiendo, o si intenta hacer una conversión a NULL. La macro es el nombre del widget en mayúsculas. Por ejemplo el código anterior debería escribirse de la siguiente manera:

```
GtkVBox *vbox;
...
gtk_box_pack_start(GTK_BOX(vbox), ...);
...
```

GNOME usa exactamente la misma forma así que cualquier concimiento que haya adquirido sobre GTK+ puede ser usado para los widget de GNOME, simplemente cambie el prefijo GTK por GNOME.

Programa de Ejemplo Hola Mundo

Aquí esta el código de ejemplo prometido para el programa ’Hola mundo’. No usa ningún contenedor avanzado, solamente una ventana y un botón dentro del cual se le ha añadido una etiqueta. Ilustra el funcionamiento básico de un GUI escrito en GTK+. No se asuste de su tamaño, casi todo son comentarios.

```
/* ejemplo holamundo holamundo.c */

#include <gtk/gtk.h>

/*  esto es una función de retrollamada. los argumentos son ignora-
dos en este
 *  ejemplo... Mas retrollamadas más adelante. */
void
hello (GtkWidget *widget, gpointer data)
{
        g_print ("Hola Mundo\n");
}

gint
delete_event(GtkWidget *widget, GdkEvent *event, gpointer data)
{
        g_print ("Evento delete\n");
        /* si devuelve FALSE en el manejador de la
         * señal "delete_event", GTK emitirá la señal "destroy".
         * Devolviendo TRUE da a entender que no quiere
         * que la ventana sea destruida. Esto es útil para
         * que aparezcan los diálogos típicos de '¿está seguro
         * de que desea salir?'. */

        /* Cambiando TRUE a FALSE la ventana principal
         * sera destruida con un "delete_event". */

        return (TRUE);
}

/* otra retrollamada */
void
destroy (GtkWidget *widget, gpointer data)
{
        gtk_main_quit ();
}

int
main (int argc, char *argv[])
{
        /* GtkWidget es el tipo para el almacenamiento de los wid-
gets */
        GtkWidget *window;
        GtkWidget *button;

        /* Esta función se debe llamar en todas la aplicaciones GTK
         * antes de ninguna otra los argumentos son pasados desde la
         * línea de órdenes y devueltos a la aplicación. */
        gtk_init (&argc, &argv);

        /* crear una nueva ventana */
        window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

        /* cuando a la ventana se le pasa la señal "delete_event"
         * (la envía el usuario a través del gestor de ventanas), le
         * pedimos que llame a la función delete_event () tal y como
         * está definida arriba. El dato pasado a la función de
         * retrollamada es NULL y es ignorado en la función de
         * retrollamada. */
```

```
        gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                            GTK_SIGNAL_FUNC (delete_event), NULL);

        /* aquí conectamos el evento "destroy" al manejador
         * de la señal. Este evento sucede cuando llamamos a
         * gtk_widget_destroy() en la ventana, o si devolvemos
         * 'FALSE' en la retrollamada "delete_event". */
        gtk_signal_connect (GTK_OBJECT (window), "destroy",
                            GTK_SIGNAL_FUNC (destroy), NULL);

        /* configura el ancho del borde de la ventana. */
        gtk_container_border_width (GTK_CONTAINER (window), 10);

        /* crea un nuevo botón con la etiqueta "Hola Mundo". */
        button = gtk_button_new_with_label ("Hola Mundo");

        /* Cuando el botón recibe la señal "clicked", llamará
         * a la función hello() pasándole NULL como argumento.
         * La función hello() esta definida arriba. */
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
                            GTK_SIGNAL_FUNC (hello), NULL);

        /* Esto causará que la ventana se destruya por la llamada
         * a gtk_widget_destroy(window) cuando se emita "clicked".
         * De nuevo, la señal "destroy" podría venir desde aquí o
         * desde el gestor de ventanas. */
        gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                                   GTK_SIGNAL_FUNC (gtk_widget_destroy),
                                   GTK_OBJECT (window));

        /* esto pone el botón dentro de una ventana
         * (un contenedor gtk). */
        gtk_container_add (GTK_CONTAINER (window), button);

        /* el paso final es ver este recién creado widget... */
        gtk_widget_show (button);

        /* y la ventana */
        gtk_widget_show (window);

        /* Todas las aplicaciones GTK deben tener un gtk_main().
         * Aquí termina el control y se es-
pera hasta que suceda un evento
         * (como una pul-
sación del teclado o un evento del ratón). */
        gtk_main ();

        return 0;
}
/* Fin del ejemplo */
```

Para más información mire el archivo de cabecera en <prefijo>/include/gtk/ y <prefijo>/include/gdk/ ('prefijo' será el directorio donde haya instalado los archivos de cabecera de GTK+) y la documentación en el sitio web www.gtk.org[2].

Notas

1. http://www.gtk.org/
2. http://www.gtk.org/

Capítulo 3. Programación en GNOME

Introducción

Qué es un programa GNOME

Un programa GNOME es una aplicación GUI (interfaz gráfico de usuario) para GTK+ que hace uso de las bibliotecas GNOME. Las bibliotecas GNOME hacen posible tener un aspecto similar entre aplicaciones y hacen las cosas más sencillas y fáciles de programar. Además las bibliotecas GNOME añaden una completa gama de widgets que extiende el conjunto de widgets que ofrece GTK+.

Un Programa GNOME sencillo

El siguiente programa crea una ventana GNOME básica y añade una caja horizontal dentro de la cual se incluyen dos botones, los cuales (cuando son presionados) imprimen una cadena hacia la salida estándar del terminal desde el que comenzó la aplicación. La semántica y la estructura de un programa GNOME es muy similar a la programación pura en GTK+.

Necesitará compilar este código, así que debería mirar el capítulo Construyendo Aplicaciones GNOME para obtener mas información sobre como realizar este paso. Aquí tiene un sencillo Makefile que compilará buttons.c para usted. Para compilar otros ejemplos, sólo cambie 'buttons' por el nombre apropiado.

```
CFLAGS=-g -Wall `gnome-config --cflags gnome gnomeui`
LDFLAGS=`gnome-config --libs gnome gnomeui`
all: buttons
```

Y aquí esta buttons.c:

```
/*
 * Sencillo programa GNOME, no integrado en el ár-
bol GNOME, sin usar i18n
 * buttons.c
 */
/* La cabecera GNOME más sencilla */
#include <gnome.h>

/* una llamada para los botones */
static void
button_clicked(GtkWidget *button, gpointer data)
{
        /* la cadena que se imprim-
irá es pasada por el campo 'data'  */

        char *string = data;
        /* imprime una cadena en la salida estándar*/
        g_print(string);
}

/* llamada cuando el usuario cierra la ventana*/
static gint
delete_event(GtkWidget *widget, GdkEvent *event, gpointer data)
{
```

```
        /* señal al bucle principal para salir*/
        gtk_main_quit();
        /* devuelve FALSE para continuar cerrando la ventana */
        return FALSE;
}

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

        /* Inicializa GNOME, esto es muy parecido a gtk_init */
        gnome_init ("buttons-basic-example", "0.1", argc, argv);

        /* Crea un widget GNOME, el cual establece una ven-
tana básica
        para su aplicación*/


        app = gnome_app_new ("buttons-basic-example",
                             "Aplicación GNOME Básica");

        /* conecta "delete_event", el cual es el evento que se obtiene
        cuando el usuario cierra la ventana desde el gestor de
        ventanas, con gtk_main_quit (La función que
        provoca que el bucle gtk_main salga, y consecuentemente
        termine la aplicación)*/

        gtk_signal_connect (GTK_OBJECT (app), "delete_event",
                            GTK_SIGNAL_FUNC (delete_event),
                            NULL);

        /* crea una caja horizon-
tal para los botones y la añade dentro
        de la aplicación del widget*/

        hbox = gtk_hbox_new (FALSE,5);
        gnome_app_set_contents (GNOME_APP (app), hbox);

        /* crea un botón y lo añade a la caja horizontal */
        button = gtk_button_new_with_label("Botón 1");
        gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
                            GTK_SIGNAL_FUNC (button_clicked),
                            "Botón 1\n");

        /* y otro botón*/
        button = gtk_button_new_with_label("Botón 2");
        gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
                            GTK_SIGNAL_FUNC (button_clicked),
                            "Botón 2\n");

        /* muestra todo dentro de este widget y el widget mismo */
        gtk_widget_show_all(app);

        /* entra en el bucle principal */
        gtk_main ();

        return 0;
}
```

Por favor, observe el uso de *gnome_init* en lugar de *gtk_init*, y del widget *GnomeApp* en lugar del *GtkWindow* habitual. Entraremos en detalle sobre esto mas tarde.

Descripción

Ahora veremos las diferentes bibliotecas que vamos a cubrir. Primero echaremos un vistazo a la biblioteca *libgnome* y a sus funcionalidades, después describiremos *libgnomeui* para completar todas las funcionalidades básicas de un programa GNOME. Veremos también *gnome-canvas* en detalle: un widget extremadamente potente y útil.

Usando la Biblioteca libgnome

La biblioteca *libgnome* es la biblioteca de utilidades para aplicaciones GNOME no específica del conjunto de herramientas (GTK+). Incluye aspectos como lectura de archivos de configuración, manejo del archivo .desktop, funciones útiles especiales similares a las contenidas en GLib, obtención de la localización de los archivos estándar para GNOME, manejo de tipos MIME, manejo de meta-datos en archivos, sonido, "triggers", y otros aspectos útiles que uno querría usar en cualquier aplicación. Por ejemplo, si usted está escribiendo una aplicación en, digamos Motif, y quiere que su aplicación sea compatible con GNOME, entonces podría hacer uso de esta biblioteca para conseguirlo.

Archivos de Configuración

Las funciones *gnome-config* proporcionan una forma sencilla de almacenar información de configuración dentro de archivos. Para ver una lista completa de las funciones, mire en el archivo de cabecera *libgnome/gnome-config.h*.

Todas las funciones trabajan con un camino o ruta (N.T: path). El camino es semejante a un camino de Unix, y representa la ruta al archivo de configuración (empezando ésta desde el directorio *~/.gnome/*) y la clave que estamos buscando dentro. Así, */some/config/path/file/sectionA/keyB*, se refiere al archivo *~/.gnome/some/config/path/file*, y dentro del archivo usando la sección *sectionA* y la clave *keyB*.

Leyendo Información de Configuración

Para leer información de los archivos de configuración se usan las funciones *gnome_config_get_**. El * se cambia por el tipo de dato, el cual puede ser *int*, *float*, *string*, *bool* y *vector*. Las funciones *int* trabajan con *gint*, las funciones *float* trabajan con *gdouble*, las funciones *string* trabajan con *gchar **, las funciones *bool*trabajan con *gboolean* y las *vector* manejan un entero con un número de elementos y un vector de cadenas que contiene dichos elementos (*gint* y *gchar***). Las funciones *gnome_config_get_**, pueden aceptar valores por defecto si usted los añade a la ruta después de un signo '='. Si necesita saber si el valor por omisión fue usado, puede añadir un *_with_default* al nombre de la función y pasarle un *gboolean **, a través del cual la función devuelve si se empleó ese valor por defecto o si realmente se encontró otro valor válido en el archivo de configuración. A continuación se muestra un ejemplo:

```
gint contador;
gchar *texto;
```

```
gboolean def;
...
contador = gnome_config_get_int_with_default("/ejemplo/seccion/contador=1",
                                              &def);
if(def) g_print("Usamos el contador por defecto\n");
texto = gnome_config_get_string("/ejemplo/seccion/texto=TEXT");
...
g_free(text);
```

Observe que la cadena devuelta por *gnome_config_get_string* debería ser liberada con *g_free*, el vector de *gnome_config_get_vector* debería ser también liberado con *g_free*.

Escribiendo la Configuración

Para escribir la configuración en un archivo, se usan las funciones *gnome_config_set_**. Su uso es muy parecido a las anteriores *gnome_config_get_**. Los tipos se usan exactamente igual. Excepto con las funciones "set", se pasan los datos que se quieren almacenar después de la ruta o camino y no hay valor por omisión dentro de ella. Si el directorio en la ruta no existe, será creado cuando las funciones escriban en el disco. Después de configurar todos sus datos, necesitará llamar a *gnome_config_sync* para escribir sus datos a un archivo. La biblioteca no escribirá los datos a un archivo inmediatamente por razones de eficiencia. A continuación se muestra un ejemplo:

```
char *text;
int counter;
...
/*después de haber establecido un texto y un contador a algún valor,
podemos escribirlos a nuestro lugar de configuración*/
gnome_config_set_int("/example/section/counter",counter);
gnome_config_set_string("/example/section/text",text);
gnome_config_sync();
```

Funciones Privadas

Si quiere almacenar información comprometedora, que otros usuarios no deberían leer, use las funciones *gnome_config_private_**: éstas se comportan de igual modo que las anteriores, con la excepción de que estas escriben en un directorio privado llamado *~/.gnome_private* sobre el que se fuerzan los permisos 0700. Esto no es extremadamente seguro, pero debido a las descerebradas restricciones de exportación de los EEUU, no se puede usar encriptación. La función *gnome_config_sync* (y algunas otras) no tiene una equivalencia privada, ya que funciona para ambos tipos de funciones.

Usando Gnome-Config para Archivos Arbitrarios

Si desea usar *gnome-config* para lectura y escritura de archivos arbitrarios en el sistema de archivos (estos archivos estarán en formato *gnome-config*), usted puede añadir ′=′ al principio de la ruta y otro ′=′ al final del nombre del archivo. A continuación se muestra un ejemplo:

```
char buf[256];
...
/* escribe algunos datos en un archivo temporal */
```

```
g_snprintf(buf,256,"=%s=/seccion/clave",tmpnam(tmpnam));
gnome_config_set_int(buf,999);
gnome_config_sync();
```

Observe que realmente no tiene sentido usar las versiones privadas cuando se usa una ruta absoluta arbitraria, ya que no habrá ninguna diferencia entre privadas y normales.

Prefijos Automáticos

Algunas veces, especialmente si tiene una ruta larga, será útil tener una cadena dada que se pase de forma automática como prefijo a la ruta. Esto es para lo que están *gnome_config_push_prefix* y *gnome_config_pop_prefix*. Se pasa la cadena que ha de servir como prefijo a *gnome_config_push_prefix* y se llama a *gnome_config_pop_prefix* cuando haya acabado de usarlo. Observe que estas funciones son comunes a las funciones de configuración privadas y normales. Ejemplo:

```
gnome_config_push_prefix("/archivo/seccion/");
gnome_config_set_int("clave1",1);
gnome_config_set_int("clave2",2);
gnome_config_set_int("clave3",-88);
gnome_config_pop_prefix();
```

Miscelánea Gnome-Config

Si necesita borrar su archivo de configuración, debe usar *gnome_config_clean_file*. Esta función planificará qué archivo sera borrado en el próximo *gnome_config_sync*. Usted puede hacer un *gnome_config_clean_file* y después usar el archivo y hacer un *gnome_config_sync*, y esto tendrá el comportamiento esperado.

Si ha escrito o leído de un archivo y quiere sacarlo de memoria (los cambios no se guardarán en disco hasta hacer 'sync') use *gnome_config_drop_file*. Esto se puede usar para deshacer los cambios hechos a este archivo, o simplemente ahorrar recursos, ya que de otra forma *gnome-config* conservará una copia de los datos en memoria para un acceso más rápido.

Archivos .desktop

Los archivos *.desktop* son los que contienen información sobre programas. Estos archivos estan en formato *gnome-config* y son leídos internamente usando *gnome-config*. Su aplicación necesitará uno de estos archivos si quiere que sea accesible en el menú de GNOME.

Puede usar las funciones *gnome_desktop_entry_** para manipular esos archivos. Estas funciones trabajan con una estructura llamada *GnomeDesktopEntry* y debería de mirar al archivo de cabecera *libgnome/gnome-dentry.h* para ver el formato de esta estructura.

Las funciones básicas que se usan para manipular estos archivos son *gnome_desktop_entry_load* que devuelve una nueva estructura *GnomeDesktopEntry* asignada, *gnome_desktop_entry_launch* que toma la estructura *GnomeDesktopEntry* como un argumento y lanza el programa que esta describe y *gnome_desktop_entry_free* que libera la memoria asignada dentro de la estructura.

