Desarrollo multiplataforma en C.

Programar con NAPPGUI

Francisco García Collado

Desarrollo multiplataforma en lenguaje C

Como crear aplicaciones de alto rendimiento para sistemas Windows, macOS y Linux.

Versión 1.4.0.4516 [1]

frang@nappgui.com
www.nappgui.com

30 Septiembre 2023

[1] Este libro se ha editado en LaTeXgenerado automáticamente por **ndoc**.

Índice general

Parte 1

Guía del usuario

Capítulo 1

Inicio rápido

"...el número de instalaciones UNIX ha aumentado a 10, y se esperan más..."

Dennis Ritchie y Ken Thompson - Junio de 1972

NAppGUI es un SDK para desarrollar proyectos software en lenguaje C que funcionen en cualquier plataforma de escritorio (Windows, macOS o Linux) (Figura 1.1). Se puede utilizar C++ pero no es indispensable. Podemos desarrollar un programa completo utilizando únicamente ANSI-C.

1.1. Inicio rápido en Windows

Antes de empezar necesitas tener instaladas estas herramientas (Figura 1.2):

- Visual Studio[1] para compilar bajo Windows. Microsoft ofrece la versión gratuita *Community*.
- CMake[2]. Herramienta multiplataforma para crear los proyectos de compilación de forma automática, a partir del código fuente. Ten la precaución de seleccionar ***Add***

[1] `https://visualstudio.microsoft.com/vs/`
[2] `https://cmake.org/download/`

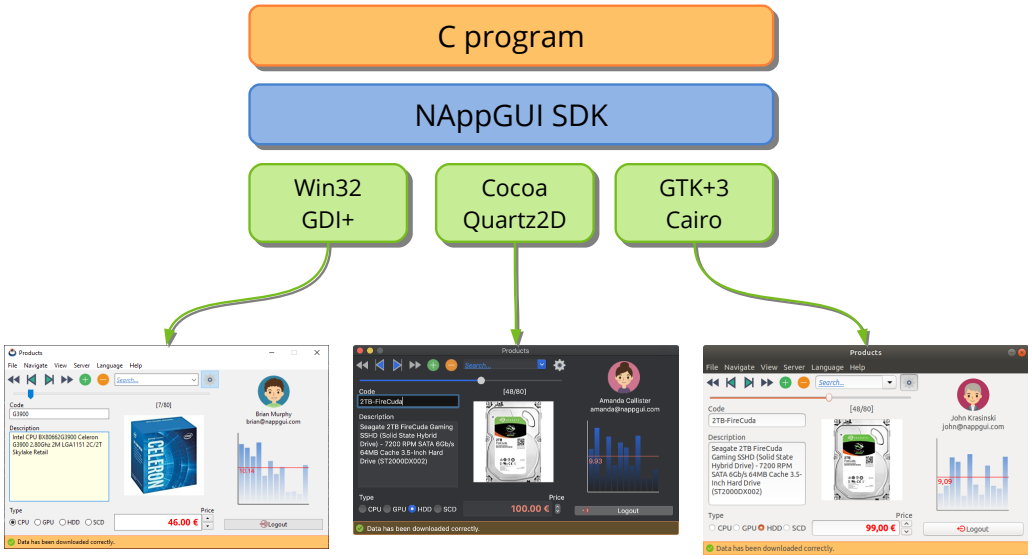


Figura 1.1: NAppGUI permite portar de manera sencilla aplicaciones escritas en ANSI C.

CMake to the system PATH for all users durante la instalación (Figura 1.3).

- Git[3]. Para descargar el proyecto desde GitHub.

Figura 1.2: Herramientas básicas en Windows.

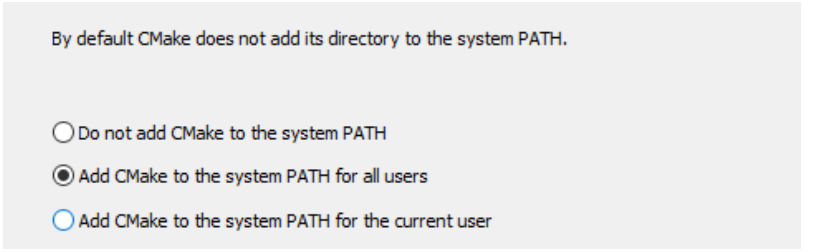


Figura 1.3: Acceso a CMake desde la línea de comandos.

Desde una consola en Windows:

```
git clone --depth 1 https://github.com/frang75/nappgui_src.git
cd nappgui_src
cmake -S ./src -B ./build
cmake --build ./build --config Debug
```

[3] https://git-scm.com/

Una vez compiladas, ya podrás ejecutar las aplicaciones de ejemplo existentes en el directorio `\build\Debug\bin` (Figura 1.4).

```
.\build\Debug\bin\Die.exe
.\build\Debug\bin\Bricks.exe
.\build\Debug\bin\Products.exe
.\build\Debug\bin\Col2dHello.exe
.\build\Debug\bin\GuiHello.exe
...
```

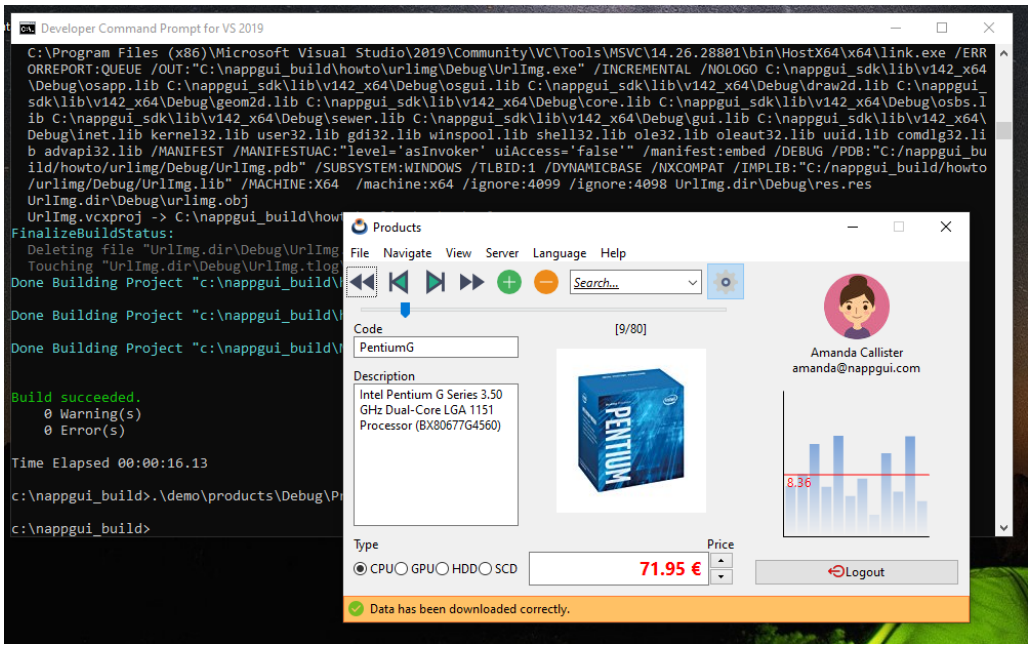


Figura 1.4: Ejecución del programa de ejemplo **Products** tras la compilación.

1.2. Inicio rápido en macOS

Antes de empezar, asegurate que tienes instalado y configurado Xcode[4], entorno imprescindible para el desarrollo bajo macOS. También necesitarás descargar e instalar CMake desde www.cmake.org[5] (Figura 1.5).

> *For defecto, CMake no configura el acceso por línea de comandos en macOS. Puedes crear enlaces simbólicos con* ***sudo "/Applications/CMake.app/Contents/bin/cmake-gui" –install****.*

[4] `https://developer.apple.com/xcode/`
[5] `https://www.cmake.org`

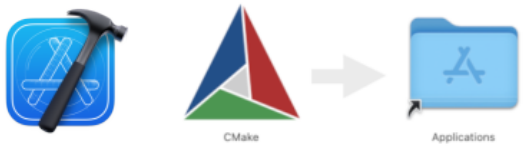


Figura 1.5: Xcode y CMake en macOS.

Abre un terminal en macOS:

```
git clone --depth 1 https://github.com/frang75/nappgui_src.git
cd nappgui_src
cmake -G Xcode -S ./src -B ./build
cmake --build ./build --config Debug
```

Una vez compiladas, ya podrás ejecutar las aplicaciones de ejemplo existentes en el directorio `/build/Debug/bin` (Figura 1.6).

```
./build/Debug/bin/Die.app/Contents/MacOS/Die
./build/Debug/bin/Bricks.app/Contents/MacOS/Bricks
./build/Debug/bin/Products.app/Contents/MacOS/Products
./build/Debug/bin/Col2dHello.app/Contents/MacOS/Col2dHello
./build/Debug/bin/GuiHello.app/Contents/MacOS/GuiHello
...
```

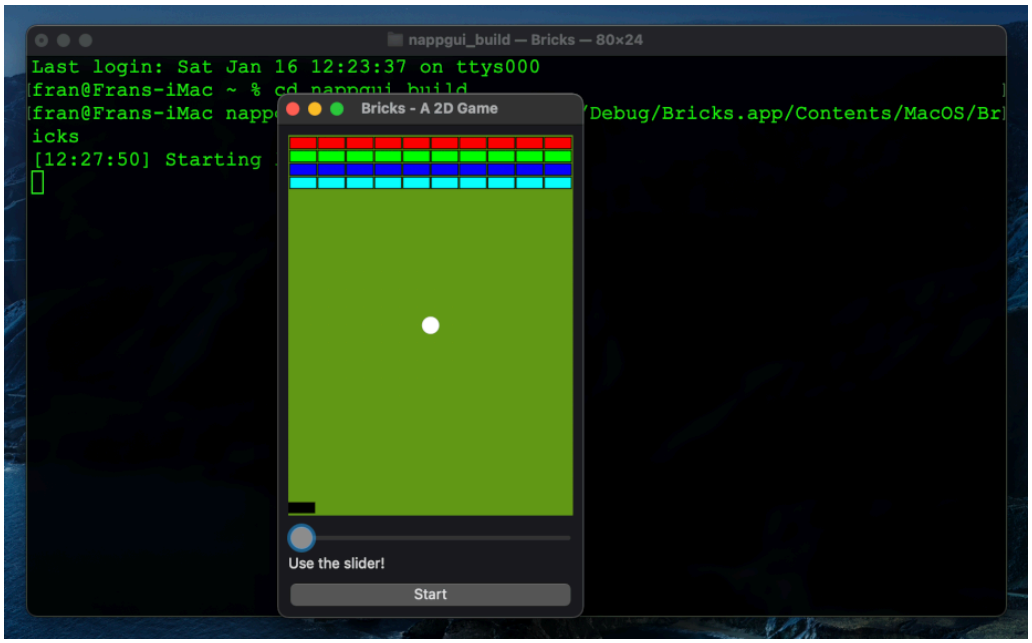


Figura 1.6: Ejecución del programa de ejemplo **Bricks** tras la compilación.

1.3. Inicio rápido en Linux

Antes de empezar, asegúrate que tienes instalados los compiladores, herramientas y librerías necesarias:

```
// Development tools
sudo apt-get install build-essential
sudo apt-get install git
sudo apt-get install cmake

// Development libraries
sudo apt-get install libgtk-3-dev
sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
sudo apt-get install libcurl4-openssl-dev
```

NAppGUI necesita, como mínimo ***gcc 4.6***, ***gtk3*** *(para aplicaciones de escritorio),* ***OpenGL*** *(si tu aplicación utiliza gráficos 3D) y* ***Curl*** *(para protocolos Web). Todo esto se incluye a partir de Ubuntu 12.04 LTS o distribuciones similares.*

Abre un terminal:

```
git clone --depth 1 https://github.com/frang75/nappgui_src.git
cd nappgui_src
cmake -S ./src -B ./build -DCMAKE_BUILD_CONFIG=Debug
cmake --build ./build -j 4
```

Una vez compiladas, ya podrás lanzar las aplicaciones de ejemplo existentes en el directorio `/build/Debug/bin` (Figura 1.7).

```
./build/Debug/bin/Die
./build/Debug/bin/Bricks
./build/Debug/bin/Products
./build/Debug/bin/Col2dHello
./build/Debug/bin/GuiHello
...
```

1.4. Licencia MIT

NAppGUI se distribuye bajo licencia MIT lo que significa, en esencia, que tienes total libertad para utilizar este software de forma libre y gratuita, tanto en proyectos comerciales como libres. La única restricción es que debes incluir una copia de esta Licencia[6] en cada parte sustancial del software que distribuyas.