Un ejemplo de archivo .desktop para su aplicación podría ser algo así:

```
[Desktop Entry]
Name=Clock
Name[cz]=Hodiny
Comment=Digital Clock
Comment[cz]=Digitalni Hodiny
Exec=digital-clock
Icon=clock.png
Terminal=0
Type=Application
```

Habrá observado que hay traducciones al checo para los campos Name y Comment. Para otros programas GNOME observe su archivo .desktop, está normalmente en algún lugar bajo *<prefijo>/share/apps/*, el cual contiene la jerarquía del sistema de menús. Para que el sistema encuentre su icono debería estar situado dentro del directorio *<prefijo>/share/pixmaps*. Observe que 'prefijo' se refiere a la ruta donde GNOME fue instalado.

Utilidades y Archivos

Archivos

Hay una forma estándar de encontrar archivos que pertenezcan a la instalación de GNOME, usted no debería realmente usar su propio código para encontrarlos ya que dispone de funciones para obtener los nombres de los archivos para los iconos, sonido u otros datos. Estas funciones son sólo para encontrar archivos que fueron instalados con las bibliotecas GNOME. No hay en este momento funciones para tratar con los datos instalados por su aplicación. Dichas funciones son:

Tabla 3-1. Funciones para encontrar archivos

Prototipo	Descripción
char *gnome_libdir_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de la biblioteca o NULL si el archivo no existe
char *gnome_unconditional_libdir_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de la biblioteca
char *gnome_datadir_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de datos o NULL si el archivo no existe
char *gnome_unconditional_datadir_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de datos
char *gnome_sound_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de sonido o NULL si el archivo no existe
char *gnome_unconditional_sound_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de sonido
char *gnome_pixmap_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de 'pixmaps' o NULL si el archivo no existe

Prototipo	**Descripción**
char *gnome_unconditional_pixmap_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio 'pixmaps'
char *gnome_config_file (const char *nombre_arch)	Obtiene la ruta completa de un archivo en el directorio de configuración o NULL si el archivo no existe
char *gnome_unconditional_config_file (const char *filename)	Obtiene el path completo de un archivo en el directorio de configuración

Estas funciones devuelven una cadena cuya memoria se asignó con *g_malloc* así que debería usar *g_free* sobre la cadena cuando termine. Las funciones *gnome_unconditional_** no comprobarán si el archivo actualmente existe y siempre devolverán un nombre de archivo. Las funciones normales efectuarán comprobaciones y devolverán *NULL* si el archivo no existe. Así que no debería usar estas funciones cuando esté guardando. Este ejemplo obtendrá un pixmap del directorio estándar de pixmaps, por ejemplo necesitamos obtener el icono "gnome-help.png":

```
gchar *nombre;
...
nombre = gnome_pixmap_file("gnome-help.png");
if(!nombre) {
        g_warning("gnome-help.png ¡no existe!");
} else {
        /*aquí usamos el archivo*/
        ...
        g_free(nombre);
}
```

También son de interés las funciones (actualmente macros) *gnome_util_home_file* y *gnome_util_user_home*. *gnome_util_home_file* toma un argumento (cadena) y devuelve una nueva cadena precedida por el directorio de inicio del usuario ($HOME) seguido de '.gnome'. Así por ejemplo si le pasa, digamos *archivillo*, devolvería */home/jirka/.gnome/archivillo*. Parecida es *gnome_util_user_home*, toma un argumento y devuelve el archivo con solo el directorio de inicio añadido. Así que si se le pasa *.oculto*, devolvería */home/jirka/.oculto*.

Utilidades

Hay también numerosas funciones de GLib para hacer su vida mas fácil manejando archivos: *g_file_exists*, la cual toma un nombre de archivo y devuelve *TRUE* si existe o *FALSE* si no existe; o *g_concat_dir_and_file*, que toma un nombre de archivo y un nombre de directorio, teniendo en cuenta el '/' (esto es útil cuando se trabaja con cadenas donde no se quiere chequear el '/', sólo se quiere añadir un directorio a algún archivo u otro directorio). Observe que tendrá que usar *g_free* sobre la cadena que devuelven. Para más funciones de utilidad, mire dentro de *libgnome/gnome-util.h*, está muy bien comentada.

Tipos MIME

Algunas veces es útil saber el tipo MIME de un archivo. Se puede hacer esto usando la función *gnome_mime_type_or_default*, que toma dos argumentos: el nombre del archivo y una cadena con el tipo MIME por defecto que será devuelta si no se puede averiguar el tipo MIME del archivo. Se intenta descubrir el tipo de un archivo mirando en el nombre del mismo. La cadena que devuelve la función es un puntero a una base de datos interna y no debería liberarla ya que podría desembocar más tarde en una violación de segmento. También puede usar *gnome_mime_type*, que devolverá NULL si no puede averiguar el tipo MIME.

Es también posible trabajar con listas URI, como aquellas usadas en las acciones de "arrastrar y soltar". Normalmente, de una lista URI usted querrá extraer los nombres de los archivos que recibió. Para eso use la función *gnome_uri_list_extract_filenames*, la cual toma la lista URI como un argumento de tipo cadena, y devuelve una *GList** de cadenas asignadas dinámicamente. Una vez haya terminado con los archivos, deberá liberar las cadenas y la lista. Puede usar la función *gnome_uri_list_free_strings* que hace esto para usted.

En el siguiente ejemplo se implementa un manejador de "arrastrar y soltar" que acepta archivos y devuelve la información sobre el tipo MIME de los mismos, después usted podría escribir código que hiciera operaciones basadas en el tipo MIME de los archivos.

```
/* este es el manejador para la señal 'dragdatareceive', asum-
iendo que
   nuestro widget solo acepta el tipo MIME de datos "text/uri-list".
   Para una explicación más detallada de como implemen-
tar las operaciones
   de arrastrar y soltar, consulte la documentación de GTK+ */

static void
dnd_drop_internal (GtkWidget        *widget,
                   GdkDragContext   *context,
                   gint              x,
                   gint              y,
                   GtkSelectionData *selection_data,
                   guint             info,
                   guint             time)
{
        GList *files, *li;

        /*aquí extraemos los nombres de los archivos desde la lista
          URI que recibimos */
         files = gnome_uri_list_extract_filenames(selection_data-
>data);

        /*Pasamos por un bucle a los archivos y obten-
emos sus tipos MIME*/
        for(li = files; li!=NULL ; li = g_list_next(li)) {
                  char *mimetype;
                  char *filename = li->data;

    /*suponemos el tipo MIME del archivo*/
                  mimetype = gnome_mime_type(filename);

                  /*si no lo podemos averiguar, simplemente saltamos
                   al siguiente nombre de archivo*/
                  if(!mimetype) continue;

                  /*A continuación va el código que realmente puede
                   hacer algo basado en los tipos MIME del archivo que
                   recibimos*/
                  ...
        }
```

```
        /*Libera la lista de archivos que obtuvimos*/
        gnome_uri_list_free_strings (files);
}
```

Obsérvese lo fácil que es descubrir qué archivos obtuvo y de qué tipo eran. Ahora sólo tiene que añadir algún código en vez de los tres puntos ('...') para hacer algo útil con esa información.

Meta-Datos

Algunas veces es útil almacenar alguna información junto con el nombre de archivo, esto se puede hacer fácilmente con *gnome-metadata*: un conjunto de funciones para gestionar estos datos. Ya que Unix no soporta de forma nativa meta-datos, usted debe ayudarle. Por ejemplo si su aplicación copia, renombra o borra archivos, use las siguientes funciones para mantener la consistencia en los meta-datos:

Tabla 3-2. Funciones de Meta-Datos

Prototipo	Descripción
int gnome_metadata_rename (const char *desde, const char *hacia)	Notifica a la base de datos de los meta-datos que un archivo ha sido renombrado
int gnome_metadata_copy (const char *desde, const char *hacia)	Notifica a la base de datos de los meta-datos que un archivo ha sido copiado
int gnome_metadata_delete (const char *archivo)	Notifica a la base de datos de los metadatos que un archivo ha sido borrado
int gnome_metadata_set (const char *archivo, const char *nombre, int tam, const char *datos)	Establece los datos con la clave 'nombre', asociados con el archivo 'archivo'. El dato que se incluye es el apuntado por 'datos' y su tamaño esta representado en bytes por 'tam'. En caso de éxito se devuelve GNOME_METADATA_OK.
int gnome_metadata_get (const char *archivo, const char *nombre, int *tam, char **buffer)	Obtiene datos con la clave 'nombre', asociados con el archivo 'archivo'. Los datos son almacenados en 'buffer' y en 'tam' el tamaño de dicho buffer. En caso de éxito se devuelve GNOME_METADATA_OK.
char **gnome_metadata_list (const char *archivo)	Obtiene una lista de claves para las cuales hay datos asociados en 'archivo'. La lista será asignada dinámicamente y será terminada con una cadena NULL. Debe liberarla con g_strfreev

Tabla 3-3. Valores de Retorno de los Meta-Datos

Nombre	Descripción
GNOME_METADATA_OK	No hay error (Actualmente 0)

Nombre	Descripción
GNOME_METADATA_IO_ERROR	Error de Entrada/Salida u otro error de comunicaciones o almacenamiento de bajo nivel.
GNOME_METADATA_NOT_FOUND	Información no encontrada.

Estas funciones actualmente no hacen operaciones sobre los archivos, sólo cambian los meta-datos de forma consistente. Así que si su aplicación hace alguna de esas operaciones, debe notificar a la base de datos de los meta-datos los cambios. No debe confiar en que los datos van a ser almacenados. Sólo los datos no críticos deberían ser almacenados en los meta-datos, ya que si su aplicación no notifica a la base de datos los cambios con las funciones mencionadas anteriormente, se perderán los datos que haya asociado al archivo. Esas funciones devolverán 0 o *GNOME_METADATA_OK* si no hubo error, o un código de error en caso contrario (descritos anteriormente).

Si quiere usar los meta-datos para asociar información a los archivos, usted debe usar las funciones *gnome_metadata_set*, *gnome_metadata_remove* y *gnome_metadata_get*. De nuevo estas funciones devuelven un entero, el cual es un *GNOME_METADATA_OK* en caso de que no hubiera error, o uno de los códigos de error que se usaban en las funciones anteriores en caso contrario.

Las funciones operan con una clave que es una cadena para la cual almacenan una porción de datos. Los datos son representados por un entero para el tamaño y un puntero a una cadena. *gnome_metadata_set* toma como primer argumento el nombre del archivo, el nombre o la clave del dato como segundo argumento, el tamaño como el tercero y un puntero al dato actual como cuarto argumento. Esta función sólo asocia el dato para cada archivo y clave. *gnome_metadata_remove* borrara un elemento particular sobre un archivo, así que toma un archivo y después el nombre de la clave como segundo argumento. *gnome_metadata_get* toma el nombre del archivo como primer argumento y el nombre de la clave como segundo, después devuelve el tamaño del dato mediante un puntero a entero que se pasa con el tercer argumento y el actual dato a través de un puntero a puntero que se pasa como cuarto argumento. El dato devuelto es asignado dinámicamente y debería ser liberado después de su uso. A continuación se muestra un pequeño ejemplo (en la vida real debería comprobar también el valor devuelto por la función para los errores).

```
int tam;
char *data;
...
/*Establece algunos datos ficticios asociados a un archivo*/
gnome_metadata_set("/algun/nombre/archivo","bogus",5,"BLAH");
...
/*Recupera de nuevo los datos*/
gnome_metadata_get("/algun/nombre/archivo","bogus",&tam,&data);
```

Usando el Marco de Desarrollo GnomeApp

Aunque el marco de desarrollo de aplicaciones GnomeApp pertenece a la biblioteca libgnomeui, realmente merece ser tratado de forma independiente. GnomeApp es la pieza que transforma un programa en una verdadera aplicación GNOME. Hace la programación simple y elegante y dota a las aplicaciones de muchos rasgos distintivos, haciéndolas consistentes con el entorno y configurables por el usuario.

Programando ayudado sólo por GTK+, usted tendría que implementar multitud de aspectos en sus aplicaciones, reinventando la rueda en cada momento. Por contra, si emplea GnomeApp éste se hará cargo de la configuración de la IU (Interfaz de Usuario) por usted, permitiendo aún que el usuario pueda personalizarla y garantizando la consistencia entre diferentes aplicaciones.

Introducción a GnomeApp

GnomeApp es el widget básico detrás de cada aplicación en GNOME. Será su ventana principal y contendrá el documento sobre el que estaremos trabajando, los menús, barras de herramientas y barras de estado de la aplicación. Recordará así mismo la posición de las barras, permitiendo al usuario recuperarlas tal y como las dejó la última vez que utilizó el programa.

Creando una Ventana GnomeApp

Crear un nuevo widget *GnomeApp* es tan sencillo como llamar a *gnome_app_new* con el nombre de la aplicación, el cual es por lo general el nombre del ejecutable o cualquier otro que sea único relacionado con su programa, y el título de la ventana principal como argumentos. Luego usted deberá crear contenidos e ir añadiéndolos al widget *GnomeApp* mediante el método *gnome_app_set_contents*.

Añadir barras de menús, barras de herramientas y barras de estado no resulta más complicado: simplemente utilice *gnome_app_set_toolbar*, *gnome_app_set_menus* o *gnome_app_set_statusbar*. *gnome_app_set_toolbar* es adecuado para aplicaciones sencillas con tan solo una única barra de herramientas, si necesita mayor complejidad, use *gnome_app_add_toolbar*, que le permite añadir tantas barras de herramientas como precise.

Creación de Barras de Menús y de Herramientas

Creación Automática de Menús y Barras de Herramientas

En muchas de sus aplicaciones, usted no creará sus menús directamente sino que usará funciones de *libgnomeui/gnome-app-helper.h* que se ocupen de todo el proceso por usted. Todo lo que necesita es rellenar un par de estructuras con la información pertinente, llamar a *gnome_app_create_menus* o *gnome_app_create_toolbar* pasando dichas estructuras y ¡voila!, su aplicación tendrá ya menús y barras de herramientas. Algunas veces deseará pasar un puntero con la dirección de cierta información adicional a todas las retrollamadas que trabajen con esas estructuras, en tales ocasiones deberá usar *gnome_app_create_toolbar_with_data* y *gnome_app_create_menus_with_data*, y proveer un parámetro extra (El puntero).

Definición de la Estructura GnomeUIInfo

En este capítulo se describe la estructura que usted necesita rellenar para crear los menús de su aplicación (Realmente rellenará un vector de dichas estructuras). Se incluyen además los enumerados que precisará para completar la estructura.

```
/* Estos valores identifican la fuente pixmap que será agregado al
elemento del menú  */
typedef enum {
        GNOME_APP_PIXMAP_NONE,          /* Sin pixmap */
```

```
        GNOME_APP_PIXMAP_STOCK,         /* Usa un pixmap de GNOME
                                           (GnomeStock) */
        GNOME_APP_PIXMAP_DATA,          /* Usa un pixmap xpm inser-
tado en el
                                           código (Un vector) */
        GNOME_APP_PIXMAP_FILENAME       /* Usa el pixmap con-
tenido en el
                                           archivo especificado */
} GnomeUIPixmapType;

/* Esta es la estructura que define un elemento den-
tro de una barra de menús
 * o en una barra de herramientas. La idea es crear un vec-
tor de estas
 * estructuras con la información necesaria para crear los menús.
 * La forma más conveniente de trabajar con esta estruc-
tura es usar las macros
 * GNOMEUIINFO_* definidas más abajo.*/
typedef struct {
        GnomeUIInfoType type;     /* Tipo de elemento
        gchar *label;             /* Cadena usada en la etiqueta */
        gchar *hint;              /* Para los elemen-
tos pertenecientes a una
                                     barra de herramien-
tas se trata de la
                                     sugerencia. Para los elemen-
tos de un menú,
                                     el mensaje de la barra de es-
tado */
        gpointer moreinfo;        /* Para un elemento, un ele-
mento conmutable,
                                     o elemento de op-
ción, esto es un puntero a
                                     la función que será lla-
mada cuando el
                                     elemento sea acti-
vado. Para un subárbol,
                                     es un puntero a otro vec-
tor de estructuras
                                     GnomeUIInfo. Para un ele-
mento de opción
                                     principal, un puntero a un vec-
tor de
                                     estructuras GnomeUI-
Info que representa el
                                     grupo de elementos de op-
ción asociados.
                                     Para un elemento de ayuda, es-
pecifica el
                                     nodo que debe cargar (i.e. El
                                     identificador de la apli-
cación), NULL
                                     indica el nombre del pro-
grama principal.
                                     Si se trata de los datos del constructor,
                                     apunta a la estruc-
tura GnomeUIBuilderData
                                     para los elementos consecu-
tivos [N.T:
                                     haz que me entiendan, por fa-
vor :-)] */
        gpointer user_data;       /* Puntero a los datos que pasere-
mos a las
```

```
                                     retrollamadas */
        gpointer unused_data;      /* Reservado para una fu-
tura extensión,
                                     debería ser NULL por ahora */
        GnomeUIPixmapType pixmap_type;  /* Tipo de pixmap para
                                           el elemento */
        gpointer pixmap_info;      /* Puntero a la información del
                                     pixmap:

                                     Para GNOME_APP_PIXMAP_STOCK, un
                                     puntero al nombre del icono.

                                     Para GNOME_APP_PIXMAP_DATA, un
                                     puntero al xpm (Al vector).

                                     Para GNOME_APP_PIXMAP_FILENAME, un
                                     puntero al nombre del archivo
                                     (Una cadena) */
        guint accelera-
tor_key;    /* Atajo del teclado, o 0 si no hay */
        GdkModifierType ac_mods;   /* Máscara de las teclas modifi-
cadas por
                                     el atajo */

        GtkWidget *widget;         /* Rel-
lenado por gnome_app_create*,
                                     podrá ajus-
tarlo una vez haya sido
                                     creado */
} GnomeUIInfo;
```

No se preocupe si le cuesta recordar todos los elementos o no comprende lo que significan. Si usted no sabe lo que representa un miembro de la estructura, sencillamente dejeló en NULL o 0. También puede recurrir a copiar el menú de otra aplicación que haga lo que usted desea y modificarlo un poco para ajustarlo a su propio desarrollo, esto es a menudo mejor y más rápido que perder el tiempo hurgando en la estructura.

Macros Útiles

La mayoría de las veces, crear entradas en un menú será muy simple gracias a ciertas macros. Por ejemplo, para finalizar un menú, usaríamos la macro *GNOMEUIINFO_END* o para insertar un separador *GNOMEUIINFO_SEPARATOR*. El resto de los elementos pueden ser también creados mediante macros, aunque deberemos proporcionar cierta información a las mismas. Por ejemplo, si desea crear un elemento con un xpm insertado en código, usted puede usar la macro *GNOMEUIINFO_ITEM(etiqueta, sugerencia, retrollamada, xpm_data)*, donde etiqueta es el texto que aparecerá en el menú, sugerencia es la ayuda que recibirá el usuario cuando se coloque sobre el elemento (*NULL* indica ninguna sugerencia), retrollamada es la función que es invocada cuando el usuario presiona el elemento, y xpm_data es un puntero al vector xpm que usted desea usar como icono. Si no quiere que aparezca el icono, use *GNOMEUIINFO_ITEM_NONE(etiqueta, sugerencia, retrollamada)*. Si lo que busca es añadir un elemento con un icono estándar de GNOME (De estos hablaremos más adelante) deberá usar *GNOMEUIINFO_ITEM_STOCK(etiqueta, sugerencia, retrollamada, stock_id)* donde stock_id es el identificador del icono que quiera usar. Luego, para crear su barra de menús principal o para insertar submenús dentro ella, puede usar *GNOMEUIINFO_SUBTREE(etiqueta, arbol)* y

GNOMEUIINFO_SUBTREE_STOCK(etiqueta, arbol, stock_id), donde 'arbol' es el vector de estructuras *GnomeUIInfo* que usted quiere emplear como submenú. Existen otras pocas macros que también manejan esta estructura, pero por lo general las que se usan más a menudo son éstas. Como ha visto, no precisa comprender la compleja estructura *GnomeUIInfo* y sus miembros para crear efectivos menús.

Macros para crear Elementos y Menús Genéricos

Todas las aplicaciones contienen un par de menús genéricos, así que para mantener una cierta consistencia, se aportan una serie de macros que rellenaran estos menús todo por usted, sólo tendrá que escribir las retrollamas necesarias y adaptarlos a un poco a su aplicación. Las ventajas de usar macros son consistencia entre aplicaciones, posibilidad de configuración por parte del usuario e internacionalización de los programas.

Elementos de Menú

La mayoría de estas macros tienen la forma: *GNOMEUIINFO_MENU_<nombre>_ITEM (retrollamada, datos)*. Hay una excepción, la macro que crea el elemento "Nuevo xxx" (N.T: nuevo archivo de un procesador de textos, p.ej). La guía de estilo de GNOME especifica que la palabra "Nuevo" debe estar dentro de la etiqueta del elemento. No se menciona ni el nombre completo del elemento ni su sugerencia particular, a diferencia del resto de los elementos genéricos. Para permitir que el programador pueda ajustar estos campos, se opta por utilizar una macro con una sintaxis particular para el elemento "Nuevo xxx": *GNOMEUIINFO_MENU_NEW_ITEM(etiqueta, sugerencia, retrollamada, datos)*. La "etiqueta" debería comenzar con "Nuevo ". Téngase en cuenta que si usted tiene más elementos "Nuevo", necesita usar la macro que crea un subárbol para elementos de este tipo, este procedimiento será explicado más tarde.

Tabla 3-4. El Menú Archivo

Macro	Descripción
GNOMEUI-INFO_MENU_NEW_ITEM(etiqueta, suger, cb, datos)	Elemento "Nuevo" (Necesita una etiqueta y una sugerencia)
GNOMEUI-INFO_MENU_OPEN_ITEM(cb, datos)	Elemento "Abrir"
GNOMEUI-INFO_MENU_SAVE_ITEM(cb, datos)	Elemento "Guardar"
GNOMEUI-INFO_MENU_SAVE_AS_ITEM(cb, datos)	Elemento "Guardar como"
GNOMEUI-INFO_MENU_REVERT_ITEM(cb, datos)	Elemento "Revertir"
GNOMEUI-INFO_MENU_PRINT_ITEM(cb, datos)	Elemento "Imprimir"
GNOMEUI-INFO_MENU_PRINT_SETUP_ITEM(cb, datos)	Elemento "Ajustes para impresión"

Macro	Descripción
GNOMEUIINFO_MENU_CLOSE_ITEM(cb, datos)	Elemento "Cerrar"
GNOMEUIINFO_MENU_EXIT_ITEM(cb, datos)	Elemento "Salir"

Tabla 3-5. El Menú Editar

Macro	Descripción
GNOMEUIINFO_MENU_CUT_ITEM(cb, datos)	Elemento "Cortar"
GNOMEUIINFO_MENU_COPY_ITEM(cb, datos)	Elemento "Copiar"
GNOMEUIINFO_MENU_PASTE_ITEM(cb, datos)	Elemento "Pegar"
GNOMEUIINFO_MENU_SELECT_ALL_ITEM(cb, datos)	Elemento "Marcar Todo"
GNOMEUIINFO_MENU_CLEAR_ITEM(cb, datos)	Elemento "Limpiar"
GNOMEUIINFO_MENU_UNDO_ITEM(cb, datos)	Elemento "Deshacer"
GNOMEUIINFO_MENU_REDO_ITEM(cb, datos)	Elemento "Rehacer"
GNOMEUIINFO_MENU_FIND_ITEM(cb, datos)	Elemento "Buscar"
GNOMEUIINFO_MENU_FIND_AGAIN_ITEM(cb, datos)	Elemento "Buscar Otra vez"
GNOMEUIINFO_MENU_REPLACE_ITEM(cb, datos)	Elemento "Reemplazar"
GNOMEUIINFO_MENU_PROPERTIES_ITEM(cb, datos)	Elemento "Propiedades", las propiedades del objeto manipulado

Tabla 3-6. El Menú Opciones

Macro	Descripción
GNOMEUIINFO_MENU_PREFERENCES_ITEM(cb, datos)	Elemento "Preferencias", ajustar preferencias de la aplicación

Tabla 3-7. El Menú Ventana

Macro	Descripción
GNOMEUI-INFO_MENU_NEW_WINDOW_ITEM(cb, datos)	Elemento "Nueva ventana"
GNOMEUI-INFO_MENU_CLOSE_WINDOW_ITEM(cb, datos)	Elemento "Cerrar ventana"

Tabla 3-8. El Menú Ayuda

Macro	Descripción
GNOMEUI-INFO_MENU_ABOUT_ITEM(cb, datos)	Elemento "Acerca de"

Tabla 3-9. El Menú Juego

Macro	Descripción
GNOMEUI-INFO_MENU_NEW_GAME_ITEM(cb, datos)	Elemento "Nuevo juego"
GNOMEUI-INFO_MENU_PAUSE_GAME_ITEM(cb, datos)	Elemento "Pausa"
GNOMEUI-INFO_MENU_RESTART_GAME_ITEM(cb, datos)	Elemento "Reiniciar"
GNOMEUI-INFO_MENU_UNDO_MOVE_ITEM(cb, datos)	Elemento "Anular"
GNOMEUI-INFO_MENU_REDO_MOVE_ITEM(cb, datos)	Elemento "Rehacer"
GNOMEUI-INFO_MENU_HINT_ITEM(cb, datos)	Elemento "Sugerencia"
GNOMEUI-INFO_MENU_SCORES_ITEM(cb, datos)	Elemento "Puntuación"
GNOMEUI-INFO_MENU_END_GAME_ITEM(cb, datos)	Elemento "Terminar juego"

Menús, árboles y subárboles

Ya hemos mencionado anteriormente el subárbol "Nuevo". Para usarlo, debería utilizar la macro *GNOMEUIINFO_MENU_NEW_SUBTREE (arbol)*, donde arbol es otro vector de estructuras GnomeUIInfo con diferentes elementos "Nuevo".

Existen también menús de "primer nivel" (N.T: toplevel) genéricos, entendiéndose como menús de "primer nivel" aquellos que se encuentran directamente sobe la barra de menús. El mecanismo para trabajar con ellos es el mismo: debe pasar el vector de estructuras GnomeUIInfo a la macro correspondiente.

Tabla 3-10. Macros para crear Menús de Primer Nivel.

Macro	Descripción
GNOMEUIINFO_MENU_FILE_TREE(arbol)	Menú "Archivo"
GNOMEUIINFO_MENU_EDIT_TREE(arbol)	Menú "Editar"
GNOMEUIINFO_MENU_VIEW_TREE(arbol)	Menú "Presentación"
GNOMEUIINFO_MENU_SETTINGS_TREE(arbol)	Menú "Opciones"
GNOMEUIINFO_MENU_FILES_TREE(arbol)	Menú "Archivos"
GNOMEUIINFO_MENU_WINDOWS_TREE(arbol)	Menú "Ventanas"
GNOMEUIINFO_MENU_HELP_TREE(arbol)	Menú "Ayuda"
GNOMEUIINFO_MENU_GAME_TREE(arbol)	Menú "Juego"

A veces, usted deseará trabajar con la ruta (N.T: path) de un elemento de cierto menú, por ejemplo para añadir elementos al menú "Documentos". Para esto, necesitará las macros de la forma *GNOME_MENU_<nombre>_STRING* y *GNOME_MENU_<nombre>_PATH*. Dichas macros se expandirán a la cadena apropiada. La macro que termina en *_STRING* se expandirá al nombre del menú, y la macro que finaliza en *_PATH* al nombre del menú seguido de un "/". <nombre> puede ser uno de los siguientes identificadores: FILE, EDIT, VIEW, SETTINGS, NEW, FILES o WINDOWS.

Menú de Ayuda

Su aplicación debería contener un menú de ayuda, que podría ser definido de esta manera:

```
GNOMEUIINFO_HELP("mi_aplicación"),
GNOMEUIINFO_MENU_ABOUT_ITEM(retrollamada, datos),
GNOMEUIINFO_END
```

La macro GNOMEUIINFO_HELP toma el nombre de su aplicación e inspecciona los archivos de ayuda de GNOME buscando el apropiado. *FIXME[ORIGINAL]:*

necesitamos añadir alguna sección acerca de los archivos de ayuda y algunos ejemplos.

Ejemplo

Una aplicación muy sencilla que hace uso de todo lo estudiado hasta ahora:

```
/*
 * Un sencillo no integrado en el árbol GNOME, sin usar i18n
 * uiinfo.c
 */
/* La cabecera GNOME más básica */
#include <gnome.h>

/* Una retrollamada para los botones */
static void
a_callback(GtkWidget *button, gpointer data)
{
        /* Imprime una cadena cada vez que es llamada */
        g_print("Dentro de la Retrollamada\n");
}

GnomeUIInfo file_menu[] = {
        GNOMEUIINFO_MENU_EXIT_ITEM(gtk_main_quit,NULL),
        GNOMEUIINFO_END
};

GnomeUIInfo some_menu[] = {
        GNOMEUIINFO_ITEM_NONE("_Menuitem","Un elemento de menú",
                              a_callback),
        GNOMEUIINFO_SEPARATOR,
        GNOMEUIINFO_ITEM_NONE("M_enuitem2","Otro elemento de menú",
                              a_callback),
        GNOMEUIINFO_END
};

GnomeUIInfo menubar[] = {
        GNOMEUIINFO_MENU_FILE_TREE(file_menu),
        GNOMEUIINFO_SUBTREE("_Un menú",some_menu),
        GNOMEUIINFO_END
};

GnomeUIInfo toolbar[] = {
        GNOMEUIINFO_ITEM_STOCK("Salir","Salir del programa",
                               gtk_main_quit,
                               GNOME_STOCK_PIXMAP_EXIT),
        GNOMEUIINFO_END
};

int
main(int argc, char *argv[])
{
        GtkWidget *app;
        GtkWidget *button;
        GtkWidget *hbox;
        GtkWidget *label;

        /* Inicializa GNOME, muy similar a gtk_init */
        gnome_init ("menu-basic-example", "0.1", argc, argv);

        /* Crea un widget GnomeApp, el cual crea una ventana
           para su aplicación */
```

```
        app = gnome_app_new ("menu-basic-example",
                             "Aplicación básica GNOME");

        /* Engancha "delete_event", el evento que es emitido
           cuando el usuario cierra la ventana, a gtk_main_quit,
           que es la función que fuerza a salir del bucle
           gtk_main y finaliza la aplicación */
        gtk_signal_connect (GTK_OBJECT (app), "delete_event",
                            GTK_SIGNAL_FUNC (gtk_main_quit),
                            NULL);

        /* Crea una etiqueta como contenido */
        label = gtk_label_new("BLAH BLAH BLAH BLAH BLAH");

        /* Añade la etiqueta a la ventana */
        gnome_app_set_contents (GNOME_APP (app), label);

        /* Crea los menús para la aplicación */
        gnome_app_create_menus (GNOME_APP (app), menubar);

        /* Crea la barra de herramientas de la aplicación */
        gnome_app_create_toolbar (GNOME_APP (app), toolbar);

        /* Muestra todos los contenidos de app y a él mismo */
        gtk_widget_show_all(app);

        /* Entra en el bucle principal */
        gtk_main ();

        return 0;
}
```

¡Voila!, una aplicación con un menú y una barra de herramientas. Como puede ver, añadir nuevos elementos en los menús se reduce a añadir más definiciones al vector de estructuras GnomeUIInfo correspondiente.

Atajos del Teclado

Probablemente haya notado los guiones bajos en las etiquetas de los elementos del menú, estos especifican los atajos del teclado asociados a ese menú concreto. La forma en la que funcionan los atajos es muy similar a como lo hacen en otros entornos gráficos del mercado: podrá seleccionar un elemento determinado con sólo *<tecla>* si se encuentra dentro del menú o *alt-<tecla>* en caso contrario.

GnomeAppBar, la Barra de Estado

Toda aplicación debería incluir una barra de estado en la parte baja de la ventana principal. Esta barra debería mostrar mensajes y advertencias acerca del funcionamiento actual de la aplicación. Usted podrá manejarla a su antojo usando las funciones de mensajes de GnomeApp (descritas en la sección Dialogando con el Usuario). La barra de estado también puede aportar ayuda acerca de los elementos del menú o una barra de progreso. Estas son las funciones más importantes que trabajan con GnomeAppBar.