[6] `https://www.nappgui.com/en/legal/license.html`

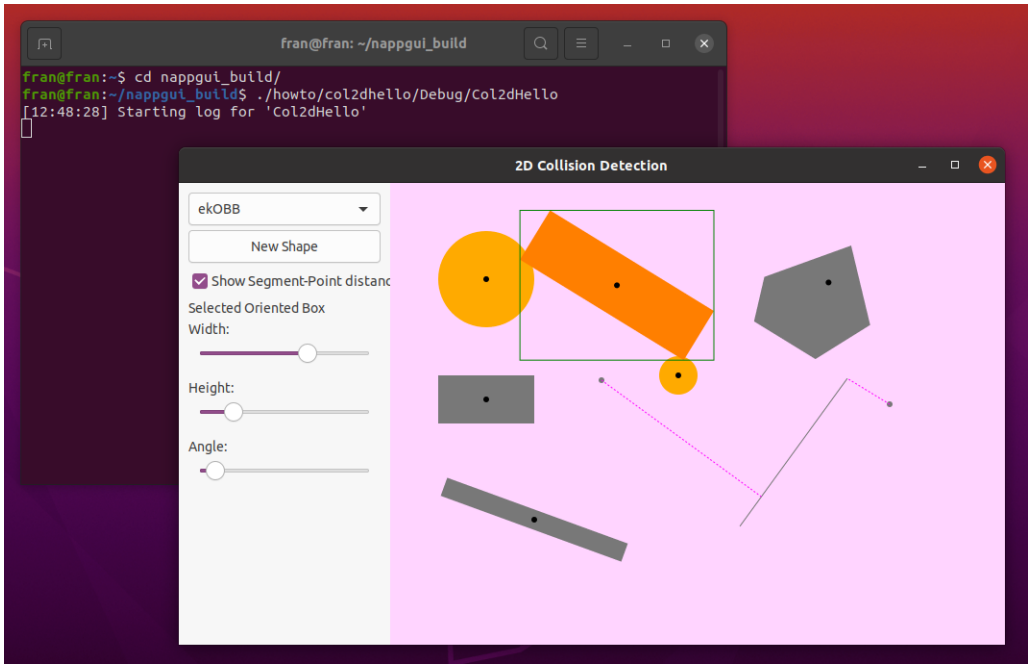


Figura 1.7: Ejecución del programa de ejemplo **Col2dHello** tras la compilación.

1.5. Conocimientos previos

Este libro no está dirigido a principiantes. Si bien el proyecto NAppGUI está orientado a simplificar la construcción de aplicaciones multiplataforma, requiere ciertos conocimientos previos por parte del usuario. Necesitarás, al menos, desenvolverte con soltura en C ó C++ ya que en ningún momento nos detendremos a explicar conceptos básicos de programación. Si vienes de Java o C# te convendría dar un repaso a los **punteros**. También necesitarás cierta habilidad con los entornos de desarrollo Visual Studio, Xcode y herramientas Unix como gcc, make o el intérprete de comandos.

Por otro lado, si eres un usuario avanzado, encontrarás un sistema sencillo para crear aplicaciones en C muy rápidas y de tamaño reducido que compilarán sin cambios en todos los entornos de escritorio. También tendrás a tu disposición un conjunto de librerías C para crear interfaces de usuario o aplicaciones por línea de comandos, sin necesidad de ensuciar tus proyectos con las engorrosas plantillas de clases que proporcionan **stl** o **boost**.

1.6. ¿Y ahora que?

- En *"Bienvenidos a NAppGUI"* (Página 13) continuamos con el tutorial.
- En *"¡Hola Mundo!"* (Página 23) vemos el código mínimo de una aplicación de escri-

torio.

- En *"Generar binarios NAppGUI"* (Página 65) vemos como compilar la versión estática o dinámica.
- En *"Compiladores e IDEs"* (Página 73) tendrás información sobre portabilidad.
- En *"Crear nueva aplicación"* (Página 101) empezarás a crear tus propias aplicaciones.
- En *"NAppGUI API"* (Página 147) tienes la documentación de las librerías y funciones.
- En *"Products"* (Página 437) tienes el código fuente de una aplicación de tamaño medio.

Capítulo 2

Bienvenidos a NAppGUI

Mientras que otros estaban contentos con escribir programas que solo resolvieran problemas, los primeros hackers estaban obsesionados con escribir programas que resolvieran problemas bien. Un nuevo programa que alcanzara el mismo resultado que otro existente pero que usase menos tarjetas perforadas se consideraba mejor, aunque hiciera lo mismo. La diferencia fundamental era cómo el programa lograba su resultado. - ***elegancia****.*

Jon Erickson - Hacking: The Art of Exploitation

NAppGUI es un SDK para crear aplicaciones nativas multiplataforma en C. Por **software nativo** entendemos aquel que se compila/ensambla utilizando las instrucciones específicas de la CPU (no está interpretado ni utiliza bytecode) y por **multiplataforma** la capacidad de generar versiones para Windows, macOS y Linux utilizando la misma base de código fuente (Figura 2.1). Desde sus primeras funciones escritas en agosto de 2010, el principal objetivo de NAppGUI ha sido simplificar al máximo la ardua tarea de crear aplicaciones con interfaz gráfica en C. Aunque ya existen diferentes soluciones, nosotros hemos optado por la simplicidad creando una ligera capa de abstracción que encapsula las tecnologías nativas, las unifica bajo un mismo API y añade cierta lógica para la gestión y automatización de tareas. Siendo algo más específicos, la filosofía en la que se basa el proyecto y algunas de sus características son:

- Rápido prototipado, evolución y mantenimiento en aplicaciones **de verdad**, al margen de los simples ejemplos que encontramos en la literatura e Internet.

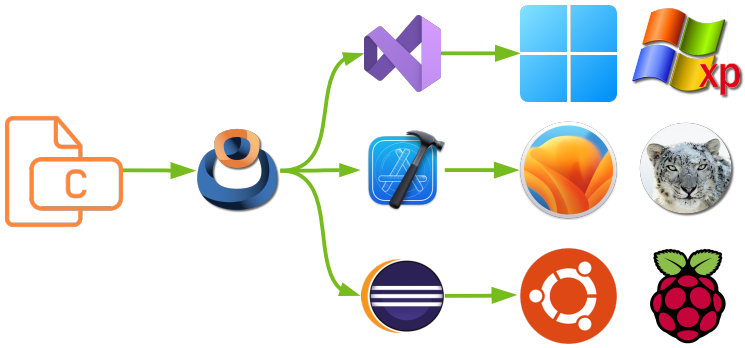


Figura 2.1: Desarrollo multiplataforma nativo con NAppGUI.

- La interfaz de usuario se describe mediante funciones ANSI-C, eliminando completamente el diseño visual. Este hecho facilita la creación de interfaces dinámicas, garantiza la portabilidad y posibilita el acceso al API desde cualquier lenguaje de programación.
- Las ventanas se componen y dimensionan automáticamente, sin que el programador tenga que indicar explícitamente las coordenadas y el tamaño de los controles.
- Es posible tener una aplicación completa en un solo archivo `.c`, eliminando los habituales archivos de recursos (`*.rc`, `*.xvid`, etc) y sus controladores asociados. El programador tiene total libertad a la hora de definir su propia estructura de archivos.
- Sincronización automática de las estructuras de datos internas con la interfaz o con canales de E/S. *"Data binding"* (Página 230).
- Gestión unificada de recursos lo que facilita la internacionalización. *"Recursos"* (Página 131).
- Traducciones entre idiomas en tiempo de ejecución sin necesidad de reiniciar la aplicación. *"Traducción en ejecuciónTraducción en ejecución"* (Página 136).
- La versión compilada de NAppGUI ocupa menos de 1Mb, y se distribuye en varias librerías estáticas que generan ejecutables de tamaño muy reducido. Esta es una gran ventaja con respecto a otras soluciones que requieren la distribución de pesadas *.DLLs*, en ocasiones más grandes que la aplicación en sí.
- Apariencia nativa: Las aplicaciones se integrarán en cada sistema respetando su estética original (Figura 2.2).
- *Back-ends*. El núcleo de NAppGUI proporciona estructuras y objetos para la creación de aplicaciones en línea de comandos, muy eficientes en servidores Windows o Linux.

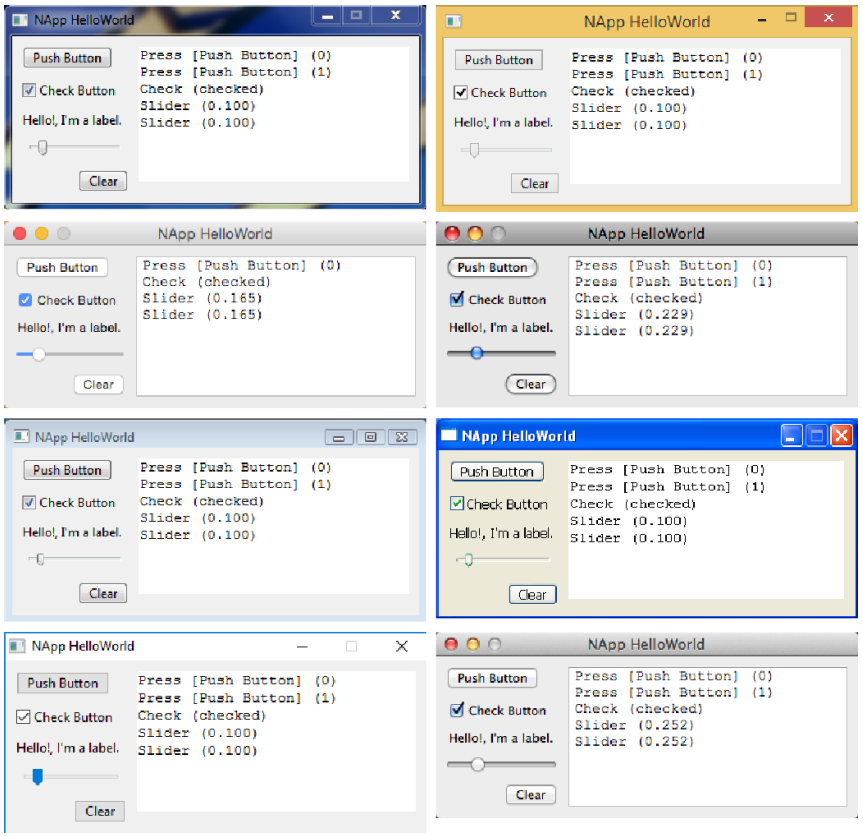


Figura 2.2: Apariencia nativa de la demo *Hello, World!*.

2.1. APIs originales

Microsoft, Apple y GNU/Linux proponen diferentes APIs para interactuar con sus sistemas. Esto significa que la misma aplicación debe ser reescrita para que funcione correctamente en cada plataforma. NAppGUI proporciona un conjunto unificado de funciones para crear interfaces gráficas de usuario y permitir el acceso directo a los recursos de la máquina (memoria, disco, red, etc.) (Figura 2.3). Cada implementación tiene en cuenta las condiciones particulares de la plataforma de destino y utiliza los comandos nativos apropiados para realizar la tarea de la manera más óptima posible.