Tabla 3-11. Métodos Importantes de GnomeAppBar

Prototipo	Descripción
GtkWidget * gnome_appbar_new (gboolean hay_progreso, gboolean hay_estado, GnomePreferencesType interactividad)	Crea un nuevo widget GnomeAppBar. opcionalmente con una barra de progreso si 'hay_progreso' es TRUE, y una barra de estado si 'hay_estado' es TRUE. La 'interactividad' nos dice si la barra ha de ser interactiva, lo cual puede ser nunca, siempre o depender de las preferencias del usuario, GNOME_PREFERENCES_USER, que es posiblemente la mejor opción. Piense que la implementación de la barra de estado interactiva aún no esta finalizada
void gnome_appbar_set_status (GnomeAppBar * appbar, const gchar * estado)	Fija el texto de la barra de estado, sin alterar appbar
void gnome_appbar_set_default (GnomeAppBar * appbar, const gchar * texto_por_defecto)	Fija el texto por defecto de la barra de estado, el texto aparecerá cuando no haya necesidad de mostrar nada
void gnome_appbar_push (GnomeAppBar * appbar, const gchar * estado)	Apila un mensaje
void gnome_appbar_pop (GnomeAppBar * appbar)	Saca un mensaje de la pila de mensajes de estado
void gnome_appbar_clear_stack (GnomeAppBar * appbar)	Vacia la pila de mensajes de estado
void gnome_appbar_refresh (GnomeAppBar * appbar)	Refresca la barra de estado y fija el actual valor de la pila al valor por defecto. Útil para limpiar mensajes provenientes del método *gnome_appbar_set_status*
void gnome_appbar_set_progress (GnomeAppBar * appbar, gfloat porcentaje)	Ajusta el porcentaje de la barra de progreso
GtkProgress * gnome_appbar_get_progress (GnomeAppBar * appbar)	Obtiene el widget GtkProgress, que es la barra de progreso, para poder manipularlo directamente

Par añadir una barra de aplicación a una ventana GnomeApp, use el método *gnome_app_set_statusbar*. Para poder visualizar las sugerencias del menú en la barra, debe llamar a *gnome_app_install_menu_hints* con un puntero a su definición GnomeUIInfo de la barra de menús principal. Por ejemplo, podríamos añadir dicha funcionalidad con el siguiente código:

```
GtkWidget *w;

w = gnome_appbar_new(FALSE, TRUE, GNOME_PREFERENCES_USER);
gnome_app_set_statusbar(GNOME_APP(app), w);

gnome_app_install_menu_hints(GNOME_APP(app), menubar);
```

Suponiendo que 'app' es su ventana de aplicación GnomeApp y 'menubar' es un puntero GnomeUIInfo a su definición de la barra de menús principal de la aplicación.

La forma en la que trabaja la barra de estado es amontonando los mensajes en una pila: el mensaje más reciente es el que se muestra en pantalla. Esto es útil ya que en muchas ocasiones le interesará mantener un mensaje concreto en la barra la mayor parte del tiempo e ir mostrando pequeños mensajes de vez en cuando. El nuevo mensaje será apilado sobre el principal y una vez mostrado cederá paso al mensaje original, que volverá así a estar en la cima de la pila y a aparecer en la barra.

Debe tener en cuenta que la barra de estado puede hacerse cargo de muchos detalles automáticamente, sin precisar nuestra supervisión directa. Por ejemplo, de mostrar mensajes informativos en la barra se ocupa *gnome_app_flash* descrito más tarde en la sección Dialogando con el Usuario.

Dialogando con el Usuario

En ocasiones, usted deseará comunicarse con el usuario, puede que necesite mostrar un error o un mensaje indicando el estado de su operación. Existen opciones accesibles al usuario que marcan dónde y cómo se recibe esta información. Si quiere que su código se comporte de forma consistente con esta característica, deberá usar las siguientes funciones cuando precise crear un nuevo cuadro de diálogo.

Tabla 3-12. Funciones de Mensajes de GnomeApp

Prototipo	Descripción
GtkWidget * gnome_app_message (GnomeApp * app, const gchar * mensaje)	Muestra un mensaje y solicita confirmación al usuario. Si se trata de un diálogo, devolverá un puntero a dicho diálogo que puede ser obviado sin problemas a menos que desee hacer algo con él posteriormente.
void gnome_app_flash (GnomeApp * app, const gchar * flash)	Muestra un mensaje en la barra de estado durante un par de segundos. Evidentemente, su uso debería restringirse a mensajes no críticos o no necesarios para el correcto uso de la aplicación.
GtkWidget * gnome_app_error (GnomeApp * app, const gchar * error)	Similar a gnome_app_message, pero muestra un error al usuario en vez de un mensaje.
GtkWidget * gnome_app_warning (GnomeApp * app, const gchar * warning)	También similar a gnome_app_message, esta vez aparece una advertencia.

El siguiente ejemplo muestra como usar estas funciones.

```
FILE *fp;

fp = fopen(archivo,"r");
if(!fp) {
        char *err = g_strdup_printf(_("No pude abrir: %s"),
                                    archivo);
        gnome_app_error(GNOME_APP(app),err);
        g_free(err);
        ...
}
```

Puede que lo que necesite su aplicación es solicitar datos del usuario. Las siguientes funciones cubren este supuesto y reciben como argumento un puntero a una función, del tipo GnomeReplyCallback o GnomeStringCallback. La función (el contenido del puntero) deber tener como parámetros un entero o una cadena de caracteres y un puntero de datos. No cuente con que la retrollamada sea invocada siempre, pudiera ser que la línea de estado estuviese ocupada en cuyo caso se obviará la retrollamada y se devolverá NULL. Si la función se ejecuta devolverá el widget de diálogo. Otro aspecto destacable es que se trata de funciones no bloqueantes, lo que quiere decir que no se debe actuar una vez se han llamado como si ya hubieran acabado su ejecución.

Tabla 3-13. Funciones de Petición de Datos de GnomeApp

Prototipo	Descripción
GtkWidget * gnome_app_question (GnomeApp * app, const gchar * pregunta, GnomeReplyCallback retrollamada, gpointer datos)	Una pregunta cuya contestación es sí o no. La retrollamada de respuesta recibe 0 ó 1 respectivamente.
GtkWidget * gnome_app_question_modal (GnomeApp * app, const gchar * pregunta, GnomeReplyCallback retrollamada, gpointer datos)	Similar a la anterior, aunque permite al usuario interactuar con el resto de la aplicación a pesar de que no haya respondido o no haya cerrado el diálogo.
GtkWidget * gnome_app_ok_cancel (GnomeApp * app, const gchar * mensaje, GnomeReplyCallback retrollamada, gpointer datos)	Una pregunta del tipo Sí o Cancelar.
GtkWidget * gnome_app_ok_cancel_modal (GnomeApp * app, const gchar * mensaje, GnomeReplyCallback retrollamada, gpointer datos)	La versión modal de la función gnome_app_ok_cancel.
GtkWidget * gnome_app_request_string (GnomeApp * app, const gchar * inductor, GnomeStringCallback retrollamada, gpointer datos)	Pregunta al usuario con un inductor de órdenes (N.T: prompt) igual a 'inductor'. La retrollamada recibirá NULL si el usuario cancela el diálogo. La memoria que ocupa la cadena que recibe la retrollamada deberá ser liberada por el programador.
GtkWidget * gnome_app_request_password (GnomeApp * app, const gchar * inductor, GnomeStringCallback retrollamada, gpointer datos)	Pide al usuario una contraseña, mostrando 'inductor'. Idéntica a gnome_app_request_string, con la salvedad de que el texto no se muestra en pantalla.

Por ejemplo, solicitaremos al usuario confirmación para eliminar un objeto.

```
static void
really_delete_handler(int respuesta, gpointer datos)
{
        GtkWidget *un_widget;

        un_widget = GTK_WIDGET(datos);

        if(respuesta == 0) {
```

```
                ... /* "Sí" ha sido seleccionado */
        }
}

...

/* Mostrar una pregunta del tipo Sí/No, pasare-
mos some_widget como el
   argumento 'datos' al manejador really_delete_handler */
gnome_app_question(GNOME_APP(app),_("¿Desea borrar el objeto?"),
                   really_delete_handler,
                   un_widget);
```

Usando la Biblioteca libgnomeui

Además de GnomeApp, la biblioteca libgnomeui contiene un gran número de otros objetos y widgets, incluyendo diálogos estándar, manejo de sesiones, MDI y otros útiles widgets. También contiene el widget GnomeCanvas que merece un trato aparte. Esta es la biblioteca que hace más fácil la vida de los programadores. Es importante que use estos widgets mejor que simplemente los de GTK+, ya que estos widgets le darán la consistencia y las características que espera el usuario de aplicaciones GNOME.

Iconos Predefinidos

Muchas veces deseará usar botones y opciones estándares (como pueden ser *Abrir* o *Guardar como...*), y también deseará proporcionar iconos en las opciones del menú, con los botones de la barra de herramientas o los botones de las ventanas de diálogo; para facilitar la navegación puede usar algunos de los iconos definidos de *gnome-libs*. A estos iconos se les llama *iconos predefinidos o iconos por defecto*. Ya ha visto un ejemplo de cómo usar iconos predefinidos en menús y barras de herramientas (usando la definición apropiada en *libgnomeui/ gnome-stock.h*). También hay botones predefinidos pudiendo usar un widget de botón basado en una descripción predefinida.

Aquí hay una lista de los iconos predefinidos *normales* de GNOME, estos tienen un tamaño normalizado para su uso en barras de herramientas y en otros lugares donde necesite un icono con tamaño normalizado. Están dados como definiciones de cadenas de caracteres constantes y su significado debería ser obvio.

```
#define GNOME_STOCK_PIXMAP_NEW         "New"
#define GNOME_STOCK_PIXMAP_OPEN        "Open"
#define GNOME_STOCK_PIXMAP_CLOSE       "Close"
#define GNOME_STOCK_PIXMAP_REVERT      "Revert"
#define GNOME_STOCK_PIXMAP_SAVE        "Save"
#define GNOME_STOCK_PIXMAP_SAVE_AS     "Save As"
#define GNOME_STOCK_PIXMAP_CUT         "Cut"
#define GNOME_STOCK_PIXMAP_COPY        "Copy"
#define GNOME_STOCK_PIXMAP_PASTE       "Paste"
#define GNOME_STOCK_PIXMAP_PROPERTIES  "Properties"
#define GNOME_STOCK_PIXMAP_PREFERENCES "Preferences"
#define GNOME_STOCK_PIXMAP_HELP        "Help"
#define GNOME_STOCK_PIXMAP_SCORES      "Scores"
#define GNOME_STOCK_PIXMAP_PRINT       "Print"
#define GNOME_STOCK_PIXMAP_SEARCH      "Search"
#define GNOME_STOCK_PIXMAP_SRCHRPL     "Search/Replace"
```

```
#define GNOME_STOCK_PIXMAP_BACK        "Back"
#define GNOME_STOCK_PIXMAP_FORWARD     "Forward"
#define GNOME_STOCK_PIXMAP_FIRST       "First"
#define GNOME_STOCK_PIXMAP_LAST        "Last"
#define GNOME_STOCK_PIXMAP_HOME        "Home"
#define GNOME_STOCK_PIXMAP_STOP        "Stop"
#define GNOME_STOCK_PIXMAP_REFRESH     "Refresh"
#define GNOME_STOCK_PIXMAP_UNDO        "Undo"
#define GNOME_STOCK_PIXMAP_REDO        "Redo"
#define GNOME_STOCK_PIXMAP_TIMER       "Timer"
#define GNOME_STOCK_PIXMAP_TIMER_STOP  "Timer Stopped"
#define GNOME_STOCK_PIXMAP_MAIL        "Mail"
#define GNOME_STOCK_PIXMAP_MAIL_RCV    "Receive Mail"
#define GNOME_STOCK_PIXMAP_MAIL_SND    "Send Mail"
#define GNOME_STOCK_PIXMAP_MAIL_RPL    "Reply to Mail"
#define GNOME_STOCK_PIXMAP_MAIL_FWD    "Forward Mail"
#define GNOME_STOCK_PIXMAP_MAIL_NEW    "New Mail"
#define GNOME_STOCK_PIXMAP_TRASH       "Trash"
#define GNOME_STOCK_PIXMAP_TRASH_FULL  "Trash Full"
#define GNOME_STOCK_PIXMAP_UNDELETE    "Undelete"
#define GNOME_STOCK_PIXMAP_SPELLCHECK  "Spellchecker"
#define GNOME_STOCK_PIXMAP_MIC         "Microphone"
#define GNOME_STOCK_PIXMAP_LINE_IN     "Line In"
#define GNOME_STOCK_PIXMAP_CDROM       "Cdrom"
#define GNOME_STOCK_PIXMAP_VOLUME      "Volume"
#define GNOME_STOCK_PIXMAP_BOOK_RED    "Book Red"
#define GNOME_STOCK_PIXMAP_BOOK_GREEN  "Book Green"
#define GNOME_STOCK_PIXMAP_BOOK_BLUE   "Book Blue"
#define GNOME_STOCK_PIXMAP_BOOK_YELLOW "Book Yellow"
#define GNOME_STOCK_PIXMAP_BOOK_OPEN   "Book Open"
#define GNOME_STOCK_PIXMAP_ABOUT       "About"
#define GNOME_STOCK_PIXMAP_QUIT        "Quit"
#define GNOME_STOCK_PIXMAP_MULTIPLE    "Multiple"
#define GNOME_STOCK_PIXMAP_NOT         "Not"
#define GNOME_STOCK_PIXMAP_CONVERT     "Convert"
#define GNOME_STOCK_PIXMAP_JUMP_TO     "Jump To"
#define GNOME_STOCK_PIXMAP_UP          "Up"
#define GNOME_STOCK_PIXMAP_DOWN        "Down"
#define GNOME_STOCK_PIXMAP_TOP         "Top"
#define GNOME_STOCK_PIXMAP_BOTTOM      "Bottom"
#define GNOME_STOCK_PIXMAP_ATTACH      "Attach"
#define GNOME_STOCK_PIXMAP_INDEX       "Index"
#define GNOME_STOCK_PIXMAP_FONT        "Font"
#define GNOME_STOCK_PIXMAP_EXEC        "Exec"

#define GNOME_STOCK_PIXMAP_ALIGN_LEFT  "Left"
#define GNOME_STOCK_PIXMAP_ALIGN_RIGHT "Right"
#define GNOME_STOCK_PIXMAP_ALIGN_CENTER "Center"
#define GNOME_STOCK_PIXMAP_ALIGN_JUSTIFY "Justify"

#define GNOME_STOCK_PIXMAP_TEXT_BOLD    "Bold"
#define GNOME_STOCK_PIXMAP_TEXT_ITALIC  "Italic"
#define GNOME_STOCK_PIXMAP_TEXT_UNDERLINE "Underline"
#define GNOME_STOCK_PIXMAP_TEXT_STRIKEOUT "Strikeout"

#define GNOME_STOCK_PIXMAP_EXIT        GNOME_STOCK_PIXMAP_QUIT
```

Si quiere usarlo fuera de *GnomeUIInfo*, necesita obtener el widget con el pixmap. Lo que se hace es llamar a la función *gnome_stock_pixmap_widget* con su ventana principal como primer argumento (por eso que puede copiar su estilo) y el nombre del icono (uno de los definidos antes) como segundo argumento. Esto devuelve un nuevo widget que puede usar como un pixmap.

Para los menús puede usar la variedad *_MENU_* de los pixmaps predefinidos. Estos son más pequeños y son lo que debe usar para los elementos del menú por defecto en su definición de *GnomeUIInfo*.

```
#define GNOME_STOCK_MENU_BLANK        "Menu_"
#define GNOME_STOCK_MENU_NEW          "Menu_New"
#define GNOME_STOCK_MENU_SAVE         "Menu_Save"
#define GNOME_STOCK_MENU_SAVE_AS      "Menu_Save As"
#define GNOME_STOCK_MENU_REVERT       "Menu_Revert"
#define GNOME_STOCK_MENU_OPEN         "Menu_Open"
#define GNOME_STOCK_MENU_CLOSE        "Menu_Close"
#define GNOME_STOCK_MENU_QUIT         "Menu_Quit"
#define GNOME_STOCK_MENU_CUT          "Menu_Cut"
#define GNOME_STOCK_MENU_COPY         "Menu_Copy"
#define GNOME_STOCK_MENU_PASTE        "Menu_Paste"
#define GNOME_STOCK_MENU_PROP         "Menu_Properties"
#define GNOME_STOCK_MENU_PREF         "Menu_Preferences"
#define GNOME_STOCK_MENU_ABOUT        "Menu_About"
#define GNOME_STOCK_MENU_SCORES       "Menu_Scores"
#define GNOME_STOCK_MENU_UNDO         "Menu_Undo"
#define GNOME_STOCK_MENU_REDO         "Menu_Redo"
#define GNOME_STOCK_MENU_PRINT        "Menu_Print"
#define GNOME_STOCK_MENU_SEARCH       "Menu_Search"
#define GNOME_STOCK_MENU_SRCHRPL      "Menu_Search/Replace"
#define GNOME_STOCK_MENU_BACK         "Menu_Back"
#define GNOME_STOCK_MENU_FORWARD      "Menu_Forward"
#define GNOME_STOCK_MENU_FIRST        "Menu_First"
#define GNOME_STOCK_MENU_LAST         "Menu_Last"
#define GNOME_STOCK_MENU_HOME         "Menu_Home"
#define GNOME_STOCK_MENU_STOP         "Menu_Stop"
#define GNOME_STOCK_MENU_REFRESH      "Menu_Refresh"
#define GNOME_STOCK_MENU_MAIL         "Menu_Mail"
#define GNOME_STOCK_MENU_MAIL_RCV     "Menu_Receive Mail"
#define GNOME_STOCK_MENU_MAIL_SND     "Menu_Send Mail"
#define GNOME_STOCK_MENU_MAIL_RPL     "Menu_Reply to Mail"
#define GNOME_STOCK_MENU_MAIL_FWD     "Menu_Forward Mail"
#define GNOME_STOCK_MENU_MAIL_NEW     "Menu_New Mail"
#define GNOME_STOCK_MENU_TRASH        "Menu_Trash"
#define GNOME_STOCK_MENU_TRASH_FULL   "Menu_Trash Full"
#define GNOME_STOCK_MENU_UNDELETE     "Menu_Undelete"
#define GNOME_STOCK_MENU_TIMER        "Menu_Timer"
#define GNOME_STOCK_MENU_TIMER_STOP   "Menu_Timer Stopped"
#define GNOME_STOCK_MENU_SPELLCHECK   "Menu_Spellchecker"
#define GNOME_STOCK_MENU_MIC          "Menu_Microphone"
#define GNOME_STOCK_MENU_LINE_IN      "Menu_Line In"
#define GNOME_STOCK_MENU_CDROM        "Menu_Cdrom"
#define GNOME_STOCK_MENU_VOLUME       "Menu_Volume"
#define GNOME_STOCK_MENU_BOOK_RED     "Menu_Book Red"
#define GNOME_STOCK_MENU_BOOK_GREEN   "Menu_Book Green"
#define GNOME_STOCK_MENU_BOOK_BLUE    "Menu_Book Blue"
#define GNOME_STOCK_MENU_BOOK_YELLOW  "Menu_Book Yellow"
#define GNOME_STOCK_MENU_BOOK_OPEN    "Menu_Book Open"
#define GNOME_STOCK_MENU_CONVERT      "Menu_Convert"
#define GNOME_STOCK_MENU_JUMP_TO      "Menu_Jump To"
#define GNOME_STOCK_MENU_UP           "Menu_Up"
#define GNOME_STOCK_MENU_DOWN         "Menu_Down"
#define GNOME_STOCK_MENU_TOP          "Menu_Top"
#define GNOME_STOCK_MENU_BOTTOM       "Menu_Bottom"
#define GNOME_STOCK_MENU_ATTACH       "Menu_Attach"
#define GNOME_STOCK_MENU_INDEX        "Menu_Index"
#define GNOME_STOCK_MENU_FONT         "Menu_Font"
#define GNOME_STOCK_MENU_EXEC         "Menu_Exec"
```

```
#define GNOME_STOCK_MENU_ALIGN_LEFT     "Menu_Left"
#define GNOME_STOCK_MENU_ALIGN_RIGHT    "Menu_Right"
#define GNOME_STOCK_MENU_ALIGN_CENTER   "Menu_Center"
#define GNOME_STOCK_MENU_ALIGN_JUSTIFY  "Menu_Justify"

#define GNOME_STOCK_MENU_TEXT_BOLD      "Menu_Bold"
#define GNOME_STOCK_MENU_TEXT_ITALIC    "Menu_Italic"
#define GNOME_STOCK_MENU_TEXT_UNDERLINE "Menu_Underline"
#define GNOME_STOCK_MENU_TEXT_STRIKEOUT "Menu_Strikeout"

#define GNOME_STOCK_MENU_EXIT     GNOME_STOCK_MENU_QUIT
```

Si está construyendo el menú usted mismo y quiere utilizar un elemento de menú con icono y etiqueta predefinida, puede usar la función *gnome_stock_menu_item*. Toma el tipo de icono predefinido (uno de los definidos arriba) como primer argumento, el texto del menú como segundo argumento, y devuelve un widget recién creado de elemento de menú.

Hay botones predefinidos. Estos están para el uso en ventanas de diálogo (ver más adelante).

```
#define GNOME_STOCK_BUTTON_OK     "Button_Ok"
#define GNOME_STOCK_BUTTON_CANCEL "Button_Cancel"
#define GNOME_STOCK_BUTTON_YES    "Button_Yes"
#define GNOME_STOCK_BUTTON_NO     "Button_No"
#define GNOME_STOCK_BUTTON_CLOSE  "Button_Close"
#define GNOME_STOCK_BUTTON_APPLY  "Button_Apply"
#define GNOME_STOCK_BUTTON_HELP   "Button_Help"
#define GNOME_STOCK_BUTTON_NEXT   "Button_Next"
#define GNOME_STOCK_BUTTON_PREV   "Button_Prev"
#define GNOME_STOCK_BUTTON_UP     "Button_Up"
#define GNOME_STOCK_BUTTON_DOWN   "Button_Down"
#define GNOME_STOCK_BUTTON_FONT   "Button_Font"
```

Para tomar un widget de botón con el texto e icono de predefinido, puede usar la función *gnome_stock_button* con el tipo de botón (uno de los definidos arriba) como argumento. Ahora bien, algunas veces querrá hacer una mezcla de botones predefinidos y ordinarios, para conseguirlo llame a la función *gnome_stock_or_ordinary_button* con el tipo de botón predefinido y un texto para la etiqueta del botón. La función revisa si es una de las cadenas anteriores, y si no está crea un widget de botón ordinario con el texto como etiqueta. Aquí hay un ejemplo, crea tres botones empaquetándolos en una caja (no incluimos el código para hacer la caja), dos de los botones están predefinidos y el otro es normal:

```
GtkWidget *w;
GtkWidget *caja;
int i;
char *botones[]={
        GNOME_STOCK_BUTTON_OK,
        GNOME_STOCK_BUTTON_CANCEL,
        "Foo",
        NULL
};
...
/* bucle a través de todas las cadenas en el vector de botones*/
for(i = 0; botones[i] != NULL; i++) {
        /* creamos el botón, predefinido u ordinario */
        w = gnome_stock_or_ordinary_button(botones[i]);
```

```
        /* Lo mostramos y lo empaquetamos */
        gtk_widget_show(w);
        gtk_box_pack_start(GTK_BOX(caja),w,FALSE,FALSE,0);
 /* Deberíamos unir las señales y más cosas aquí */
        ...
}
```

Diálogos

Diálogos Genéricos

Si necesita crear su propio diálogo personalizado, *gnome-dialog* es el camino para hacerlo. Puede soportar tanto diálogos modales como no modales. De todas formas, es definitivamente mucho más cómodo para los usuarios de su programa si usa diálogos no modales, si es posible, porque los diálogos no modales tienden a tener problemas asociados, y algunas veces pueden causar errores extraños. Por ejemplo, si un diálogo no modal se asocia con una ventana, sería bueno conectar la señal `destroy` de la ventana para que también destruya la ventana del diálogo, de otra manera se podría producir un error al actuar sobre un documento o ventana que ya no existe. Sin embargo, los diálogos modales (los que son más fáciles de programar) normalmente son bastante engorrosos de usar, por eso evítelos si puede.

Para hacer un nuevo widget *GnomeDialog*, use la función *gnome_dialog_new*. Se pasa el título del diálogo como primer argumento, y después múltiples argumentos, con los títulos de los botones, terminando con un NULL. Los títulos de los botones también pueden ser las definiciones *GNOME_STOCK_BUTTON_**, si es que desea botones predefinidos en su diálogo. Ahora necesita dotar de un contenido al diálogo, el diálogo se crea con una caja vertical (*GtkVBox*) para que usted lo use simplemente con *GNOME_DIALOG(dialog)->vbox*. Con esto puede introducir el contenido.

Debería también establecer la ventana principal de la aplicación (su GnomeApp) como el padre del diálogo. Esto permite al gestor de ventanas manejar la ventana más apropiadamente que una simple ventana genérica. Se completa con la siguiente llamada:

```
gnome_dialog_set_parent(GNOME_DIALOG(dialog), GTK_WINDOW(app));
```

En este punto tiene que decidir si desea hacer un diálogo modal o no modal. En el caso de que quiera un diálogo modal, todo lo que necesita hacer es llamar a la función *gnome_dialog_run_and_close* y ella lo creará, esperará a que un usuario presione un botón o cierre el diálogo, y entonces cerrará el diálogo. En el caso de que no quiera que el diálogo se cierre cuando algún botón sea pulsado, use la función *gnome_dialog_run* y después de obtener un resultado, haga lo que necesite para ese botón en particular. Entonces, si desea continuar con el diálogo, vuelva a *gnome_dialog_run*, y si desea cerrarlo, use *gnome_dialog_close*. Aquí hay un ejemplo.

```
GtkWidget *dlg;
GtkWidget *etiqueta;
int i;
...
/*creamos un nuevo diálogo, NO olvide el NULL al final,
  ¡es muy importante!*/
dlg = gnome_dialog_new("Un diálogo",
                       GNOME_STOCK_BUTTON_OK,
```

```
                       GNOME_STOCK_BUTTON_APPLY,
                       GNOME_STOCK_BUTTON_CLOSE,
                       NULL);
/* asumimos que app es un puntero a nuestra ventana GnomeApp */
gnome_dialog_set_parent(GNOME_DIALOG(dlg), GTK_WINDOW(app));
...
/*añadimos algún contenido al diálogo aquí*/
etiqueta = gtk_label_new("Algún contenido aleatorio");
gtk_box_pack_start(GTK_BOX(GNOME_DIALOG(dlg)->vbox),etiqueta,
                   FALSE,FALSE,0);
...
/*hacemos un bucle infinito*/
for(;;) {
        i = gnome_dialog_run(GNOME_DIALOG(dlg));
        if(i == 0 || i == 2) {
                /* el usuario presionó OK o close, por eso
                saldremos del bucle y cerraremos el diálogo*/
                gnome_dialog_close(GNOME_DIALOG(dlg));
                break;
        } else if(i < 0) {
       /* el usuario cerró el diálogo desde el gestor
          de ventanas*/
                break;
        } else if(i == 1) {
       /*el usuario presionó aplicar, no queremos que
         se cierre*/
                ...
        }
}
```

Por defecto el diálogo se destruye cuando es cerrado, por eso no se tiene que preocupar por su destrucción. Puede cambiar esta conducta si lo desea.

Si va a hacer una ventana de diálogo no modal, las cosas se vuelven más complicadas. Cree un diálogo como antes, pero tiene que conectar la señal *clicked* del widget *GnomeDialog*. Esta señal tiene como su segundo argumento el número del botón que fue presionado. Después de esto debería usar la función *gnome_dialog_set_close* para decir a *GnomeDialog* que queremos cerrar el diálogo cuando el usuario presionó cualquier botón, si quiere este comportamiento. De otro modo deberá llamar a *gnome_dialog_close* en la función que atiende a la señal *clicked*, en los botones en los que se debe cerrar el diálogo. Después de esto sólo hay que mostrar el diálogo con *gtk_widget_show*. A continuación mostramos un ejemplo:

```
/*la función de tratamiento de clicked*/
static void
dialog_clicked(GnomeDialog *dlg, int boton, gpointer data)
{
        switch(boton) {
        case 1:
    /*el usuario presionó aplicar*/
                ...
                return;
        case 0:
    /*el usuario presionó OK*/
                ...
                  /*seguir hacia abajo para cerrar*/
        case 2:
    /*el usuario presionó cerrar*/
                gnome_dialog_close(dlg);
                break;
        }
```

```
}

/*en algún lugar del archivo fuente*/
...
GtkWidget *dlg;
...
/*crear un nuevo diálogo, NO olvide NULL al final, ¡es
  es muy importante!*/
dlg = gnome_dialog_new("Un diálogo",
                       GNOME_STOCK_BUTTON_OK,
                       GNOME_STOCK_BUTTON_APPLY,
                       GNOME_STOCK_BUTTON_CLOSE,
                       NULL);
/* asumimos que app es un puntero a nuestra ventana GnomeApp */
gnome_dialog_set_parent(GNOME_DIALOG(dlg), GTK_WINDOW(app));
...
/*añadimos algún contenido al diálogo aquí*/
...
/*asociamos la función de tratamiento de clicked*/
gtk_signal_connect(GTK_OBJECT(dlg),"clicked",
                   GTK_SIGNAL_FUNC(dialog_clicked),
                   NULL);
/*mostramos el diálogo, note que esto no es un diálogo
  modal, por eso el programa no se bloquea aquí, sigue*/
gtk_widget_show(dlg);
```

Esto implementa el mismo diálogo que el ejemplo modal anterior, sólo que no modal. Asegúrese de que tiene alguna forma de destruir el diálogo en caso de que no sea necesario, por ejemplo, si un diálogo va a modificar algún objeto, debería ser destruido cuando ese objeto es destruido.

Diálogos de mensaje

GnomeMessageBox es un objeto derivado de *GnomeDialog*. Se maneja exactamente de la misma manera, la única diferencia es que automáticamente se introduce una etiqueta simple y un icono del tipo del mensaje que contenga el diálogo. Los tipos de diálogos de mensaje son los que siguen:

```
#define GNOME_MESSAGE_BOX_INFO      "info"
#define GNOME_MESSAGE_BOX_WARNING   "warning"
#define GNOME_MESSAGE_BOX_ERROR     "error"
#define GNOME_MESSAGE_BOX_QUESTION  "question"
#define GNOME_MESSAGE_BOX_GENERIC   "generic"
```

Para crear un diálogo de mensaje, use la función *gnome_message_box_new* con el texto del mensaje como primer argumento, el segundo es el tipo de mensaje (uno de los definidos antes), y después cualquier número de botones terminado por un NULL exactamente como en el caso de *GnomeDialog*. Una vez creado se usa como *GnomeDialog*.

Diálogos de propiedades

Si tiene que configurar alguna propiedad en su aplicación, tendría que usar el diálogo *GnomePropertyBox* para realizar esta configuración y así hacer las aplicaciones más consistentes. Nuevamente este objeto se deriva de *GnomeDialog*, por eso su uso es similar. La diferencia es que *GnomePropertyBox* define algunas nuevas señales, estas son *apply* (aplicar) y *help* (ayuda). Ambas pasan el número de página como segundo argumento, lo que es útil para mostrar la ayuda apropiada en caso de que se solicite. No es así para *apply*, ya que este sistema se creó para tener un botón de aplicar por cada página y todavía no está acabado, por lo que debería ignorar cualquier señal *apply* con un número de página distinto de -1, que es el número de aplicación *global* a todas las páginas. Puede conseguir un apply por cada página introduciendo su propio código, pero no es seguro que este código se llegue a completar. Debería ser más seguro aplicar los cambios globalmente, ya que es lo que está implementado en *gnome-libs 1.0*.

Para usar diálogos de propiedades, llame a *gnome_property_box_new*, esto creará un diálogo nuevo con un cuaderno de notas y los siguientes cuatro botones:

OK, que llamará a la función que se encarga de atender la señal 'apply' pasándole todas las páginas una a una y terminando en la página -1, momento en el que cerrará el diálogo.

Aplicar, que efectua la misma operación que el anterior con la salvedad de que no cierra el diálogo.

Cerrar, sólo cerrará el diálogo, y *Ayuda* que llamará a la función de tratamiento de ayuda si usted ligó alguna.

Después de crear el diálogo debe conectar la señal *apply* al manejador correspondiente que usted implemente, y recomendamos que conecte también la señal *destroy* a la ventana apropiada para destruir los datos asociados con el diálogo de propiedades cuando dicha ventana se cierre.