2.2. Basado en C

A pesar de que hoy en día disponemos de una gran cantidad de lenguajes de programación, el lenguaje C sigue siendo el más potente y portable del mundo. El núcleo de Windows, macOS, Linux, Android, iOS y otros grandes programas están escritos en gran parte en C. En el mundo de las aplicaciones, su uso se ha visto disminuido un poco a favor

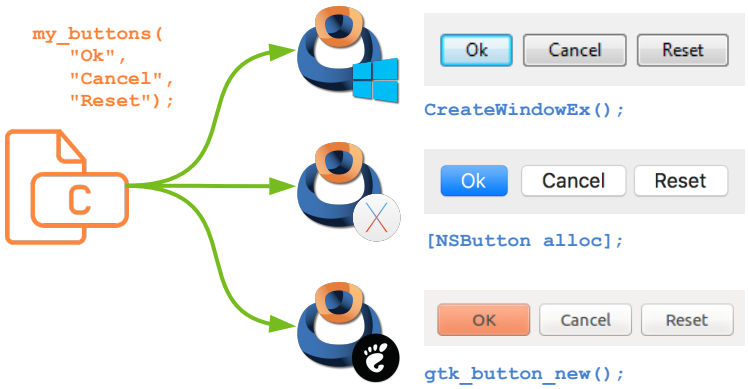


Figura 2.3: Llamadas a las APIs nativas, desde el mismo código fuente.

de otros con más *glamour*. Tal vez esta sea una de las razones por las que la ley de Wirth[1] sea cada vez día más cierta.

> *"El software se ralentiza más deprisa de lo que se acelera el hardware."*

NAppGUI está escrito, prácticamente en su totalidad, en lenguaje C con pequeñas partes en C++ y Objective-C. Este lenguaje es ampliamente soportado y compatible entre plataformas. En su desarrollo hemos prescindido de lenguajes minoritarios, propietarios o vinculados a una marca como: C#, Swift, Java u Objective-C. También de los interpretados (como Python o JavaScript) y de aquellos basados en máquinas virtuales (Java y C#) por la penalización en el rendimiento (Figura 2.4). Finalmente, no hemos utilizado C++, ya que no presentamos NAppGUI como una jerarquía de clases sino como una biblioteca de funciones. Nuestros objetivos han sido minimizar el impacto del SDK, simplificar la programación, aumentar la legibilidad y producir binarios de alto rendimiento.

Figura 2.4: Intérprete, máquina virtual y código binario. Cuanto más nos acercamos al lenguaje máquina, mayor rendimiento obtendremos del software.

[1] https://en.wikipedia.org/wiki/Wirth%27s_law

2.3. Sin editores visuales

La creación de interfaces gráficas puede convertirse en un proceso tedioso, ya que es difícil saber de antemano el tamaño final de elementos que contienen texto o imágenes, como es el caso de los botones. Por otro lado, las ventanas son entidades dinámicas sujetas a cambios en tiempo de ejecución (tamaño, traducción, cambio de subpaneles, áreas ocultas, etc.). Cuando usamos un editor visual, tenemos que ubicar los elementos en la posición y tamaño exactos (Figura 2.5). Esta es una tarea que requiere un uso intensivo del ratón, lo que ralentiza la conexión entre los objetos GUI y los controladores de eventos. En el ciclo de desarrollo, si los textos u otros elementos cambian (y por supuesto, cambiarán), tendremos que reubicar de nuevo los componentes a mano. Este problema crece en soluciones multilingües. Mantener a los desarrolladores moviendo píxeles y llenando formularios de propiedades es costoso para las empresas y muy aburrido para ellos. Todo esto sin mencionar que todos estos diseños visuales no serán compatibles entre plataformas (.rc Windows, .xib macOS, .glade GTK/Gnome, etc.).

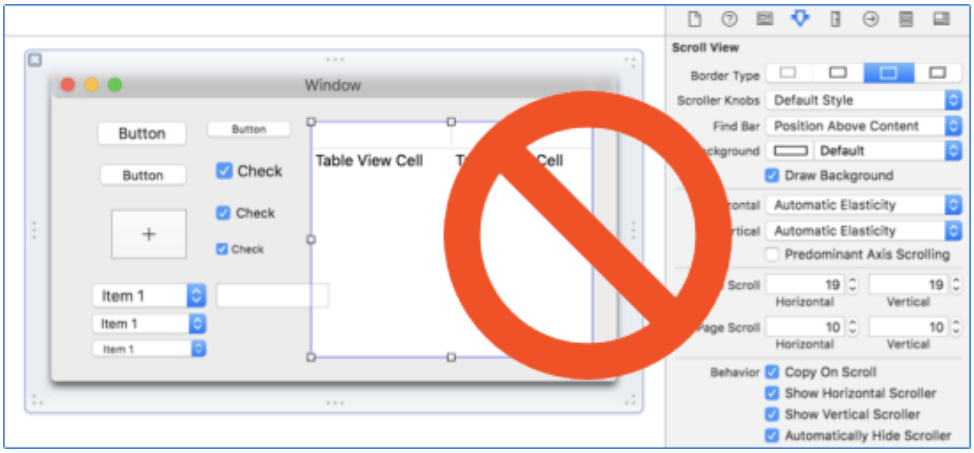


Figura 2.5: Los editores de recursos no son buenos aliados para crear complejas interfaces dinámicas.

> *Muchos programadores prefieren no mover las manos del teclado, ya que lo consideran mucho más productivo.*

NAppGUI utiliza una estrategia declarativa, donde solo es necesario indicar la celda donde se ubicará el elemento dentro de una rejilla rectangular (`Layout`). El tamaño y posición final se calcularán en tiempo de ejecución, realizando una composición recursiva de los *layouts* y *sublayouts* en función de su contenido (Listado 2.1) .

Listado 2.1: Creación de una ventana.

```
Panel *panel = panel_create();
Layout *layout = layout_create(1, 3);
```

```
Label *label = label_create();
Button *button = button_push();
TextView *view = textview_create();
Window *window = window_create(ekWINDOW_STD);
label_text(label, "Hello!, I'm a label");
button_text(button, "Click Me!");
layout_label(layout, label, 0, 0);
layout_button(layout, button, 0, 1);
layout_textview(layout, view, 0, 2);
layout_hsize(layout, 0, 250);
layout_vsize(layout, 2, 100);
layout_margin(layout, 5);
layout_vmargin(layout, 0, 5);
layout_vmargin(layout, 1, 5);
panel_layout(panel, layout);
window_panel(window, panel);
window_title(window, "Hello, World!");
```

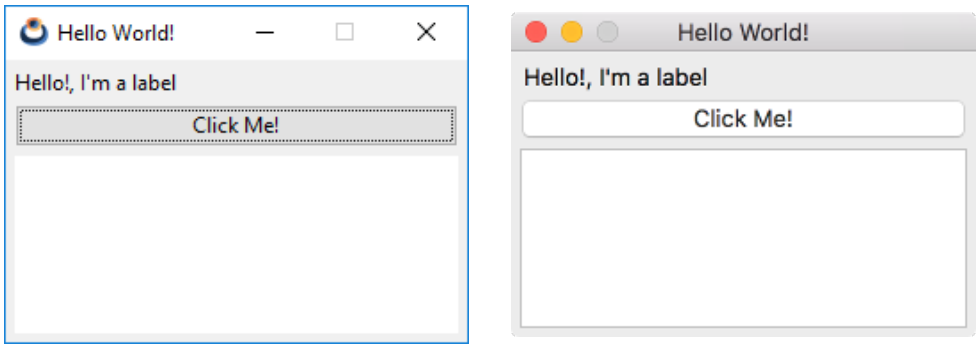


Figura 2.6: La composición declarativa es rápida, adaptable y portátil.

2.4. Dependencias

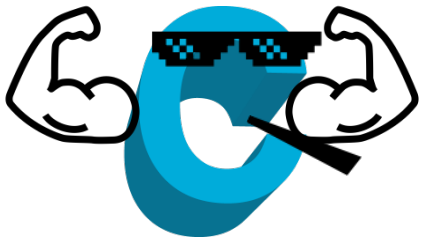
NAppGUI no utiliza librerías de terceros, tan solo conecta con las APIs nativas de cada sistema operativo. Este hecho, junto con el uso de C y el enlace estático, posibilita que:

- Las aplicaciones no necesitan entornos de ejecución adicionales, como ocurre con Python, Java o C#. Van directamente a la CPU a través del *scheduler* del sistema.
- Toda la aplicación puede estar contenida en un solo archivo `.exe`. Se enlaza el mínimo código posible y no es necesario distribuir ninguna `.dll` adicional.

A partir de la versión 1.3, NAppGUI soporta la generación de librerías dinámicas.

- Las aplicaciones ocupan muy poco espacio en disco, ya que todas sus dependencias se encuentran presentes de forma natural en los sistemas donde se ejecutan.

- El rendimiento es máximo, ya que están compiladas en código máquina nativo, utilizando el mayor nivel de optimización que soporta cada CPU.
- Se pueden editar, compilar y ejecutar en plataformas obsoletas a día de hoy como un Pentium III con Visual Studio 2005 y WindowsXP.
- Con NAppGUI podemos moverlas de Windows a macOS o Linux, sin tocar una sola línea de código fuente. Ver *"Compiladores e IDEs"* (Página 73).



Tres paquetes dentro del SDK actuarán como *wrappers* tecnológicos (Figura 2.7), ocultando los detalles específicos de cada plataforma bajo una interfaz común, sin que esto suponga una sobrecarga para el programa.

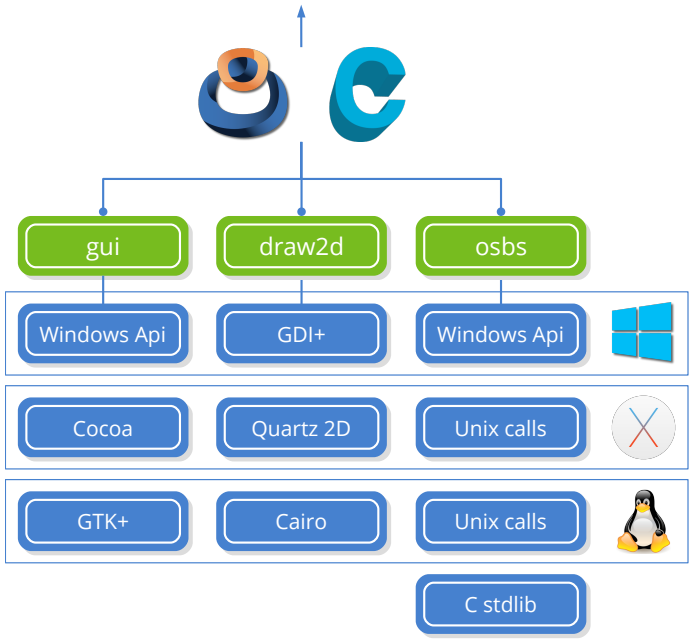


Figura 2.7: Diferentes tecnologías en la base de NAppGUI. En *"NAppGUI API"* (Página 147) tienes el esquema completo.

- *"Osbs"* (Página 168): *Operating System Basic Services*. API sobre archivos y directorios, procesos, hebras, memoria, etc.
- *"Draw2D"* (Página 262): API sobre dibujo vectorial 2d, imágenes y fuentes tipográficas.

- *"Gui"* (Página 303): API sobre interfaces gráficas: Ventanas, controles y menús.
- **Llamadas al sistema Unix**: En sistemas tipo Unix (Linux, macOS) es la forma en que un programa se comunica con el núcleo para realizar alguna tarea relacionada con archivos, procesos, memoria, red o hardware en general.
- **API de Windows**: Es el API de más bajo nivel proporcionado por Microsoft para la programación bajo Windows. Es muy amplio e integra diferentes aspectos del desarrollo:
  - kernel32.dll: El equivalente a las llamadas Unix (archivos, procesos, memoria, etc).
  - ws2_32.dll: Proporciona funciones de red TCP/IP (Las llamadas Unix incluyen el soporte TCP/IP).
  - user32.dll, comctl32.dll, comdlg32.dll, uxtheme.dll: Implementa controles estándar para interfaces gráficas de usuario (etiquetas, cajas de edición, combos, barras de progreso, diálogos comunes, etc.).
- **Cocoa**: API de programación orientado a objetos de los sistemas Mac OSX (ahora macOS). Está escrito en Objective-C, por lo tanto, no es accesible directamente desde C "puro". Cocoa se basa en OpenStep, el API de NeXTSTEP, el sistema operativo creado por Steve Jobs cuando fue despedido de Apple. En 1996, Apple compra NeXT y recupera a Jobs, que utiliza su tecnología como base para el nuevo Macintosh. Muchas clases en Cocoa aún conservan el prefijo NS como herencia NeXTSTEP. Aunque hay un API basado en C de nivel inferior llamado Carbon, este se encuentra discontinuado desde Mac OSX 10.4 Tiger. No tiene acceso a todas las funcionalidades del sistema ni es compatible con las aplicaciones de 64 bits. Por tanto, Cocoa es el API actual de más bajo nivel para los sistemas de Apple.
- **Gtk+**: Acrónimo de **G**IMP **T**ool**K**it. Es una librería de alto nivel para crear interfaces gráficas con multitud de objetos predefinidos (llamados *widgets*). Es una de las más extendidas en sistemas GNU/Linux, pero en realidad es multiplataforma con versiones para Windows y macOS. Entornos de escritorio como Gnome, Xfce o aplicaciones como GIMP se basan en GTK.
- **GDI+**: Es la evolución de GDI *(Graphics Device Interface)*, un API de dibujo vectorial 2d desarrollado por Microsoft para la primera versión de Windows de 16 bits. GDI+ se introdujo con WindowsXP como un conjunto de clases C++ y está encapsulado en la plataforma .NET a través del espacio de nombres `System.Drawing`. También es accesible directamente desde C a través del *GDI+ Flat API*, pero Microsoft recomienda utilizarla a través de las clases de C++. Incorpora mejoras sustanciales con respecto a GDI, como coordenadas de punto flotante, transformaciones afines, antialias, sombreado con degradados y compatibilidad con formatos de imagen como JPG, PNG o GIF. El dibujo con máscaras y la incompatibilidad con PDF

son los dos inconvenientes más notables en comparación con Quartz 2D y Cairo, sus "competidores" directos en otras plataformas.

- **Quartz 2D**: Es el nombre comercial de *Core Graphics*, la poderosa API de dibujo del macOS. Al igual que Cocoa, Core Graphics es una evolución de las bibliotecas gráficas de NeXTSTEP y llegó a Apple después de la adquisición de NeXT. Quartz 2D se basa en los formatos Adobe PostScript y PDF, incorporando canal alfa y antialias. Los Mac clásicos (pre-NeXT) utilizaban la biblioteca *QuickDraw*, desarrollada inicialmente por Bill Atkinson para el Apple Lisa. Los macs modernos aún incorporan QuickDraw, pero Xcode ya no proporciona cabeceras, por lo que no se puede utilizar en nuevos proyectos. Core Graphics es un API basado en C y todas sus funciones comienzan con el prefijo **CG**.

- **Cairo**: Cairo es una librería de dibujo vectorial 2d basada en C. A diferencia de GDI+ o Quartz 2D, es multiplataforma, se puede descargar de forma independiente e incorporarla en cualquier proyecto (bajo licencia LGPL). Desde su versión 3, GTK+ utiliza Cairo para todas las tareas de dibujado de widgets. GTK+2 también utilizaba Cairo para generar los documentos PDF para imprimir. NAppGUI usa Cairo para implementar el API **draw2d** en la plataforma GNU/Linux, ya que esta librería se encuentra de forma natural en todos los entornos de escritorio basados en GTK+: Gnome, Cinnamon, LXDE, Mate, Pantheon, Sugar o Xfce. Técnicamente, Cairo es bastante avanzada, equiparándose a Quartz 2D en términos de funcionalidad. Admite transformaciones afines, máscaras de imagen, curvas de bezier, procesamiento de texto y dibujo sobre superficies PDF y PostScript.

- **C stdlib**: C es un lenguaje pequeño y hermoso, pero no proporciona funciones de apoyo adicionales. Durante la década de 1970, el lenguaje C se hizo muy popular y los usuarios comenzaron a compartir ideas sobre cómo resolver tareas comunes y repetitivas. Con su estandarización en la década de los 80, algunas de estas ideas se convirtieron en la librería estándar de C, que proporciona un conjunto básico de funciones matemáticas, manipulación de cadenas, conversiones de tipo y entrada/salida. NAppGUI integra de una forma u otra la funcionalidad de la librería estándar, por lo que no recomendamos su uso en aplicaciones finales (ver *"Sewer"* (Página 151)).

2.5. Bajo y alto nivel

Durante su diseño e implementación, NAppGUI ha tratado de mantener un balance equilibrado entre la programación de bajo y alto nivel. Los amantes del bajo nivel encontrarán una especie de *librería C extendida y multiplataforma* para acceder al sistema, los elementos de interfaz y los comandos de dibujo. No obstante, aún conservarán el poder para crear código optimizado y el acceso directo a la memoria. ¡Recuerda, estamos en C!

Por otro lado, NAppGUI integra algunas soluciones de alto nivel como la gestión de re-

cursos, la composición de interfaces, las traducciones automáticas o la vinculación de datos entre otras. NAppGUI también incorpora scripts CMake para la creación automatizada de proyectos en Visual Studio, Xcode o Eclipse/Make.

Finalmente, son los desarrolladores los que deciden con qué librerías enlazar según las necesidades del proyecto y con el grado de automatización que deseen adoptar. Cada aplicación basada en NAppGUI realiza un enlace estático de todas sus dependencias, por lo que ni el ejecutable ni su distribución final tendrán rastro de código binario innecesario. De esta manera, produciremos ejecutables autocontenidos de tamaño reducido que no requerirán de instalador ni incluirán megabytes de dependencias en forma de .DLLs.

Capítulo 3

¡Hola Mundo!

Érase una vez una compañía llamada Taligent. Taligent fue creada por IBM y Apple para desarrollar un conjunto de herramientas y bibliotecas como Cocoa. En el momento en que Taligent alcanzó cierto grado de madurez, Aaron conoció a uno de sus ingenieros en una feria comercial y le pidió que creara una aplicación simple: Una ventana con un botón. Cuando se hace clic en el botón, las palabras "¡Hola mundo!" Aparecen en un campo de texto. El ingeniero creó un proyecto y comenzó a crear subclases locamente tanto para la ventana como para el botón y el controlador de eventos. Luego comenzó a generar código: docenas de líneas para obtener el botón y el campo de texto en la ventana. Después de 45 minutos, todavía estaba intentando que la aplicación funcionara. Un par de años después, Taligent cerró silenciosamente sus puertas para siempre.

Hillegass, Preble & Chandler - Cocoa Programming for OSX.

Poco podemos decir del significado del programa *Hello World!* cada vez que nos enfrentamos a una nueva tecnología o metodología de programación. Por lo tanto, vamos al grano.

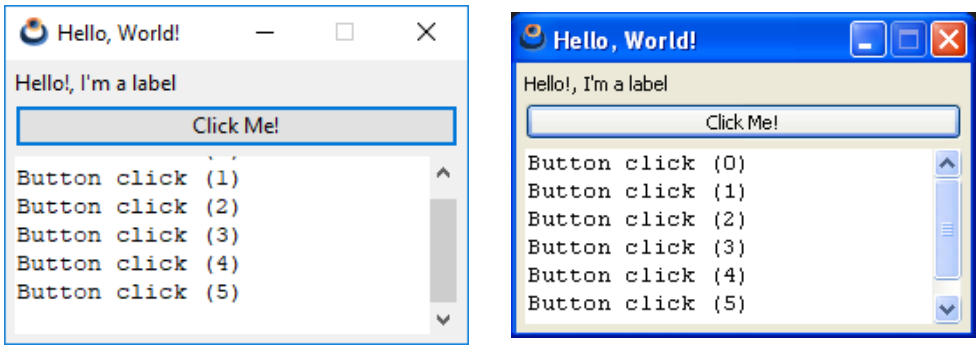


Figura 3.1: Windows 10 y WindowsXP.

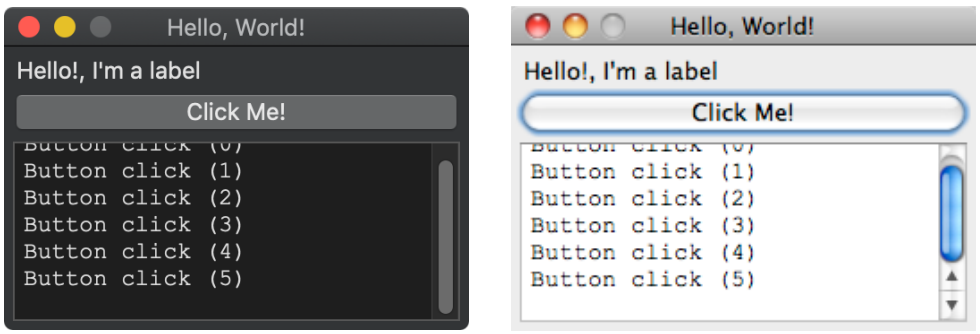


Figura 3.2: macOS 10.14 Mojave y MacOSX 10.6 Snow Leopard.

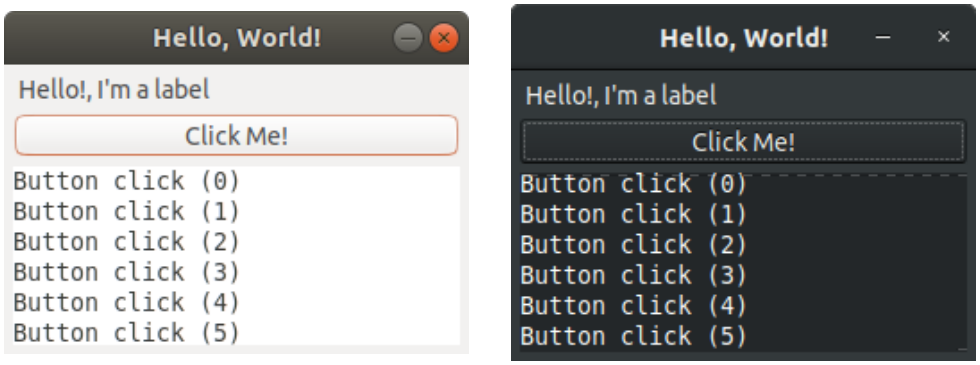


Figura 3.3: GTK+3 Ambiance (Ubuntu) y Adwaita Dark (Raspbian).