Ahora ya puede crear las diferentes páginas de su diálogo de propiedades, añadiéndolas con *gnome_property_box_append_page*, este método toma su página como segundo argumento y una etiqueta como tercero (normalmente será simplemente un *GtkLabel*). Tiene también que conectar las diferentes señales a los widgets de sus páginas para que al editar alguno de ellos, el diálogo de propiedades aparezca como cambiado (de otra forma los botones Aplicar y OK no serán sensibles). Esto se hace llamando a *gnome_property_box_changed* cada vez que el usuario cambia algo en los widgets. Por ejemplo en el widget de entrada (y derivados) se conecta la señal *changed*. Siga este ejemplo:

```
/*rutina de tratamiento de aplicar*/
static void
property_apply(GnomePropertyBox *caja, int pag_num, gpointer data)
{
        /*ignora los números de página distintos de -1*/
        if(pag_num!=-1)
                return;
        /*hacer la rutina de aplicar aquí*/
        ...
}
...
/*en algún lugar del archivo fuente*/
GtkWidget *pcaja;
GtkWidget *widget;
...
pcaja = gnome_property_box_new();
gtk_signal_connect(GTK_OBJECT(pcaja),"apply",
        GTK_SIGNAL_FUNC(property_apply),NULL);
...
/*se crea una página para la caja de propiedades y se añade
  al contenedor llamado widget*/
gnome_property_box_append_page(GNOME_PROPERTY_BOX(pcaja),
          widget, gtk_label_new("UnaPágina"));
/*ahora se añaden otras páginas de una manera similar*/
...
```

```
/*mostramos la caja de diálogo*/
gtk_widget_show_all(pcaja);
```

Diálogo de Selección de Archivos

GNOME no tiene su propio diálogo de selección de archivos, aunque está planeado crearlo en un futuro, por ahora tiene que usar el típico diálogo de *GTK+*.

El uso del diálogo de selección de archivos es muy simple. Cree el diálogo con *gtk_file_selection_new*, pasando el título del diálogo como argumento. Después de esto conecte la señal *clicked* a los botones *OK* y *Cancelar*. Por ejemplo para un diálogo de "abrir archivo", puede probar que el archivo es del tipo correcto cuando el usuario presiona *OK* y si todo va bien cierre el diálogo (normalmente con *gtk_widget_destroy*), o bien para el diálogo de guardar archivo, puede preguntar si el archivo existe. Normalmente es más seguro y sencillo hacer el diálogo de selección de archivos no modal. Asi se asegura de que destruirá el diálogo de selección de archivo cuando el objeto o ventana vaya a trabajar con él. Aquí está la función que invoca al diálogo de "guardar como" para *Achtung*, un programa de presentaciones en el que estamos trabajando.

```
void
presentation_save_as (AchtungPresentation *p)
{
        GtkFileSelection *fsel;

        g_return_if_fail (p != NULL);
        g_return_if_fail (p->doc != NULL);

        fsel = (GtkFileSelection *)
        gtk_file_selection_new (_("Guardar presentación como..."));
        if (p->real_file && p->filename)
                gtk_file_selection_set_filename (fsel, p->filename);

        gtk_object_set_data(GTK_OBJECT(fsel),"p",p);

        /* Conectamos las señales con OK y Cancelar */
        gtk_signal_connect (GTK_OBJECT (fsel->ok_button), "clicked",
                            GTK_SIGNAL_FUNC (save_ok), fsel);
        gtk_signal_connect_object (GTK_OBJECT (fsel->cancel_button),
                                   "clicked",
                                   GTK_SIGNAL_FUNC (gtk_widget_destroy),
                                   GTK_OBJECT(fsel));

        gtk_window_position (GTK_WINDOW (fsel), GTK_WIN_POS_MOUSE);

        /*si la presentación muere también lo harán los diálogos*/
        gtk_signal_connect_object_while_alive(GTK_OBJECT (p), "destroy",
                                   GTK_SIGNAL_FUNC (gtk_widget_destroy),
                                   GTK_OBJECT(fsel));

        gtk_widget_show (GTK_WIDGET (fsel));
}
```

Actualmente, esta función es un método "Guardar como" de la clase *AchtungPresentation*, utilizando lenguaje de orientación a objetos. *AchtungPresentation* es un GTKObject que usamos para guardar todos los datos de la presentación (este es un ejemplo bonito de cómo usar GtkObject para cosas no relacionadas directamente con los widgets o la programación de GUI's). Primero se revisan los argumentos de la función con *g_return_if_fail*, esto se hace sólo por motivos de depuración. Después creamos un nuevo *GtkFileSelection* con el título "Guardar presentación como...". Ignore la macro *_()* alrededor de las cadenas por el momento, se usa para la internacionalización. Después de todo esto se verifica si la presentación ya tiene un nombre de archivo asociado, y si es así, se pone este nombre en el diálogo de selección de archivo. Entonces conectamos el botón *OK* con la función *save_ok* definida en algún lugar del archivo y pasamos el diálogo como argumento. Entonces usamos *connect_object* para unir el botón *Cancel* con la destrucción del diálogo de selección. El método *connect_object* es similar a *connect* pero cuando llama a la función pasará el objeto del campo de datos como primer argumento de la función. Por eso al conectarlo con *gtk_widget_destroy* destruirá el objeto pasado en el campo de datos, que es el diálogo de selección de archivos. Entonces situamos el diálogo cerca del ratón. En un futuro, cuando este diálogo se derive de *GnomeDialog*, no necesitará hacerlo, porque se hará de acuerdo con el resto de los diálogos de GNOME. Todavía usamos otro método conectado a una señal... Esta vez *gtk_signal_connect_object_while_alive*, el cual es similar a *connect_object*, pero tiene un preciosa peculiaridad. La señal se desconectará cuando el objeto pasado en el campo de datos muera. Necesitamos que ocurra porque el diálogo de selección de archivos con toda probabilidad será destruido antes de que la presentación misma sea destruida, podría intentar destruir un diálogo de selección de archivos que en realidad no existe y con probabilidad causará una violación de segmento, colgándose. Este camino es seguro y si el diálogo de selección de archivo está cuando la presentación es destruida, es destruida con él.

Diálogo Acerca de...

Probablemente quiera tener una opción "Acerca de..." en el menú "Ayuda" de su aplicación, y debería mostrarse en un diálogo estándar. Hay un diálogo en GNOME para este tipo de propósitos, la clase GnomeAbout. Se crea con *gnome_about_new*, que tiene el siguiente prototipo

```
GtkWidget* gnome_about_new(const gchar *title, /* Nombre de la apli-
cación. */
                           const gchar *version, /* Versión. */
                           const gchar *copyright, /* Copy-
right(una línea.) */
                           const gchar **au-
thors, /* Lista con los autores
                           terminada con NULL. */
                           const gchar *comments, /* Otros comentar-
ios. */
                           const gchar *logo /* Un archivo pixmap
                           con el logo. */
                           );
```

Después de crear la caja de diálogo, debería establecer la ventana de la aplicación como su padre con *gnome_dialog_set_parent*. Y entonces puede mostrar el diálogo. Lo siguiente implementa un diálogo "acerca de", asumimos que VERSION ha sido definida con #define para ser una cadena con el número de la versión. También pasamos como logotipo un NULL, indicando que no tenemos una imagen que mostrar.

```
GtkWidget* dlg;
char *authors[] = {
        "George Lebl",
        NULL;
```

```
};

dlg = gnome_about_new("Alguna aplicación", /* Nombre de la apli-
cación. */
                      VERSION, /* Versión. */
                      "(c) 1999 George Lebl", /* Copy-
right (una línea.) */
                      authors, /* Lista con los autores, termi-
nado con NULL*/
                      "bla bla bla", /* Otros comentarios. */
                      NULL /* Un archivo pixmap con el logo. */
                      );

gnome_dialog_set_parent(GNOME_DIALOG(dlg), GTK_WINDOW(app));

gtk_widget_show(dlg);
```

Entradas

Algunas veces, especialmente en los diálogos de propiedades, querrá añadir campos para introducir texto, archivos, pixmaps, iconos o reales de doble precisión (N.T: double). Esto es lo que hace los widgets *gnome-*entry*.

GnomeEntry

Es una entrada para texto regular, pero incluye historia de los valores introducidos anteriormente. Note que este widget no está derivado de *GtkEntry*, sino que contiene a un objeto de esta clase. Todo esto significa que no puede usar los métodos de *GtkEntry* sobre este objeto directamente, tiene que obtener un puntero al objeto *GtkEntry* que hay dentro de *GnomeEntry*. Cuando llama a *gnome_entry_new*, se le pasa una cadena *history_id*, es un identificador único para identificar a esta entrada o a este tipo de entradas en su aplicación. Todas las entradas que comparten *history_id* tendrán una historia común. Después de que cree un *GnomeEntry* tendrá que usar la función *gnome_entry_gtk_entry* para obtener el puntero al objeto *GtkEntry* y asociarle alguna señal o manipular el texto. Aquí hay un ejemplo:

```
GtkWidget *gnomeentry;
GtkWidget *gtkentry;
...
gnomeentry = gnome_entry_new("texto1");

/* tomamos GtkEntry para enlazarlo con la señal "changed" y conocer
   cuándo el usuario ha cambiado la entrada */
gtkentry = gnome_entry_gtk_entry(GNOME_ENTRY(gnomeentry));
gtk_signal_connect(GTK_OBJECT(gtkentry),"changed",
                   GTK_SIGNAL_FUNC(entry_changed), NULL);
```

GnomeFileEntry

GnomeEntry es la base para *GnomeFileEntry*. De nuevo no está derivado, sino que una instancia de *GnomeEntry*, está contenida en *GnomeFileEntry*. Este tipo de jerarquía está presente en todos los widget de entrada de GNOME. *GnomeFileEntry* añade un botón de búsqueda en la parte derecha de la entrada, acepta también archivos que provengan del gestor de archivos por medio del mecanismo de "arrastrar y soltar", por ejemplo. Su uso es sumamente similar a *GnomeEntry*. Se crea la entrada con *gnome_file_entry_new*. El primer argumento es el *history_id* del *GnomeEntry*, y el segundo argumento es el título de la caja de diálogo del buscador. Para conseguir el *GtkEntry*, de nuevo use el método gtk_entry, llamado *gnome_file_entry_gtk_entry*. Para obtener finalmente el nombre del archivo, puede leer el texto de la misma forma que *GtkEntry*, o podría usar otros métodos, *gnome_file_entry_get_full_path* , el cual toma una bandera *file_must_exist* como segundo argumento. Si éste está activado (TRUE), la función devuelve NULL cuando no existe el archivo. Si es FALSE o el archivo existe, la función devuelve el camino entero hasta el archivo.

GnomePixmapEntry

Esto es una entrada para introducir imágenes cualquier tipo. Vuelve a incluir (no se deriva de) *GnomeFileEntry*, por eso puede hacer todo lo que *GnomeFileEntry* puede hacer (incluso aceptar los archivos arrastrados desde el gestor de archivos). De todas formas esta entrada añade un cuadro de previsualización para la imagen que se encuentra en la entrada. También su diálogo de selección de archivo incluye un cuadro de previsualización en el lado derecho del listado de archivos.

Su uso es muy similar a las entradas anteriores. Se llama a *gnome_pixmap_entry_new* con los mismos argumentos que *GnomeFileEntry*, con una bandera añadida, *do_preview*. Esta bandera especifica si la caja de previsualización es visible o no. Pero tenga cuidado, esto no ahorra memoria, sólo ahorra espacio. Use de nuevo *gnome_pixmap_entry_gtk_entry* para obtener el widget *GtkEntry*. Para leer el nombre del archivo de la imagen, si pudo ser cargada como una imagen en el previsualizado (usando Imlib), puede usar *gnome_pixmap_entry_get_filename*, que devuelve NULL si el archivo de la imagen no existe o no pudo ser cargada, y el nombre completo del archivo en el caso contrario.

GnomeIconEntry

La entrada de iconos es muy similar a *GnomePixmapEntry*, pero está pensado para imágenes con el tamaño estándar de los iconos de 48x48. También al margen de la caja de previsualización hay un botón con la imagen escalada a 48x48. Si presiona el botón, obtendrá una lista de imágenes del mismo directorio en el que está el icono actual. Para crear una entrada de icono use *gnome_icon_entry_new* con *history_id* y *browse_dialog_title* como argumentos tipo cadena. Una vez que tiene un icono existente con una una imagen real, use *gnome_icon_entry_get_filename* que funciona como *gnome_pixmap_entry_get_filename*. También puede conseguir *GtkEntry* usando *gnome_icon_entry_gtk_entry*. Ejemplo:

```
GtkWidget *iconentry;
GtkWidget *gtkentry;
char *algunarchivo;
...

iconentry = gnome_icon_entry_new("icono","Explorar...");

/* queremos entablecer "somefile" como icono por defecto */
gnome_icon_entry_set_icon(GNOME_ICON_ENTRY(iconentry), algunarchivo);

/* tomamos el GtkEntry para enlazar con la señal "changed" */
gtkentry = gnome_icon_entry_gtk_entry(GNOME_ICON_ENTRY(iconentry));
gtk_signal_connect(GTK_OBJECT(gtkentry),"changed",
                   GTK_SIGNAL_FUNC(entry_changed), NULL);
```

```
...
/* aquí queremos tomar el icono seleccionado */
char *icono;
icono = gnome_icon_entry_get_filename(GNOME_ICON_ENTRY(iconentry));
...
/* nos aseguramos de liberar el icono después de usarlo */
g_free(icon);
```

GnomeNumberEntry

GnomeNumberEntry es un widget para la introducción de números de doble precisión con una calculadora. La mayor parte del tiempo para la entrada de números querrá usar el widget *GtkSpinButton* , de todas formas para aplicaciones como el cálculo de hipotecas, o programas de finanzas, donde las calculadoras son necesarias, querrá usar este tipo de entrada. Básicamente es un widget *GnomeEntry* con un botón a la derecha que llama a un diálogo con una calculadora. El usuario puede usar la calculadora y presionar OK, entonces el número de la entrada es actualizado con lo que tenga la calculadora. Para crear uno de estos widget, use *gnome_number_entry_new*, pasando como primer argumento el *history_id* y el título del diálogo de la calculadora como segundo argumento. Para obtener el widget *GtkEntry* sólo use *gnome_number_entry_gtk_entry*. Para leer el número como un valor tipo *double* (real de doble precisión), use el método *gnome_number_entry_get_number*.

Utilizando Imágenes

Cuando necesite utilizar imágenes en sus aplicaciones, lo más probable es que quiera emplear *GnomePixmap*. Este widget permite usar imágenes de una manera más fácil y sin tener que aprender Imlib, la biblioteca de imágenes usada por este widget.

Hay varias funciones *new* (constructores) para *GnomePixmap*, dependiendo del origen del mapa de pixels. El más usado probablemente sea *gnome_pixmap_new_from_file* el cual toma un archivo (que es una imagen que puede cargar Imlib) y crea un widget. Hay también *gnome_pixmap_new_from_file_at_size* , idéntica a la anterior, con la salvedad de que puede fijar el tamaño al que será escalada la imagen. Si ya tiene cargada la imagen con Imlib (en el caso de que quiera hacer primero otras cosas con el mapa de pixels), puede utilizar *gnome_pixmap_new_from_imlib* y *gnome_pixmap_new_from_imlib_at_size*, la cual toma *GdkImlibImage* como el primer argumento. Si ya tiene el widget y quiere cambiar la imagen de dentro, puede usar *gnome_pixmap_load_** que tiene casi la misma sintaxis que las funciones "new" anteriores, a excepción de que recibe un *GnomePixmap* como primer argumento, y por supuesto cambia en su nomenclatura el _new_from_ por _load_from_.