3.1. El programa completo

Listado 3.1: demo/hello/main.c

```
/* NAppGUI Hello World */

#include <nappgui.h>
```

```
typedef struct _app_t App;

struct _app_t
{
    Window *window;
    TextView *text;
    uint32_t clicks;
};

/*-------------------------------------------------------------------------*/

static void i_OnButton(App *app, Event *e)
{
    String *msg = str_printf("Button click (%d)\n", app->clicks);
    textview_writef(app->text, tc(msg));
    str_destroy(&msg);
    app->clicks += 1;
    unref(e);
}

/*-------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 3);
    Label *label = label_create();
    Button *button = button_push();
    TextView *text = textview_create();
    app->text = text;
    label_text(label, "Hello!, I'm a label");
    button_text(button, "Click Me!");
    button_OnClick(button, listener(app, i_OnButton, App));
    layout_label(layout, label, 0, 0);
    layout_button(layout, button, 0, 1);
    layout_textview(layout, text, 0, 2);
    layout_hsize(layout, 0, 250);
    layout_vsize(layout, 2, 100);
    layout_margin(layout, 5);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    panel_layout(panel, layout);
    return panel;
}

/*-------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
```

```
    unref(app);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STD);
    window_panel(app->window, panel);
    window_title(app->window, "Hello, World!");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

3.2. El esqueleto

Una aplicación NAppGUI comienza en `osmain`, una macro multiplataforma que unifica la iniciación de un programa de escritorio bajo diferentes sistemas. Está definida en `#include "osmain.h"` y recibirá cuatro parámetros: constructor, destructor, argumentos (`char_t`) y el tipo de objeto. De esta forma, cualquier esqueleto básico se ve así:

```
#include "nappgui.h"

typedef struct _app_t App;
struct _app_t
{
    Window *window;
};

static App *i_create(void)
{
```

```
    App *app = heap_new0(App);
    return app;
}

static void i_destroy(App **app)
{
    heap_delete(app, App);
}

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

La directiva `#include "nappgui.h"`, incluye gran parte de NAppGUI con una sola instrucción. Si lo prefieres, puedes optar por incluir las cabeceras por separado según sea necesario. En este caso, deberíamos reemplazar un solo `#include` por once. En el Manual de Referencia, se indica que cabecera incluir según el módulo de funciones que vayamos a utilizar.

```
#include "gui.h"
#include "button.h"
#include "heap.h"
#include "label.h"
#include "layout.h"
#include "listener.h"
#include "panel.h"
#include "strings.h"
#include "v2d.h"
#include "vtext.h"
#include "window.h"
```

3.3. El constructor

El primer parámetro de `osmain` es el constructor de la aplicación. Tan pronto como se inicia el programa, ciertas estructuras internas deben inicializarse, así como arrancar el ciclo de mensajes inherente a todas las aplicaciones de escritorio. Cuando todo esté listo, se llamará al constructor para crear el **objeto aplicación**. Este objeto puede ser de cualquier tipo y no necesita derivarse de ningún `class Application` o similar, estamos en C ;-). Dada la sencillez de este ejemplo, el objeto aplicación tan solo contiene una ventana.

```
static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STD);
    window_panel(app->window, panel);
    return app;
}
```

3.4. El panel principal

Para crear la ventana principal, necesitamos el **panel principal**, un contenedor que integra todos los controles de interfaz que se muestran en la ventana. El espacio dentro del panel está organizado en una cuadrícula invisible llamada `Layout`. Cada panel puede tener varios layouts y alternar entre ellos, pero al menos uno es necesario. Dentro de sus celdas ubicaremos los diferentes controles de interfaz.

```
static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 3);
    Label *label = label_create();
    Button *button = button_push();
    TextView *text = textview_create();
    label_text(label, "Hello!, I'm a label");
    button_text(button, "Click Me!");
    layout_label(layout, label, 0, 0);
    layout_button(layout, button, 0, 1);
    layout_textview(layout, text, 0, 2);
    panel_layout(panel, layout);
    return panel;
}
```

3.5. El destructor

Cuando la aplicación termine, `osmain` llamará al destructor (segundo parámetro de la macro) para liberar el objeto aplicación y todo lo que dependa de él, con el fin de realizar una salida limpia del programa. Pondremos **mucho énfasis en esto**, ya que el hecho de no liberar correctamente toda la memoria se considerará un grave error de programación.

```
static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}
```

3.6. Lanzar la ventana

Por defecto, NAppGUI crea todas las ventanas ocultas, por lo que es necesario visualizarlas explícitamente. Establecemos un título, una posición inicial y la lanzamos con `window_show`. Observamos que en esta primera versión nuestra ventana no queda muy estética (Figura 3.4). En un momento le daremos formato.

```
static App *i_create(void)
{
```

```
    ...
    window_title(app->main_window, "Hello World!");
    window_origin(app->main_window, v2df(500, 200));
    window_show(app->main_window);
    ...
}
```

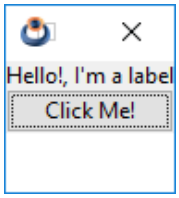


Figura 3.4: Primera versión de *¡Hola, Mundo!* (sin formato).

3.7. Formato del Layout

Para mejorar la apariencia de nuestra ventana, vamos a darle un poco de formato al diseño. Concretamente, vamos a establecer un ancho de columna y un alto para la tercera fila (control de texto). Después dejaremos un margen en el borde y una separación entre filas. (Figura 3.5).

```
layout_hsize(layout, 0, 200);
layout_vsize(layout, 2, 100);
layout_margin(layout, 5);
layout_vmargin(layout, 0, 5);
layout_vmargin(layout, 1, 5);
```

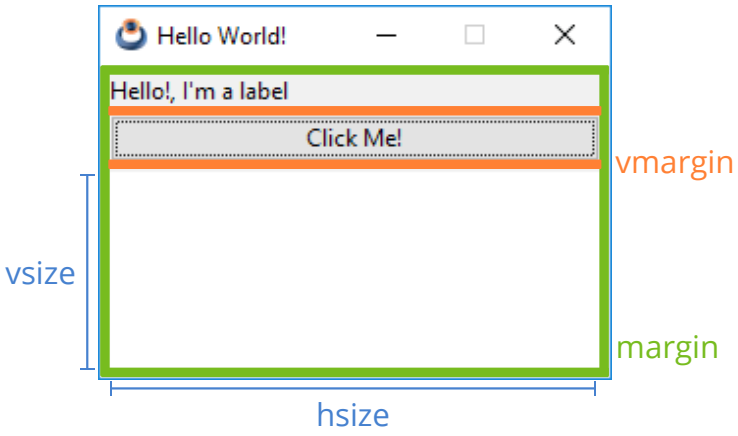


Figura 3.5: *¡Hola, Mundo!* tras el formateo del Layout.

3.8. Saliendo del programa

Cuando presionamos el botón de cerrar la ventana principal, el programa no finaliza su ejecución. Esto es típico de las aplicaciones macOS, donde aún siguen corriendo en el Dock aunque no haya ninguna ventana abierta. NAppGUI sigue el mismo criterio de no cerrar el programa, por lo que debemos realizar una llamada explícita a la función `osapp_finish`. Para ello, capturaremos el **evento del botón**, a través de la macro `listener`.

```
static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
}

static App *i_create(void)
{
   window_OnClose(app->main_window, listener(app, i_OnClose, App));
}
```

3.9. Eventos Button

Finalmente, capturaremos el evento *clic* del botón e imprimiremos un mensaje en el cuadro de texto cada vez que se pulse. Vamos a implementar el manejador `i_OnButton`, responsable de componer y mostrar el mensaje y lo conectaremos al control Button que hemos creado anteriormente.

```
static void i_OnButton(App *app, Event *e)
{
    String *msg = str_printf("Button click (%d)\n", app->clicks);
    text_insert(app->vtext, tc(msg));
    str_destroy(&msg);
    app->clicks += 1;
}
...
button_OnClick(button, listener(app, i_OnButton, App));
```

Un evento es una acción que ocurre durante la ejecución del programa. El sistema operativo lo captura y nos lo envía a través de su controlador (definido en `listener()`). Más en "Eventos" (Página 235).

Capítulo 4

Uso de C

La mayoría de los lenguajes de programación contienen partes buenas y partes malas. Descubrí que podía ser un mejor programador usando solo las partes buenas y evitando las partes malas. Después de todo, ¿cómo se puede construir algo bueno con partes malas?

Douglas Crockford - JavaScript: The Good Parts.

Programar de forma rápida, reducir la probabilidad de error, asegurar la portabilidad y generar binarios optimizados han sido los principales propósitos de NAppGUI desde sus inicios y eso incluye una revisión del propio lenguaje C. Se ha utilizado como base un **subconjunto del ANSI-C90** con los enteros de tamaño fijo `<stdint.h>`, característica introducida en C99. Recomendamos que las aplicaciones basadas en este SDK sigan la misma filosofía. Entrando más en detalle, los objetivos perseguidos han sido estos:

- Máxima portabilidad: Incluso en compiladores ya desfasados como MSVC 8.0 (Visual Studio 2005) o GCC 4.2 (Xcode 3). Las últimas características del lenguaje pueden no estar disponibles en plataformas donde debas portar tu código (piensa en

dispositivos embebidos). También aseguras que dicho código será compatible con las futuras versiones de los principales compiladores.

- Focalizar la atención: En el "que" y no en el "como". Hay veces que volvemos complicado lo sencillo tan solo por justificar el uso de esa nueva característica tan "cool". También es posible que seas un adicto por "estar a la última", lo que te obligará a "modernizar" el código para adaptarlo a una nueva versión del estándar. Céntrate en resolver el problema que tengas delante y, si puedes invertir más tiempo, en bajar la complejidad asintótica de tu solución. NAppGUI se encargará de que tus aplicaciones funcionen allá donde sea necesario.
- Evitar características poco relevantes: Como el soporte multi-hilo (`<threads.h>`) de C11. Esto se resuelve con llamadas al sistema. Ver *"Hebras"* (Página 172).
- Rápida compilación: Determinadas construcciones del C no son más que una especie de "ensamblador portátil", que el compilador puede interpretar y traducir de una manera increíblemente eficiente.
- Binarios pequeños y rápidos: Derivada de la anterior, el código generado requerirá pocas sentencias en ensamblador y será muy sencillo de optimizar por parte del compilador.

Evidentemente, este no es lugar para aprender C ni es nuestra pretensión. El núcleo del lenguaje es pequeño y fácil de recordar, pero programar bien requiere años de práctica. Lo que haremos aquí es mostrar la mínima expresión del lenguaje que utilizamos diariamente. En definitiva, estos son nuestros estándares.

4.1. Tipos básicos

- **Vacío:** `void`.
- **Booleano:** `bool_t`. Tipo de 8 bits con solo dos valores posibles `TRUE` (1) y `FALSE` (0).
- **Enteros:** `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`, `int8_t`, `int16_t`, `int32_t`, `int64_t`. Los enteros de tamaño fijo se introdujeron en C99 mediante `<stdint.h>`. Consideramos una ventaja saber que nuestras variables tendrán el mismo tamaño en todos los sistemas. Queda prohibido el uso de `int`, `long`, `short` o `unsigned`, con la única excepción de las funciones de comparación.

```
static int i_cmp_cir(const Cir2Dd *cir1, const Cir2Dd *cir2)
{
    return (cir1->r < cir2->r) ? 1 : -1;
}

arrst_sort(circles, i_cmp_cir, Cir2Dd);
```

- **Coma flotante:** `real32_t`, `real64_t`. No se utiliza `float` ni `double` por coherencia con los tipos enteros.
- **Carácter:** `char_t` (8 bits). Se utiliza "de facto" la representación UTF8 en todo el SDK, por lo que queda prohibido el acceso aleatorio a los elementos de una cadena, ya que se trata de una codificación de longitud variable. Se deben utilizar las funciones incluidas en *"Unicode"* (Página 157) o *"Strings"* (Página 196) para manipular arrays de carácteres. No se utilizan (ni se aconseja) los tipos `wchar_t`, `char16_t`, `char32_t` . No obstante, si tuvieses cadenas *wide-char* deberás convertirlas a UTF8 antes de utilizarlas en cualquier función de NAppGUI.

Uso de cadenas UTF8

```
/* Error! */
const char_t *mystr = "Ramón tiene un camión";
while (mystr[i] != '\0')
{
    if (mystr[i] == 'ó')
    {
        /* Do something */
    }
    else
    {
        i += 1;
    }
}

/* Correct! */
const char_t *it = mystr;
uint32_t cp = unicode_to_u32(it, ekUTF8);
while (cp != '\0')
{
    if (cp == 'ó')
    {
        /* Do something */
    }
    else
    {
        it = unicode_next(it, ekUTF8);
        cp = unicode_to_u32(it, ekUTF8);
    }
}

/* Avoid using wchar_t constants (when possible).
   wchar_t uses UTF16 encoding */
const wchar_t *mywstr = L"Ramón tiene un camión";
char_t mystr[512];

unicode_convers((const char_t*)mywstr, mystr, ekUTF16, ekUTF8, sizeof(
    ↪ mystr));
```

```
/* This is a NAppGUI function (UTF8-Encoding) */
label_text(label, mystr);
```

- **Enumerados:** Su principal cometido es manejar la especialización y serán evaluados en exclusiva dentro de un `switch`. Queda prohibido asignar valores aleatorios a los elementos de un `enum`, salvo `1` al primero de ellos. Se considera `0` como **no inicializado** y `ENUM_MAX(align_t)` como **inválido**.

Definición de tipos enumerados

```
typedef enum _align_t
{
    ekTOP = 1,
    ekBOTTOM,
    ekLEFT,
    ekRIGHT
} align_t;
```

4.2. Estructuras y uniones

Definición de estructuras y uniones

```
typedef struct _layout_t Layout;
typedef union _attr_t Attr;

struct _layout_t
{
    Cell *parent;
    Panel *panel;
    bool_t is_row_major_tab;
    ArrSt(Cell) *cells;
    ArrPt(Cell) *cells_dim[2];
    real32_t dim_margin[2];
    color_t bgcolor;
    color_t skcolor;
};

union _attr_t
{
    struct _bool_
    {
        bool_t def;
    } boolt;

    struct _int_
    {
        int64_t def;
        int64_t min;
        int64_t max;
```

```
        int64_t incr;
        String *format;
    } intt;

    struct _real32_
    {
        real32_t def;
        real32_t min;
        real32_t max;
        real32_t prec;
        real32_t incr;
        uint32_t dec;
        String *format;
    } real32t;
};
```

Por lo general, las definiciones de estructuras no serán públicas y permanecerán ocultas en el `*.c`. Esto significa que no podrán declararse variables automáticas en el *"Segmento StackSegmento Stack"* (Página 164) y solo serán accesibles mediante funciones que acepten **objetos dinámicos opacos**.

Uso de punteros opacos

```
Layout *layout = layout_create(2, 2);
layout_edit(layout, edit, 0, 0);
layout_label(layout, label, 0, 1);
...
panel_layout(panel, layout);
```

```
/* Layout definition is hidden
   We do not know the content of Layout */
Layout layout;  /* Compiler error! */
```

Normalmente, todos los objetos dinámicos dispondrán de una función destructora. Si no existiera, es porque dicho objeto **solo tiene sentido como parte de otro**. Por ejemplo, no existe `layout_destroy()` ni `panel_destroy()`, pero sí `window_destroy` que se encargará de destruir toda la jerarquía de paneles y layouts asociados a la ventana.

4.3. Control

- **if/else**. Siempre abren un bloque `{...}`, a no ser que TODOS los caminos estén compuestos por una única sentencia. Por lo general, se evita el uso de funciones como argumentos del `if/else` con la excepción de **funciones puras**.

  Uso de if/else

  ```
  if (x == 1)
      i_do_something(j);
  ```

```
else
    i_do_nothing();

if (x == 1)
{
    j += 2;
    i_do_something(j);
}
else
{
    i_do_nothing();
}

if (bmath_sqrtf(sqlen) < 20.5f)
    i_do_something(j);
```

- **while**. Nada que comentar.
- **do/while**. No permitido. Utilizar `for` o `while`.
- **for**. Para bucles infinitos, se utiliza `for(;;)` en lugar de `while(TRUE)`, ya que evita warnings en algunos compiladores. Dado que hay compiladores basados en ANSI-C, como MSVC++ 8.0, **no utilizamos declaraciones de variables** dentro del `for()`, característica que fué incorporada en C99.

Uso de for

```
/* Infinite loop */
for(;;)
{
    ...
}

/* Will not work in some compilers (not used) */
for (uint32_t i = 0; i < 1024; ++i)
{
    ...
}

/* Ok */
uint32_t i = 0;
...
for (i = 0; i < 1024; ++i)
{
    ...
}
```

- **switch**. Únicamente se utiliza para discriminar entre los valores de un **enum**. NUNCA se evaluará otro tipo de datos en un `switch` ni tampoco se discriminará un `enum`

dentro una construcción `if/else`. El compilador puede optimizar drásticamente el rendimiento de una construcción con estas características.

Uso de switch

```
switch(align) {
case ekTOP:
    ...
    break;

case ekBOTTOM:
    ...
    break;

case ekLEFT:
    ...
    break;

case ekRIGHT:
    ...
    break;

cassert_default();
}
```

4.4. Funciones

- Una función puede no devolver nada (`void`), un tipo básico o un puntero.
- Los parámetros de entrada siempre son **`const`** aunque sean tipos simples pasados por valor.
- Para cualquier parámetro de entrada que no sea de tipo básico se pasará por puntero. Nunca una estructura por valor.
- Para los parámetros de salida se utilizarán siempre punteros. En C no existen las referencias.

Parámetros en funciones.

```
uint32_t myfunc(const uint32_t input1, const Layout *input2, V2Df *output1
    ↪ , real32_t *output2);
```

- Se debe reducir al máximo la cantidad de funciones públicas, que se declararán en el `*.h` y se definirán en el `*.c`.
- Las funciones de apoyo (o privadas) se definirán `static`, dentro del módulo `*.c` y no tendrán declaración.

Función pública.

```c
/* layout.h */
void layout_hsize(Layout *layout, const uint32_t col, const real32_t wid);

/* layout.c */
void layout_hsize(Layout *layout, const uint32_t col, const real32_t wid)
{
    i_LineDim *dim = NULL;
    cassert_no_null(layout);
    cassert_msg(wid >= 0.f, "Column 'width' must be positive.");
    dim = arrst_get(layout->lines_dim[0], col, i_LineDim);
    cassert_no_null(dim);
    dim->forced_size = wid;
}
```

Función privada. Solo puede llamarse dentro de `layout.c`.

```c
/* layout.c */
static Cell *i_get_cell(Layout *lay, const uint32_t c, const uint32_t r)
{
    register uint32_t position = UINT32_MAX;
    cassert_no_null(lay);
    cassert(c < arrst_size(lay->lines_dim[0], i_LineDim));
    cassert(r < arrst_size(lay->lines_dim[1], i_LineDim));
    position = r * arrst_size(lay->lines_dim[0], i_LineDim) + c;
    return arrst_get(lay->cells, position, Cell);
}
```

4.5. Ámbitos

Las variables se declaran al principio de un bloque y no podrán mezclarse con sentencias, a no ser que abramos un nuevo ámbito. Declaraciones mezcladas con sentencias es una característica de C++ añadida al estándar C99, pero no todos los compiladores de C la soportan. Sí que está permitido inicializar una variable llamando a una función.

Ámbitos de variables en C

```c
{
    /* Ok! */
    uint32_t var1 = 5;
    uint32_t var2 = i_get_value(stm);
    uint32_t var3 = i_get_value(stm);

    i_add_values(var1, var2, var3);

    /* Error in C90 compilers */
    uint32_t var4 = 6;

    /* Ok! */
```

```
    {
        uint32_t var4 = 6;
        ....
    }
}
```

4.6. Punteros

Al margen de las ventajas propias del uso de la aritmética de punteros a la hora de implementar ciertos algoritmos, en NAppGUI se utilizan punteros esencialmente en dos situaciones:

- Paso de parámetros a una función, cuando dicho parámetro no sea un tipo básico.

Paso de parámetros mediante punteros.

```
V2Df v1 = v2df(10, 43.5f);
V2Df v2 = v2df(-4.8f, val);
V2Df v3 = v2d_addf(&v1, &v2);

/* v2d.h */
V2Df v2d_addf(const V2Df *v1, const V2Df *v2);
```

- Manejo de objetos opacos. Donde la definición del `struct` no está disponible y, por tanto, la única forma de comunicarnos con el objeto es mediante funciones que acepten un puntero al mismo.

Uso de objetos opacos.

```
const V2Df pt[] = { {4,1}, {2,5}, {-3,5}, {-4,2}, {0,-3} };
Pol2Df *pol = pol2d_createf(pt, 5);
real32_t a = pol2d_areaf(pol);

...
pol2d_destroyf(&pol);

/* pol2d.h */
Pol2Df* pol2d_createf(const V2Df *points, const uint32_t n);

void pol2d_destroyf(Pol2Df **pol);

real32_t pol2d_areaf(const Pol2Df *pol);
```

Mención aparte merecen los **punteros a función** muy utilizados en C, pero menos en C++ ya que el lenguaje los oculta dentro de las **vtable**. No obstante, un puntero a función estratégicamente colocado puede simplificarnos el hecho de añadir funcionalidad especializada a objetos ya existentes, sin tener que adoptar un diseño orientado a objetos mas purista.

Listado 4.1: Uso de punteros a función.

```
typedef struct _shape_t Shape;
typedef void(*FPtr_draw)(const Shape*, DCtx *ctx);

struct _shape_t
{
    ArrSt(V2Df) *points;
    Material *material;
    ...
    FPtr_draw func_draw;
};

static void i_draw_conceptual(const Shape *shape, DCtx *ctx)
{
    /* Do simple drawing */
}

static void i_draw_realistic(const Shape *shape, DCtx *ctx)
{
    /* Do complex drawing */
}

Shape *shape[N];
Shape *shape[0] = heap_new(Shape);
Shape *shape[1] = heap_new(Shape);
shape[0]->func_draw = i_draw_conceptual;
shape[1]->func_draw = i_draw_realistic;
...

for (i = 0; i < N; ++i)
    shape[i]->func_draw(shape[i], ctx);
```

4.7. Preprocesador

En nuestros estándares se hace un uso intensivo del preprocesador, sobre todo para la comprobación de tipos en tiempo de compilación. Esto ayuda a detectar errores en el código antes de ejecutar el programa (análisis estático), al contrario que el RTTI de C++ que lo hace una vez en marcha (análisis dinámico).

Uso del preprocesador para comprobar tipos.

```
#define arrst_destroy(array, func_remove, type)\
    ((void)((array) == (ArrSt(type)**)(array)),\
    FUNC_CHECK_REMOVE(func_remove, type),\
    array_destroy_imp((Array**)(array), (FPtr_remove)func_remove, (const char_t
        ↪ *)(ARRST#type)))

ArrSt(Product) *products = arrst_create(Product);
...
```

```
static void i_remove_product(Product *product)
{

}
...

/* 'products' and 'i_remove_product' will be checked at compile time */
arrst_destroy(&products, i_remove_product, Product);
```

El tipado dinámico no es necesariamente bueno. Obtiene errores estáticos en tiempo de ejecución, que realmente deberían poder detectarse en tiempo de compilación. ***Rob Pike****.*

4.8. Comentarios

Por lo general se reducirá el uso de comentarios todo lo posible. Se pondrá un comentario al inicio de cada fichero a modo de descripción general. También utilizamos una línea de comentario como separador a la hora de implementar funciones.

stream.c

```
/* Data streams. Manage connection-oriented communication */

#include "stream.h"
#include "stream.inl"
#include "bfile.h"
#include "bmem.h"
...

/*---------------------------------------------------------------------------*/

static void i_func1(void)
{
    /* Do something */
}

/*---------------------------------------------------------------------------*/

static void i_func2(void)
{
    /* Do something */
}
```

Los comentarios tipo C++ `//Comment...` *NO están permitidos, ya que generan warnings en ciertos compiladores* `gcc -std=gnu90`*.*

Otro aspecto que queda **totalmente prohibido** es la inclusión de bloques de documentación dentro del código fuente, ni siquiera en las propias cabeceras. NAppGUI utiliza `ndoc` para tareas de documentación, una utilidad que permite crear documentos html/pdf enriquecidos con imágenes, referencias cruzadas, ejemplos, etc y que utiliza sus propios archivos totalmente separados del código. Otra ventaja añadida es la limpieza que presentan las cabeceras `*.h` de todos los módulos, donde se hace muy sencillo localizar aquello que buscamos.