Un ejemplo de su uso:

```
GtkWidget *pix;
...
/* Carga archivo1.png y lo escala a 48x48 */
pix = gnome_pixmap_new_from_file_at_size("archivo1.png",48,48);
/* Ahora podemos empaquetar pix en otro widget */
...
/*Ahora cambiaremos el pixmap que con-
tiene pix por el que se encuentra
```

```
  en archivo2.png. No lo escalaremos */
gnome_pixmap_load_file(GNOME_PIXMAP(pix),"archivo2.png");
```

Manejo de Sesiones

Sus aplicaciones pueden recordar el estado en que el usuario la abandonó la última vez y reestablecerlo tal cual cuando éste las reinicie en otro momento, incluso pueden permitir que el usuario almacene distintos de estos estados (A partir de aquí sesiones). Por ejemplo el usuario podría tener una sesión normal, pero a veces entrar en una sesión especial donde tenga las aplicaciones configuradas de diferente forma. *gnome-libs* abstrae los aspectos desagradables de esto. Usted, como programador, no debería preocuparse de las interioridades del control de sesiones a menos que sus aplicaciones tengan algún estado especialmente complicado de guardar. Para almacenar una sesión sólo necesitas el siguiente código (la mayor parte pertenece al programa de ejemplo gnome-hello-4-SM):

```
/* El manejador save_yourself, al implementarlo hemos ignorado
   tranquilamente la mayoría de sus complicados parámetros, salvará
   nuestra sesión y devolverá TRUE */

static int
save_yourself(GnomeClient *client, int phase,
              GnomeSaveStyle save_style, int shutdown,
              GnomeInteractStyle interact_style, int fast,
              gpointer client_data)
{

        /* Obtenemos el identificador que acompañará al nombre de
           nuestra configuración */

        char *prefix= gnome_client_get_config_prefix (client);

        /* Construimos una orden que nos servirá para borrar una
           sesión salvada anteriormente */

        char *argv[]= { "rm", "-r", NULL };

        /* Salvamos el actual estado */

        gnome_config_push_prefix (prefix);

        gnome_config_set_int("Seccion/Clave",un_valor);
        ...
        gnome_config_pop_prefix ();
        gnome_config_sync();

        /* Aquí comienza el verdadero uso del controlador de
           sesiones. Vamos a reini-
ciar/borrar la sesión que salvamos:
           para ellos usaremos la orden que montamos anterior-
mente */

        argv[2] = gnome_config_get_real_path (prefix);
        gnome_client_set_discard_command (client, 3, argv);

        /* Ahora ajustaremos las órdenes "clone" y "restart" de esta
           aplicación. Note que hemos us-
ado el mismo valor para ambas
```

```
           ("rm -f ..."). El controlador de sesiones automáticamente
           completará el resto de las opciones requeridas para fi-
jar el
           identificador de sesión al comienzo. El parámetro
           "client_data" fue fijado al arrancar la aplicación, en el
           momento en que "save_yourself" fue conectado. */

        argv[0]= (gchar*) client_data;
        gnome_client_set_clone_command (client, 1, argv);
        gnome_client_set_restart_command (client, 1, argv);

        return TRUE;
}

static void
die (GnomeClient *client, gpointer client_data)
{
        /* Una salida civilizada de la aplicación. No salvaremos
           ningún estado aquí, porque el contro-
lador de sesión debería
           haber enviado un mensaje a save_yourself antes.  */
        gtk_exit (0);
}

...
GnomeClient *client;
...

/* Aquí iría todo lo que haga su función main DESPUÉS de la lla-
mada a
   gnome_init */

/* Obtenemos el cliente "maestro", éste se conectó al controlador de
   sesiones durante la llamada 'gnome_init'. Toda las comunica-
ciones con
   el controlador de sesión, serán real-
izadas desde este cliente maestro */
client = gnome_master_client ();

/* Enganchamos nuestros manejadores a las señales correspondi-
entes del
   cliente maestro.  */

gtk_signal_connect (GTK_OBJECT (client), "save_yourself",
                    GTK_SIGNAL_FUNC (save_yourself),
                    (gpointer) argv[0]);
gtk_signal_connect (GTK_OBJECT (client), "die",
                    GTK_SIGNAL_FUNC (die), NULL);

/* Comprobamos si estamos conectados al controlador de sesiones */
if (GNOME_CLIENT_CONNECTED (client)) {

        /* Estamos conectados: obtendremos ahora el prefijo bajo el
           que salvamos nuestra sesión la última vez y cargaremos
           nuestros datos */

        gnome_config_push_prefix
                (gnome_client_get_config_prefix (client));

        some_value = gnome_config_get_int("Section/Key=0");

        gnome_config_pop_prefix ();
} else {
```

```
        /* No estamos conectados al controlador de
           sesiones. Continuamos la ejecución de la forma habit-
ual */
        ...
}
```

Este es un ejemplo de control de sesiones muy sencillo el cual puede ser suficiente para la mayoría de los programas, para más información puede consultar la documentación del desarrollador de GNOME.

Interfaz para Varios Documentos

La Ventana Principal MDI

Si su aplicación maneja documentos, lo más probable es que quiera que maneje varios documentos a la vez. GNOME proporciona un modelo MDI que es particularizable por el usuario y sencillo de usar. Se puede usar tres modelos del documento en pantalla. Uno al estilo de un cuaderno de notas es el más útil: los documentos pueden ser almacenados en cuadernos de notas, y si se desea pueden ser separados en diferentes ventanas. Otro es el estilo de "nivel superior" donde cada documento se separa en una ventana de nivel superior. O para terminar un estilo modal donde solo hay una ventana y los documentos pueden seleccionarse a través de un menú. (Nótese que el código que aquí se muestra pertenece a *gnome-hello-7-mdi* ejemplo de aplicación en *gnome-libs*, ligeramente modificado. Este ejemplo no es tan largo en gnome-libs porque aquí se reproducen sólo las partes del ejemplo más importantes)

Para usar las características de MDI. Tiene que sustituir la llamada *gnome_app_new* por *gnome_mdi_new* con los mismos argumentos de *gnome_app_new*. Para añadir menús y barras de herramientas, puede utilizar *gnome_mdi_set_menubar_template* y *gnome_mdi_set_toolbar_template* con la GnomeUIInfo como argumento. Para MDI, estos no son los actuales menús, añadiremos nuestras propias entradas a los menús de cada hijo. Después de esto se puede establecer dónde tendrán lugar las inclusiones del menú. Puedes llamar a *gnome_mdi_set_child_menu_path* para el nombre principal del menú del que colgarán después los propios hijos del menú introducido. Esto es en la mayoría de los casos el menú "Archivo" o "Documentos" . Entonces puede especificar la ruta (nombre del menú) para acceder al menú en el que quiera insertar la lista de los hijos, puede hacer esto haciendo la llamada *gnome_mdi_set_child_list_path* con el nombre del menú y añadir un '/' al final para especificar que quiere introducir esas entradas en el menú. Por ejemplo:

```
GtkWidget *mdi;
...
mdi = gnome_mdi_new("gnome-hello-7-mdi", "GNOME MDI Hello");
...

/* main_menu y toolbar_info son las descripciones del menú y de la
barra de herramientas, respectivamente */

gnome_mdi_set_menubar_template(mdi, main_menu);
gnome_mdi_set_toolbar_template(mdi, toolbar_info);

/* Insertamos algunos elementos en mdi (Sí desea más informa-
ción, vea
las funciones de inserción de menus de gnome-app-helper) */
```

```
gnome_mdi_set_child_menu_path(GNOME_MDI(mdi), "Archivo");
gnome_mdi_set_child_list_path(GNOME_MDI(mdi), "Hijo/");
```

En su estructura GnomeUIInfo tendrá definido un menú llamado "Archivo" y y otro llamado "Hijo". El menú "Hijo" es solamente un menú vacío, ya que no le hemos introducido ninguna entrada.

Ahora usted debe abrir la ventana principal con *gnome_mdi_open_toplevel*. Esto abrirá una ventana de nivel superior sin ningún hijo.

Si desea manejar sesiones que impliquen MDI, debe definir una función que cree un hijo (Un widget GnomeMDIChild) a partir de una cadena de caracteres (por ejemplo, la ruta completa de un archivo podría servir para que un procesador de textos recuperase la sesión). Después debe llamar a *gnome_mdi_restore_state*: este método toma como argumentos una ruta de configuración y un puntero a la función que usted implementó antes. Por ejemplo, usando el manejador de sesión mostrado anteriormente, se podría utilizar:

```
gnome_config_push_prefix (gnome_client_get_config_prefix (client));
restart_ok = gnome_mdi_restore_state(GNOME_MDI(mdi), "MDI Session",
                                     my_child_new_from_config);
gnome_config_pop_prefix ();
```

La variable restart_ok es booleana, le informa si se han cargado todos los datos correctamente.

También puede unir la señal *destroy* del objeto MDI para hacer *gtk_main_quit* cuando el MDI sea destruido.

El Hijo MDI

Para aplicaciones complejas, todos los hijos podrían ser derivados de la clase virtual *GnomeMDIChild*. Para aplicaciones sencillas no necesitará derivar nuevas clases, puede utilizar *GnomeMDIGenericChild*, y utilizar el hecho de que puede almacenar datos arbitrarios en un *GtkObject* para guardar sus propios datos en el objeto.

Para utilizar un objeto 'hijo' genérico *GnomeMDIChild*, debe crearlo con *gnome_mdi_generic_child_new*, esta función precisa de un nombre que será asignado a la nueva instancia. Además necesita configurarlo antes de poder utilizarlo.

Primero debe añadir una función que cree nuevas vistas del objeto. Una vista es sólo una nueva ventana que muestra el contenido del mismo archivo o los mismos datos que mostraba la ventana original. Esta función aceptará como parámetros un GnomeMDIChild (El objeto del que vamos a crear una nueva vista) y un puntero de datos genérico, y devolverá una instancia de *GnomeMDIGenericChild* que representa la nueva vista. Para añadir esta función haga una llamada a *gnome_mdi_generic_child_set_view_creator*, pasándole un puntero a la función y el puntero de datos genérico que se le pasarará cuando sea llamada.

El siguiente paso es ajustar los menús del objeto con *gnome_mdi_child_set_menu_template*, que recibe un vector de punteros *GnomeUIInfo* con las definiciones del menú que poseera el objeto.

Después debe llamar a *gnome_mdi_generic_child_set_config_func* para establecer una función que devuelva una cadena de caracteres (Cuya memoría debe ser asignada dinámicamente) que será guardada en el archivo de configuración. Esta cadena de caracteres se utilizará para cargar el objeto mediante una llamada a *gnome_mdi_restore_state* la siguiente vez que el usuario utilice la aplicación. La cadena debería ser evidentemente un nombre de archivo de un documento, o alguna cadena de caracteres desde la cual pueda recrear completamente la ventana/el documento.

Por último necesita establecer una función que permita ajustar la etiqueta del MDI. Esto lo hará llamando a *gnome_mdi_generic_child_set_label_func* con un puntero a dicha función. Está función debe tomar como argumentos un *GnomeMDIGenericChild* (El objeto), un puntero a la antigua etiqueta, el cual será nulo si todavía no hubierá ninguna, y un puntero de datos genérico. La etiqueta puede ser cualquier widget, por ejemplo en *gnome-hello-7-mdi* se utiliza una caja horizontal dentro de la cual se añaden un mapa de pixels y una etiqueta GTK+. Esta función o bien puede crear una etiqueta nueva y destruir la antigua, o bien establecerá la etiqueta si la etiqueta no existía.

Después de esto usted ya puede añadir el 'hijo' al MDI, si por ejemplo está cargando un nuevo archivo, utilice *gnome_mdi_add_child* y *gnome_mdi_add_view*, para añadir un nuevo hijo y una nueva vista al MDI. Si está creando un nuevo hijo desde la función *gnome_mdi_restore_state*, ésta debería devolver el hijo, el MDI lo incluirá y añadirá las vistas adecuadas. Probablemente también querrá almacenar sus datos en el widget 'hijo' en este momento.

Esto es un pequeño ejemplo de la creación de un nuevo hijo.

```
GnomeMDI *mdi;
...
GnomeMDIGenericChild *child;
...
/* Crea un nuevo hijo con el nombre 'nombre'*/
if((child = gnome_mdi_generic_child_new("nombre") != NULL) {
        /* Creamos una vista */
        gnome_mdi_generic_child_set_view_creator(child,
                                          my_child_create_view, NULL);
        /* Ajustamos una plantilla de menú para el menú de child */
        gnome_mdi_child_set_menu_template(GNOME_MDI_CHILD(child),
                                          main_child_menu);
        /* Ajustamos una función que permitirá traba-
jar con la cadena
           de configuración */
        gnome_mdi_generic_child_set_config_func(child,
                                      my_child_get_config_string, NULL);
        /* Ajustamos function que cambia o crea una etiqueta */
        gnome_mdi_generic_child_set_label_func(child, my_child_set_label,
                                               NULL);

        /* Añadimos el hijo al MDI */
        gnome_mdi_add_child(mdi, GNOME_MDI_CHILD(child));

        /* Y añadimos una nueva vista del hijo */
        gnome_mdi_add_view(mdi, GNOME_MDI_CHILD(child));
}
```

Widgets Variados

Enlaces Web con GnomeHRef

A veces querrá poner un botón en sus aplicaciones que arranque un navegador para el usuario o apuntar un navegador ya arrancado a algún sitio. Todo lo que necesita es llamar a gnome_href_new con una URL como el primer argumento y la etiqueta como el segundo. Por ejemplo:

```
GtkWidget *widget;
...
widget = gnome_href_new("http://www.gnome.org", "La página de GNOME");
```

Seleccionando Iconos con GnomeIconSelection

Normalmente querrá añadir iconos utilizando el widget GnomeIconEntry, pero a veces puede querer poner el icono listándolo en la ventana directamente dentro de su aplicación. El widget que se mostrará a continuación solamente es útil si tiene algún directorio desde el cual coger los iconos. Cree el widget con *gnome_icon_selection_new*. Entonces cuando quiera añadir un directorio de iconos, utilice el método *gnome_icon_selection_add_directory* pasándole como argumento el directorio que desea añadir. Puede añadir varios directorios. Una vez termine este paso, llame a *gnome_icon_selection_show_icons*: esto cargará y mostrará los iconos. Para seleccionar un icono específico, utilice *gnome_icon_selection_select_icon* con el nombre del archivo del icono elegido. El nombre del archivo debería ser sólo el nombre base del icono y no la ruta entera. Una vez haya seleccionado el icono, puede obtener su ruta con el método *gnome_icon_selection_get_icon* el cual coge un argumento booleano 'full_path' y devuelve un puntero a una cadena de caracteres con el nombre del archivo o NULL si no había nada elegido. Si el argumento 'full_path' es TRUE, el valor devuelto será la ruta completa del icono. Señalar que el valor devuelto apunta a memoria interna por lo tanto no tiene que liberarla. Ejemplo:

```
GtkWidget *widget;
char *some_icon_directory;
char *some_other_icon_directory;
...
/* Creamos el widget */
widget = gnome_icon_selection_new();
gnome_icon_selection_add_directory(GNOME_ICON_SELECTION(widget),
                                   some_icon_directory);
gnome_icon_selection_add_directory(GNOME_ICON_SELECTION(widget),
                                   some_other_icon_directory);
gnome_icon_selection_show_icons(GNOME_ICON_SELECTION(widget));
...

/* Queremos obtener la selección (La ruta completa) */
char *filename;
...
filename = gnome_icon_selection_get_icon(GNOME_ICON_SELECTION(widget), TRUE);
```

El Widget GnomeCanvas

Aunque el widget *GnomeCanvas* está dentro de la biblioteca libnomeui, se le dedica un capítulo separado. El lienzo (N.T: canvas) es un widget para el dibujo de gráficos de muy alto nivel y de alto rendimiento, que resulta muy sencillo de utilizar. Incluye dos modos para trabajar con gráficos: Xlib y con suaviazado. El modo Xlib es más rápido (especialmente sobre redes) y con el modo con suavizado se obtienen unos resultados visuales mejores.

Creando un Widget Lienzo

Para crear un widget lienzo, usted debe llamar a *gnome_canvas_new*. Necesita asegurarse de que el lienzo se crea con un X visual (N.T: descripción del formato de los datos en la pantalla de un servidor X, incluyendo número de bits usados por cada color, la forma en que los bits son traducidos en valores RGB para su representación y la forma en que los mismos son almacenados en memoria) y un mapa de colores adecuado. Por ejemplo, si desea dibujar imágenes Imlib dentro del lienzo, debería hacer esto:

```
GtkWidget *canvas;
...
gtk_widget_push_visual(gdk_imlib_get_visual());
gtk_widget_push_colormap(gdk_imlib_get_colormap());
canvas = gnome_canvas_new();
gtk_widget_pop_visual();
gtk_widget_pop_colormap();
```

Después de esto debería llamar a *gnome_canvas_set_pixels_per_unit* para establecer la escala del lienzo. Puede entonces hacer *gtk_widget_set_usize* para establecer el tamaño del widget, y *gnome_canvas_set_scroll_region* para indicar la región en la cual se puede dibujar, está dado en (x1, y1, x2, y2). Básicamente son los límites externos de su dibujo. Una vez el lienzo está creado puede hacer:

```
GnomeCanvas *canvas;
...
/*ya está creado el lienzo, ahora ajustamos algunos parámetros*/
gnome_canvas_set_pixels_per_unit(canvas,10);
gnome_canvas_set_scroll_region(canvas,0.0,0.0,50.0,50.0);
```

Grupos y Elementos

En el lienzo hay elementos, que son los objetos que están actualmente sobre el lienzo, y grupos, los cuales son asociaciones de elementos. Un grupo está derivado de la clase base *GnomeCanvasItem* Esto es útil para aplicar funciones a todos los elementos dentro del grupo: acciones como mover o esconder un grupo entero. Hay también un grupo por defecto, el grupo raíz. Puede obtener este grupo llamando a *gnome_canvas_root*.

Creando Elementos

Crear elementos de un lienzo no es diferente de crear otros objetos. Se usa el mecanismo de argumentos del modelo de objetos estándar de GTK+. Básicamente se llama a *gnome_canvas_item*, con el grupo ráiz del lienzo padre como el primer argumento, el tipo de objeto como segundo argumento, y el resto de argumentos dados en pares (argumento, valor), terminado con un NULL. Esto se ilustra mejor con un ejemplo:

```
GnomeCanvas *canvas;
GnomeCanvasItem *item;
```

```
...
item = gnome_canvas_item_new(gnome_canvas_root(canvas),
                             GNOME_TYPE_CANVAS_RECT,
                             "x1", 1.0,
                             "y1", 1.0,
                             "x2", 23.0,
                             "y2", 20.0,
                             "fill_color", "black",
                             NULL);
```

Note que es extremadamente importante que el valor sea del tipo exacto, el compilador no lo convertirá implícitamente por usted. Si está haciendo algún cálculo y no está seguro de que obtiene el tipo correcto, haga una conversión explicita. Según creemos, todos los números (O la gran mayoría) que se emplean en el lienzo son reales de doble precisión.

Para encontrar los argumentos que cada elemento acepta, consulte la documentación de GNOME o mire en los archivos de cabecera *libgnomeui/gnome-canvas*.h*. Estos contienen una tabla como la que se muestra a continuación justo al principio del archivo (la cual ha sido tomado de *libgnomeui/gnome-canvas-rect-ellipse.h*).

Por ejemplo, aquí están los argumentos para el rectángulo (GNOME_TYPE_CANVAS_RECT) y la elipse (GNOME_TYPE_CANVAS_ELLIPSE):

Tabla 3-14. Argumentos para los Elementos Rectángulo y Elipse

Nombre	Tipo	Read/Write (lectura/escritura)	Descripción
x1	real de doble precisión	RW	Coordenada más a la izquiera del rectángulo o elipse
y1	real de doble precisión	RW	Coordenada más arriba del rectángulo o elipse
x2	real de doble precisión	RW	Coordenada más a la derecha del rectángulo o elipse
y2	real de doble precisión	RW	Coordenada más abajo del rectángulo o elipse
fill_color	cadena	W	Especificación de color X para rellenar con un color, o un puntero a NULL para ninguno (transparente)
fill_color_gdk	GdkColor*	RW	GdkColor ubicado para rellenar
outline_color	cadena	W	Especificación de color X para perfilar con un color, o un puntero a NULL para ninguno (transparente)

Nombre	Tipo	Read/Write (lectura/escritura)	Descripción
outline_color_gdk	GdkColor*	RW	GdkColor ubicado para bordear
fill_stipple	GdkBitmap*	RW	Modelo punteado para rellenar
outline_stipple	GdkBitmap*	RW	Modelo punteado para bordear
width_pixels	entero sin signo	RW	Anchura del perfil en pixels. El perfil no será escalado cuando se cambie el factor de escala del lienzo.
width_units	real de doble precisión	RW	Anchura del perfil en unidades de lienzo. El perfil será escalado cuando se cambie el factor de escala del lienzo.

Ahora supongamos que queremos cambiar algunas de estas propiedades. Esto se hace con una llamada a *gnome_canvas_item_set*. El primer argumento de esta función es un puntero a un objeto elemento del lienzo. Los argumentos siguientes son los mismos pares de argumentos vistos antes cuando creábamos un nuevo objeto lienzo. Por ejemplo, si queremos que el rectágulo que creamos antes cambie su color a rojo, haremos esto:

```
GnomeCanvas *canvas;
GnomeCanvasItem *item;
...
gnome_canvas_item_set(item, "fill_color", "red", NULL);
```

Hay métodos para operar sobre los elementos. Por ejemplo el método *gnome_canvas_item_move* tomará x e y como segundo y tercer argumento y moverá el elemento de forma relativa a su posición en x e y. Otros métodos son *gnome_canvas_item_hide* y *gnome_canvas_item_show*, los cuales esconden y muestran el elemento, respectivamente. Para controlar el orden de los elementos en z, puede usar los métodos *gnome_canvas_item_raise_to_top* y *gnome_canvas_item_lower_to_bottom* para subir o bajar el elmento hasta lo más alto o lo más al del fondo del orden en z de su grupo padre. Para tener un control más fino sobre z se puede usar *gnome_canvas_item_raise* y *gnome_canvas_item_lower*, estos métodos toman un argumento entero más que vale 1 o más, y especifica el número de niveles que el elemento tiene que moverse en z.

Lienzo con Suavizado

Para crear un lienzo que use suavizado para representar sus elementos, en vez de *gnome_canvas_new*, debe usar *gnome_canvas_new_aa*. Siga el siguiente ejemplo para crear un nuevo lienzo con suavizado:

```
GtkWidget *canvas;
...
```

```
gtk_widget_push_visual (gdk_rgb_get_visual ());
gtk_widget_push_colormap (gdk_rgb_get_cmap ());
canvas = gnome_canvas_new_aa ();
gtk_widget_pop_colormap ();
gtk_widget_pop_visual ();
```

Después de esto puede usar el lienzo exactamente de la misma forma que un lienzo normal.

Los elementos del lienzo con suavizado pueden, generalmente, hacer más cosas que los lienzos normales. Esto se debe a las limitaciones de Xlib como librería gráfica. Por ejemplo puede hacer cualquier tipo de transformación afín en cualquiera de sus miembros, cuando en un lienzo Xlib sólo se podrían hacer transformaciones afines en ciertos objetos.

Arrastrar y Soltar

A pesar de que la técnica de "arrastrar y soltar" (N.T: drag and drop) está implementada en GTK+, hemos pensado que sería mejor cubrir esto antes que algunas partes de GNOME sean discutidas.

Aceptando la Acción de Soltar

Ya ha visto un manipulador de la acción de soltar antes, cuando fueron discutidos los tipos MIME. Básicamente, para aceptar un objeto arrastrado, tiene que decidir qué tipo MIME del dato quiere recibir. Ha visto uno para "text/uri-list". Básicamente su manipulador sólo recibirá datos de aquellos tipos MIME que ha especificado, por eso sólo necesita saber cómo decodificar esos tipos concretos.

Para especificar el tipo MIME que desea recibir, cree un vector de estructuras *GtkTargetEntry*, donde el primer elemento es una cadena con la descripción del tipo MIME, el segundo es una bandera y el tercero es un entero con la información que usted desee añadir. Puede dejar la bandera a 0. El campo de información puede ser usado si está aceptando varias entradas distintas, como este entero será pasado a la función que se encarga de aceptar la acción de soltar, usted lo podrá usar como discriminador entre los los distintos tipos de datos (mediante la sentencia "switch", por ejemplo). Si sólo tiene un tipo, déjelo en 0.

Después de esto necesita preparar el widget para que pueda ser arrastrado. Esto se hace llamando a la función *gtk_drag_dest_set*. El primer argumento es el widget que quiere preparar, el segundo es una bandera para establecer que tipos de comportamientos por defecto se usaran al arrastrar, puede dejar esto como *GTK_DEST_DEFAULT_ALL*. El argumento siguiente es el número de elementos en el vector. El último argumento es el tipo de acción que acepta. Los tipos pueden ser de cualquiera de los siguientes: *GDK_ACTION_DEFAULT*, *GDK_ACTION_COPY*, *GDK_ACTION_MOVE*, *GDK_ACTION_LINK*, *GDK_ACTION_PRIVATE* y *GDK_ACTION_ASK*. Los más útiles son *GDK_ACTION_COPY* y *GDK_ACTION_MOVE*. Si usted está, por ejemplo, pasando cadenas u otros datos, debería usar sólo *GDK_ACTION_COPY*.

Luego necesita establecer un manipulador para recibir el objeto. Este manipulador debería tener el siguiente prototipo:

```
void
target_drag_data_received  (GtkWidget          *widget,
                            GdkDragContext     *context,
```

```
                              gint                x,
                              gint                y,
                              GtkSelectionData   *data,
                              guint               info,
                              guint               time);
```

El dato está en la estructura *GtkSelectionData*, dentro del campo *data*. Eso es todo lo que necesita para hacer un arrastrar y soltar normal. Aquí tiene un ejemplo:

```
static void
target_drag_data_received  (GtkWidget          *widget,
                            GdkDragContext     *context,
                            gint                x,
                            gint                y,
                            GtkSelectionData   *data,
                            guint               info,
                            guint               time)
{
        g_print("Obtuve: %s\n",data->data);
}
...
static GtkTargetEntry target_table[] = {
        { "text/plain", 0, 0 }
}
...
gtk_drag_dest_set (widget,
                   GTK_DEST_DEFAULT_ALL,
                   target_table, 1,
                   GDK_ACTION_COPY);
gtk_signal_connect (GTK_OBJECT (widget), "drag_data_received",
                    GTK_SIGNAL_FUNC (target_drag_data_received),
                    NULL);
```

Para más información sobre arrastrar y soltar, debería leer la documentación de *GTK+* en www.gtk.org[1].

Permitiendo la Acción de Arrastrar

Ahora vamos a ver la otra cara de DND (N.T acrónimo de "Drag aNd Drop", arrastrar y soltar). Prepare el vector *GtkTargetEntry*, de la misma forma que antes. Entonces, reemplace el argumento *bandera* por una máscara con los botones del ratón que se usarán para arrastrar. Puede ser *GDK_BUTTON1_MASK | GDK_BUTTON3_MASK* para el primer y tercer botón del ratón. Luego necesita unir la señal *drag_data_get* que mandará el dato, y *drag_data_delete* si la acción es *GDK_ACTION_MOVE*, para borrar el dato después de que el movimiento se complete con éxito. Aquí hay un ejemplo simple que funcionará con el código anterior:

```
static void
source_drag_data_get  (GtkWidget          *widget,
                       GdkDragContext     *context,
                       GtkSelectionData   *selection_data,
                       guint               info,
                       guint               time,
                       gpointer            data)
{
        char string[] = "Alguna Cadena!";
```

```
        gtk_selection_data_set (selection_data,
                                selection_data->target,
                                8, string, sizeof(string));
}
...
static GtkTargetEntry target_table[] = {
        { "text/plain", 0, 0 }
};
...
gtk_drag_source_set (widget,
                     GDK_BUTTON1_MASK|GDK_BUTTON3_MASK,
                     target_table, 1,
                     GDK_ACTION_COPY);
gtk_signal_connect (GTK_OBJECT (widget), "drag_data_get",
                    GTK_SIGNAL_FUNC (source_drag_data_get),
                    NULL);
```

La función *gtk_selection_data_set* copia el dato dentro de la selección de datos, que es usado para la transferencia.

Notas

1. http://www.gtk.org/

Capítulo 4. Construyendo Aplicaciones GNOME

Usando un Sencillo Makefile

Usar un sencillo Makefile es la forma mas rápida de compilar una pequeña aplicación GNOME. Si requiere un entorno de construcción mas sofisticado, debería usar una configuración con autoconf/automake, de la cual hablaremos brevemente mas tarde.

El Guión gnome-config

La linea de órdenes que necesita el compilador C para construir una aplicación GNOME puedes ser bastante larga y podría ser muy pesado escribirla a mano. Por ello las bibliotecas GNOME instalan un guión (N.T: script) para simplificar todo esto. Se llama *gnome-config* y toma dos opciones, *—cflags* y *—libs*. La opción —cflags dará al compilador las opciones necesarias para el paso de compilación, y —libs le dará las opciones precisas que necesita pasar al enlazador. También necesita aportar otro conjunto de argumentos a este guión de configuración de GNOME. Es necesario saber qué bibliotecas desea usar. Para nuestros propósitos esta es *gnome* y *gnomeui*. Así, por ejemplo para obtener los opciones del compilador para algún programa que use las bibliotecas estándar gnome y gnomeui, debería llamar a "gnome-config —cflags gnome gnomeui".

Un Ejemplo Sencillo de Makefile

Ahora para construir un Makefile sencillo, puede usar variables *CFLAGS* y *LDFLAGS* y las reglas implícitas que como mínimo soporta el make de GNU (otras probablemente lo harán bien, pero no estoy familiarizado con otros 'makes'). Así por ejemplo digamos que tiene una aplicación que tiene un main.c, main.h, extra.c y extra.h y el ejecutable es llamado gnome-foo. Ahora construyamos un sencillo Makefile para esta aplicación.

```
CFLAGS=-g -Wall `gnome-config --cflags gnome gnomeui`
LDFLAGS=`gnome-config --libs gnome gnomeui`

all: gnome-foo

gnome-foo: main.o extra.o
main.o: main.c main.h extra.h
extra.o: extra.c extra.h

clean:
        rm -f core *.o gnome-foo
```

Este es un ejemplo de Makefile extremadamente sencillo, pero debería servir para empezar.

Usando automake/autoconf

Usar automake/autoconf esta realmente fuera del ámbito de este documento, pero debería leer los manuales en linea en http://www.gnu.org/manual/manual.html[1], o leer las paginas info si las tiene instaladas con su navegador de ayuda GNOME (gnome-help-browser).

Hay ahora un ejemplo de aplicación la cual puede ayudar a iniciarle con autoconf/automake, la configuración de internacionalización, y otros aspectos de la construcción, también puede servir como un buen ejemplo de "hola mundo". Lo puede obtener en cualquier mirror ftp de GNOME (vaya a http://www.gnome.org/ftpmirrors.shtml[2] para una lista de mirrors) en el directorio *sources/GnomeHello/*.

Notas

1. http://www.gnu.org/manual/manual.html
2. http://www.gnome.org/ftpmirrors.shtml

Capítulo 5. Conclusión

Obteniendo Más Ayuda

Una de las mejores maneras de obtener ayuda con la programación en GNOME es probablemente leer primero la documentación disponible en www.gnome.org[1], o el sitio web del desarrollador en developer.gnome.org[2]. También se puede subscribir a gnome-devel-list@gnome.org[3], para suscribirse, mande un mensaje con *subscribe* en el asunto a gnome-devel-list-request@gnome.org[4]. Para reducir el trafico en la lista debería primero consultar la documentación antes de preguntar algo. También mire en www.gnome.org/mailing-lists/[5] para obtener una lista de todas las listas de correo, incluyendo la lista *GTK+*.

A pesar de todo nosotros consideramos más útiles los archivos de cabecera de las bibliotecas como fuente de información. Esto es sobre todo porque todavía no se encuentran todas las bibliotecas documentadas, sin embargo en los archivos de cabecera encontrará siempre todas las definiciones acordes con la versión actual, aunque el manual bien podría no estarlo. La mayoría de los nombres de funciones GTK+ y GNOME son muy descriptivos y es fácil imaginarse qué es lo que hacen. Nosotros usamos sólo los archivos de cabecera. A menudo es mucho más fácil mirar el prototipo de la función y figurarse qué hace, que consultar un manual de referencia. Esto le obliga a saber qué archivo de cabecera está buscando, lo cual no es tan difícil como parece dado que el nombre de los archivos de cabecera se corresponde con el nombre de los objetos o módulos que representan. Por ejemplo, el archivo de cabecera para *gnome-config* es *libgnome/gnome-config.h*. El archivo de cabecera para *GnomeCanvas* es *libgnomeui/gnome-canvas.h*.

Futuros Desarrollos de las Bibliotecas GNOME

Este tutorial cubre la programación con la versión 1.0 de las bibliotecas GNOME, pero por supuesto hay vida despues de la 1.0. Hay muchas cosas para las bibliotecas que todavía están sólo sobre el papel. Pero no se preocupe, en el futuro se intentará mantener toda la compatibilidad que sea humanamente posible con la versión 1.0.

Aquí tiene una pequeña lista de cosas que posiblemente estarán en posteriores versiones o en las que actualmente se está trabajando.

- Más diálogos comunes, tales como los diálogos de selección de archivos (Actualmente GNOME utiliza para esto el diálogo de selección de archivo estadar de GTK+).

- Más integración con CORBA en todos los aspectos del escritorio. Se incluirá mucho más soporte para CORBA en las bibliotecas que conforman el núcleo de GNOME.

- Mejoras en el widget GnomeCanvas, incluyendo mejoras en la manipulación del canal alfa e impresión directamente desde el widget.

- Reimplementación completa del sistema de configuración.

¡Y mucho mas! ... ¡Características que todavía ni siquiera hemos imaginado!

Notas

1. http://www.gnome.org/
2. http://developer.gnome.org/
3. mailto:gnome-devel-list@gnome.org
4. mailto:gnome-devel-list-request@gnome.org
5. http://www.gnome.org/mailing-lists/