Los bloques de documentación NO están permitidos.

```
/* Forbidden, non used */
/*! Gets the area of the polygon.
    \param pol The polygon.
    \return The area.
*/
real32_t pol2d_areaf(const Pol2Dd *pol);
```

Ejemplo de cabecera en NAppGUI.

```
/* 2d convex polygon */

#include "geom2d.hxx"

__EXTERN_C

Pol2Df* pol2d_createf(const V2Df *points, const uint32_t n);

Pol2Df* pol2d_copyf(const Pol2Df *pol);

void pol2d_destroyf(Pol2Df **pol);

void pol2d_transformf(Pol2Df *pol, const T2Df *t2d);

const V2Df *pol2d_pointsf(const Pol2Df *pol);

uint32_t pol2d_nf(const Pol2Df *pol);

real32_t pol2d_areaf(const Pol2Df *pol);

bool_t pol2d_ccwf(const Pol2Df *pol);

bool_t pol2d_convexf(const Pol2Df *pol);

__END_C
```

Todos los comentarios en NAppGUI se realizan en lenguaje Inglés.

4.9. Entrada/Salida

La entrada/salida no forma parte del lenguaje C como tal. A medida que el lenguaje se iba extendiendo a mediados de los 70, se fueron agrupando una serie de rutinas útiles en lo que pasó a formar la **Librería Estándar de C**. NAppGUI encapsula toda su funcionalidad en *"Sewer"* (Página 151), *"Osbs"* (Página 168) o *"Core"* (Página 191) implementándola generalmente como llamadas al sistema operativo, mucho más directas y eficientes.

Uso de funciones seguras de E/S.

```
/* Do not use cstdlib in applications */
#include <stdio.h>
FILE *fp = fopen("/tmp/test.txt", "w+");
fprintf(fp, "This is testing for fprintf...\n");
fclose(fp);

/* Use NAppGUI functions */
#include "stream.h"
Stream *stm = stm_to_file("/tmp/test.txt", NULL);
stm_printf(stm, "This is testing for stm_printf...\n");
stm_close(&stm);
```

*No se recomienda el uso de la **Librería Estándar de C**. Busca la función equivalente en **Sewer**, **Osbs** o **Core**.*

4.10. Algoritmos matemáticos

NAppGUI utiliza los *C++ templates* para implementar cualquier función o algoritmo matemático. Con esto se consigue ofrecer versiones `float` y `double` de manera elegante y con sencillo mantenimiento. Las plantillas están ocultas y no se exponen en el API, para que su uso siga siendo compatible con ANSI-C90. Para más información *"Plantillas matemáticasPlantillas matemáticas"* (Página 53).

NAppGUI hace uso interno de C++98 `template<>` para implementar todo lo relativo al cálculo matemático.

Capítulo 5

Uso de C++

Los servidores web están escritos en C y, si no lo están, están escritos en Java o C++, que son derivados de C, o en Python o Ruby, que se implementan en C.

Rob Pike

La programación orientada a objetos (encapsulación, herencia y polimorfismo) es una herramienta muy poderosa para modelar cierto tipo de problemas. Sin embargo, en NAppGUI creemos que es incorrecto imponer una jerarquía de clases a nivel de SDK, ya que este es un nivel demasiado bajo. El SDK está más cerca del sistema operativo y de la máquina que de los problemas del mundo real resueltos por las aplicaciones, donde una estrategia orientada a objetos puede ser más acertada (o puede que no).

Aunque NAppGUI ha sido diseñado para crear aplicaciones en C "puro", es posible utilizar C++ o combinar ambos lenguajes. Daremos algunos consejos, portando nuestra aplicación *"¡Hola Mundo!"* (Página 23) a C++ .

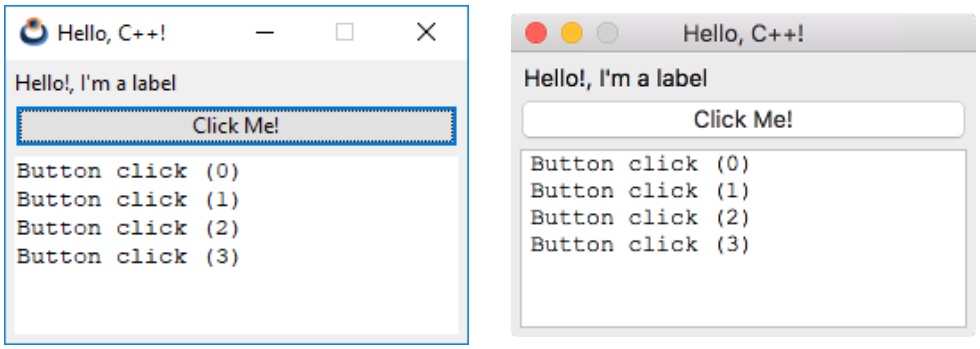


Figura 5.1: Migración de *¡Hola, mundo!* a C++.

5.1. Encapsulación

NAppGUI no impone ninguna jerarquía de clases, lo que deja al programador la libertad de realizar la encapsulación mediante sus propias clases. Evidentemente, dado que C++ incluye a C, podremos llamar a cualquier función C del SDK dentro de una función miembro. Por ejemplo, podemos encapsular la ventana principal de esta manera.

```
class MainWindow
{
public:
    MainWindow();
    ~MainWindow();

private:
    static void i_OnClose(MainWindow *window, Event *e);
    static void i_OnButton(MainWindow *window, Event *e);
    Panel *i_panel(void);

    Window *window;
    TextView *text;
    uint32_t clicks;
};
```

Como puedes ver, con respecto a la versión C, `i_panel` ya no necesita parámetros, ya que usa el puntero implícito `this` para acceder a los miembros de la clase.

5.2. Callbacks de clase

Los manejadores de evento son funciones C cuyo primer parámetro es un puntero al objeto que recibe el mensaje. Esto funciona de la misma manera usando funciones estáticas dentro de una clase de C++:

```
...
```

```
static void i_OnClose(MainWindow *window, Event *e);
...
window_OnClose(this->window, listener(this, i_OnClose, MainWindow));
...
```

Sin embargo, es posible que queramos utilizar funciones miembro como manejadores del evento, usando el puntero **this** como receptor. Para hacer esto, derivamos nuestra `MainWindow` de la interfaz `IListener` y usamos la macro `listen` en lugar de `listener()`.

```
class MainWindow : public IListener
{
...
    void i_OnClose(Event *e);
    void i_OnButton(Event *e);
...
};

void MainWindow::i_OnButton(Event *e)
{
    String *msg = str_printf("Button click (%d)\n", this->clicks);
    ...
}
...
button_OnClick(button, listen(this, MainWindow, i_OnButton));
...
```

`IListener` es una interfaz C++ que permite utilizar métodos miembro de una clase como manejadores de evento.

También es posible dirigir el evento a un objeto diferente (y de diferente clase) a la dueña control. Para ello indicamos el receptor como primer parámetro de `listen`, como vemos a continuación. La pulsación del botón de cierre será procesada en la clase `App` y no en `MainWindow`.

```
class App : public IListener
{
public:
    App();
    ~App();
    void i_OnClose(Event *e);

private:
    MainWindow *main_window;
};

class MainWindow : public IListener
{
```

```
public:
    MainWindow(App *app);
}

MainWindow::MainWindow(App *app)
{
   ...
   window_OnClose(this->window, listen(app, App, i_OnClose));
   ...
}

void App::i_OnClose(Event *e)
{
    osapp_finish();
}
```

Podemos establecer como receptor de eventos, cualquier objeto que implemente la interfaz `IListener`.

5.3. Combinar módulos C y C++

Un proyecto C/C++ selecciona el compilador en función de la extensión del archivo. Para ***.c** se utilizará el compilador C y para ***.cpp** el compilador C++. El mismo proyecto puede combinar módulos en ambos lenguajes si consideramos lo siguiente.

5.3.1. Uso de C desde C++

No hay problema si las declaraciones de funciones la cabecera de C están entre las macros: `__EXTERN_C` y `__END_C`.

```
__EXTERN_C

real32_t mymaths_add(const real32_t a, const real32_t b);

real32_t mymaths_sub(const real32_t a, const real32_t b);

__END_C
```

`__EXTERN_C` y `__END_C` son alias para `extern "C"{}`. Esto le dice al compilador de C++ que no use name mangling[a] con funciones en C.

[a] https://en.wikipedia.org/wiki/Name_mangling

5.3.2. Uso de C++ desde C

C no entiende la palabra clave `class` y dará un error de compilación al incluir cabeceras de C++. Es necesario definir una interfaz en C sobre código C++.

mywindow.h

```
__EXTERN_C

typedef struct _mywin_t MyWindow;

MyWindow *mywindow_create();

void mywindow_move(MyWindow *window, const real32_t x, const real32_t y);

__END_C
```

mywindow.cpp

```
class MainWindow
{
public:
    MainWindow();
    void move(const real32_t x, const real32_t y);
};

MyWindow *mywindow_create()
{
    return (MyWindow*)new MainWindow();
}

void mywindow_move(MyWindow *window, const real32_t x, const real32_t y)
{
    ((MainWindow*)window)->move(x, y);
}
```

5.4. Sobrecarga de new y delete

C++ utiliza los operadores `new` y `delete` para crear instancias dinámicas de objetos. Podemos hacer que las reservas se realicen a través de `Heap`, el gestor de *"Heap - Gestor de memoria"* (Página 192) que incorpora NAppGUI, con el fin de optimizar C++ y controlar los *Memory Leaks*.

```
class MainWindow : public IListener
{
    ...
    void *operator new(size_t size)
    {
        return (void*)heap_malloc((uint32_t)size, "MainWindow");
```

```
    }

    void operator delete(void *ptr, size_t size)
    {
        heap_free((byte_t**)&ptr, (uint32_t)size, "MainWindow");
    }
    ...
};
```

5.5. Hola C++ completo

Listado 5.1: demo/hellocpp/main.cpp

```
/* NAppGUI C++ Hello World */

#include <nappgui.h>

class App;

class MainWindow : public IListener
{
public:
    MainWindow(App *app);
    ~MainWindow();

    void *operator new(size_t size) { return (void*)heap_malloc((uint32_t)size,
        ↪  "MainWindow"); }
    void operator delete(void *ptr, size_t size) { heap_free((byte_t**)&ptr, (
        ↪ uint32_t)size, "MainWindow"); }

private:
    void i_OnButton(Event *e);
    Panel *i_panel(void);

    Window *window;
    TextView *text;
    uint32_t clicks;
};

/*---------------------------------------------------------------------------*/

class App : public IListener
{
public:
    App();
    ~App();
    void i_OnClose(Event *e);
    void *operator new(size_t size) { return (void*)heap_malloc((uint32_t)size,
        ↪  "App"); }
```

```
    void operator delete(void *ptr, size_t size) { heap_free((byte_t**)&ptr, (
        ↪ uint32_t)size, "App"); }

private:
    MainWindow *main_window;
};

/*--------------------------------------------------------------------------*/

void MainWindow::i_OnButton(Event *e)
{
    String *msg = str_printf("Button click (%d)\n", this->clicks);
    textview_writef(this->text, tc(msg));
    str_destroy(&msg);
    this->clicks += 1;
    unref(e);
}

/*--------------------------------------------------------------------------*/

Panel *MainWindow::i_panel(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 3);
    Label *label = label_create();
    Button *button = button_push();
    TextView *textv = textview_create();
    this->text = textv;
    label_text(label, "Hello!, I'm a label");
    button_text(button, "Click Me!");
    button_OnClick(button, IListen(this, MainWindow, i_OnButton));
    layout_label(layout, label, 0, 0);
    layout_button(layout, button, 0, 1);
    layout_textview(layout, textv, 0, 2);
    layout_hsize(layout, 0, 250);
    layout_vsize(layout, 2, 100);
    layout_margin(layout, 5);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    panel_layout(panel, layout);
    return panel;
}

/*--------------------------------------------------------------------------*/

void App::i_OnClose(Event *e)
{
    osapp_finish();
    unref(e);
}
```

```
/*-----------------------------------------------------------------------------*/

MainWindow::MainWindow(App *app)
{
    Panel *panel = i_panel();
    this->window = window_create(ekWINDOW_STD);
    this->clicks = 0;
    window_panel(this->window, panel);
    window_title(this->window, "Hello, C++!");
    window_origin(this->window, v2df(500, 200));
    window_OnClose(this->window, IListen(app, App, i_OnClose));
    window_show(this->window);
}

/*-----------------------------------------------------------------------------*/

MainWindow::~MainWindow()
{
    window_destroy(&this->window);
}

/*-----------------------------------------------------------------------------*/

App::App(void)
{
    this->main_window = new MainWindow(this);
}

/*-----------------------------------------------------------------------------*/

App::~App()
{
    delete this->main_window;
}

/*-----------------------------------------------------------------------------*/

static App *i_create(void)
{
    return new App();
}

/*-----------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    delete *app;
    *app = NULL;
}

/*-----------------------------------------------------------------------------*/
```

```
#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

5.6. Plantillas matemáticas

En NAppGUI existen dos versiones para todas las funciones y tipos matemáticos (Listado 5.2): `float (real32_t)` y `double (real64_t)`. Podemos utilizar unos u otros según convenga en cada caso.

Listado 5.2: Cabecera `bmath.h` (parcial).

```
/* Math funcions */

#include "osbs.hxx"

__EXTERN_C

real32_t bmath_cosf(const real32_t angle);

real64_t bmath_cosd(const real64_t angle);

real32_t bmath_sinf(const real32_t angle);

real64_t bmath_sind(const real64_t angle);

extern const real32_t kBMATH_PIf;
extern const real64_t kBMATH_PId;
extern const real32_t kBMATH_SQRT2f;
extern const real64_t kBMATH_SQRT2d;

__END_C
```

*Todas las funciones y tipos en simple precisión acaban con el sufijo **"f"** y las de doble precisión en **"d"**.*

Cuando implementamos funciones geométricas o algebraicas más complejas, no es fácil tener claro de antemano cual es la precisión correcta. Ante la duda siempre podemos optar por utilizar `double`, pero esto tendrá un impacto en el rendimiento, sobre todo por el uso del ancho de banda de memoria. Pensemos en el caso de mallas 3D con miles de vértices. Sería estupendo disponer de ambas versiones y poder utilizar una u otra según cada caso concreto.

Por desgracia el lenguaje C "puro" no permite programar con tipos genéricos, al margen de utilizar horribles e interminables macros. Deberemos implementar ambas versiones (`float` y `double`), con el coste de mantenimiento asociado. C++ resuelve el problema

gracias a las plantillas (`template<>`). La contrapartida es que, normalmente, debemos "abrir" la implementación e incluirla en la cabecera `.h`, ya que el compilador no sabe generar el código máquina hasta que la plantilla se instancia con un tipo de dato concreto. Esto choca de frente con nuestros *"EstándaresEstándares"* (Página 58), sobre todo en la parte relativa a la encapsulación de información. A continuación veremos como utilizar las `templates` de C++ para obtener lo mejor de ambos casos: Programación genérica, ocultar implementaciones y mantener las cabeceras "limpias".

Al igual que existe una cabecera `*.h` para cada módulo matemático, existe una contrapartida `*.hpp` utilizable únicamente desde módulos C++ (Listado 5.3).

Listado 5.3: Cabecera `bmath.hpp` (parcial).

```
/* Math funcions */

#include "osbs.hxx"

template<typename real>
struct BMath
{
    static real(*cos)(const real angle);

    static real(*sin)(const real angle);

    static const real kPI;
    static const real kSQRT2;
};
```

Estas plantillas contienen punteros a función, cuyas implementaciónes están ocultas en `bmath.cpp`. En (Listado 5.4) tenemos un ejemplo de uso.

Listado 5.4: Implementación de un algoritmo genérico.

```
#include "bmath.hpp"

template<typename real>
static void i_circle(const real r, const uint32_t n, V2D<real> *v)
{
    real a = 0, s = (2 * BMath<real>::kPI) / (real)n;
    for (uint32_t i = 0; i < n; ++i, a += s)
    {
        v[i].x = r * BMath<real>::cos(a);
        v[i].y = r * BMath<real>::sin(a);
    }
}
```

Este algoritmo está implementado dentro de un módulo C++ (Listado 5.5), pero queremos poder llamarlo desde otros módulos, tanto C como C++. Para ello definiremos los dos tipos de cabecera: `*.h` (Listado 5.6) y `*.hpp` (Listado 5.7).

Listado 5.5: mymath.cpp. Implementación.

```
#include "mymath.h"
#include "mymath.hpp"
#include "bmath.hpp"

template<typename real>
static void i_circle(const real r, const uint32_t n, V2D<real> *v)
{
    real a = 0, s = (2 * BMath<real>::kPI) / (real)n;
    for (uint32_t i = 0; i < n; ++i, a += s)
    {
        v[i].x = r * BMath<real>::cos(a);
        v[i].y = r * BMath<real>::sin(a);
    }
}

void mymath_circlef(const real32_t r, const uint32_t n, V2Df *v)
{
    i_circle<real32_t>(r, n, (V2D<real32_t>*)v);
}

void mymath_circled(const real64_t r, const uint64_t n, V2Dd *v)
{
    i_circle<real64_t>(r, n, (V2D<real64_t>*)v);
}

template<>
void(*MyMath<real32_t>::circle)(const real32_t, const uint32_t, V2D<real32_t>*)
    ↪  = i_circle<real32_t>;

template<>
void(*MyMath<real64_t>::circle)(const real64_t, const uint32_t, V2D<real64_t>*)
    ↪  = i_circle<real64_t>;
```

Listado 5.6: mymath.h. Cabecera C.

```
#include "geom2d.hxx"

__EXTERN_C

void mymath_circlef(const real32_t r, const uint32_t n, V2Df *v);

void mymath_circled(const real64_t r, const uint64_t n, V2Dd *v);

__END_C
```

Listado 5.7: mymath.hpp. Cabecera C++.

```
#include "v2d.hpp"

template<typename real>
```

```
struct MyMath
{
    void (*circle)(const real r, const uint32_t n, V2D<real> *v);
};
```

Ahora podemos utilizar nuestra librería matemática en C y C++, tanto en `float` como en `double` precisión (Listado 5.8).

Listado 5.8: Uso de `mymaths` en algoritmos genéricos C++.

```
#include "mymath.hpp"
#include "t2d.hpp"

template<typename real>
static void i_ellipse(const real r1, const real r2, const uint32_t n, V2D<real>
    ↪  *v)
{
    T2D<real> transform;
    T2D<real>::scale(&transform, r1, r2);

    MyMath<real>::circle(1, n, v);

    for (uint32_t i = 0; i < n; ++i)
        T2D<real>::vmult(&transform, &v[i]);
}
```

Capítulo 6

Gestión de errores

Siempre hay un error más que arreglar.

Ellen Ullman

Desarrollar software de cierto tamaño y complejidad puede convertirse en una tarea infernal, si no adoptamos medidas concretas para la prevención y rápida localización de los *bugs* de programación. Hablaremos a continuación de algunas estrategias que hemos utilizado en el desarrollo de NAppGUI y que puedes aplicar en tus propios proyectos.

6.1. Pruebas exhaustivas

Asegurar que nuestro software está libre de errores es tan "sencillo" como realizar una prueba para todos y cada uno de los casos a los que se va a enfrentar el programa (Figura 6.1).

Ya desde ejemplos teóricos triviales, vemos que estamos tratando con un problema exponencial (Figura 6.2), que desbordará los recursos de cualquier sistema con relativamente

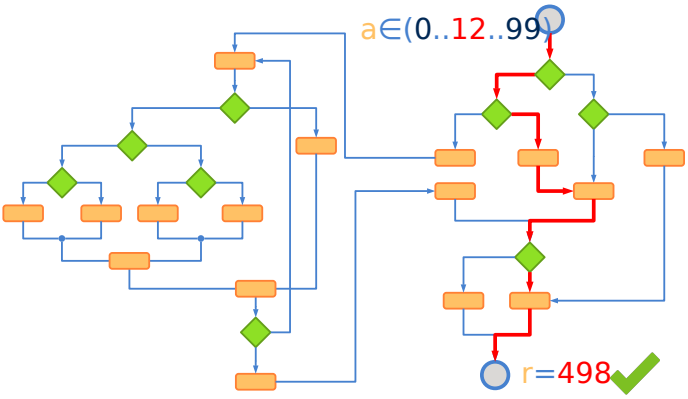


Figura 6.1: En las *pruebas exhaustivas* se utilizan todas las posibles combinaciones de los datos de entrada.

pocas variables de entrada. Por tanto, podemos intuir que será **imposible garantizar** que nuestro software esté libre de errores ya que no será viable reproducir todos sus casos de uso. Sin embargo, podemos definir una estrategia que nos ayude a minimizar el impacto que estos tendrán en el producto final, detectándolos y corrigiéndolos lo antes posible.

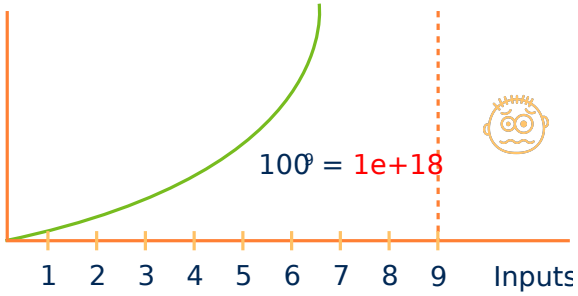


Figura 6.2: Con solo 9 variables de entrada (en rango 0..99) los recursos de cómputo se desbordarán.

6.2. Análisis estático

El análisis estático es aquel que se lleva a cabo **antes de ejecutar el programa** y consta de dos partes: El uso de **estándares** donde se aplican reglas y controles de calidad durante la propia escritura del código. Y los **avisos del compilador** que nos ayudarán a localizar potenciales errores en tiempo de compilación.

6.2.1. Estándares

El uso de estándares, entendidos como reglas que seguimos al programar, es algo esencial a la hora de mantener unos niveles mínimos de calidad en nuestros proyectos (Figura 6.3). De no aplicarlos, un programa de cierto tamaño se tornará anárquico, ilegible, difícil de mantener y complicado de entender. En este escenario será fácil añadir nuevos errores a medida que manipulamos el código fuente.

Figura 6.3: El uso de estándares reducirán la probabilidad de *bugs*.

En realidad, es complicado diferenciar entre buenos y malos estándares, ya que dependerán del tipo de proyecto, lenguajes de programación, filosofía de la empresa y objetivos a priorizar. Podemos verlos como una *Guía de Estilo* que va evolucionando con el tiempo de la mano de la experiencia. Lo verdaderamente importante es concienciarnos de su utilidad, definirlos y aplicarlos. Por ejemplo, si decidimos nombrar variables con identificadores descriptivos en Inglés y guión bajo (`product_code`), todo nuestro código debería cumplir esta regla sin excepción. Vamos a ver algunos de los estándares que aplicamos dentro de NAppGUI. No son los mejores ni tienen porque adaptarse a todos los casos. Tan solo son los nuestros:

- Utilizar un subconjunto reducido del lenguaje, como hemos visto en *"Uso de C"* (Página 31). Por ejemplo, expresiones del tipo `*((int*)block + i++)= i+1`, están totalmente prohibidas. Son perfectamente válidas en C pero poco legibles y confusas. Algunos programadores piensan que el código críptico y compacto es mucho más mantenible, pero creemos que están equivocados.

- Prohibidos los comentarios, salvo en contadas ocasiones y muy justificadas. Si algo precisa de un comentario, reescríbelo. Un comentario que contradiga mínimamente al código que pretende clarificar produce más confusión que ayuda. Y es muy sencillo que queden obsoletos.

- Interfaces públicas reducidas y limpias. Los ficheros de cabecera (`*.h`) suponen un gran nivel de abstracción ya que reducen las conexiones entre componentes software (Figura 6.4). Permiten condensar, a modo de índice, cientos o miles de líneas de código en apenas quince o veinte funciones públicas. Está completamente prohibido incluir definiciones de tipos (irán en el `*.hxx`), comentarios (por supuesto) y bloques de documentación en archivos `.h`.

- Objetos opacos. Las definiciones de objetos (`struct _object_t`) se realizarán dentro de los ficheros de implementación (`*.c`) y nunca en el `*.h`. Los objetos se manipu-

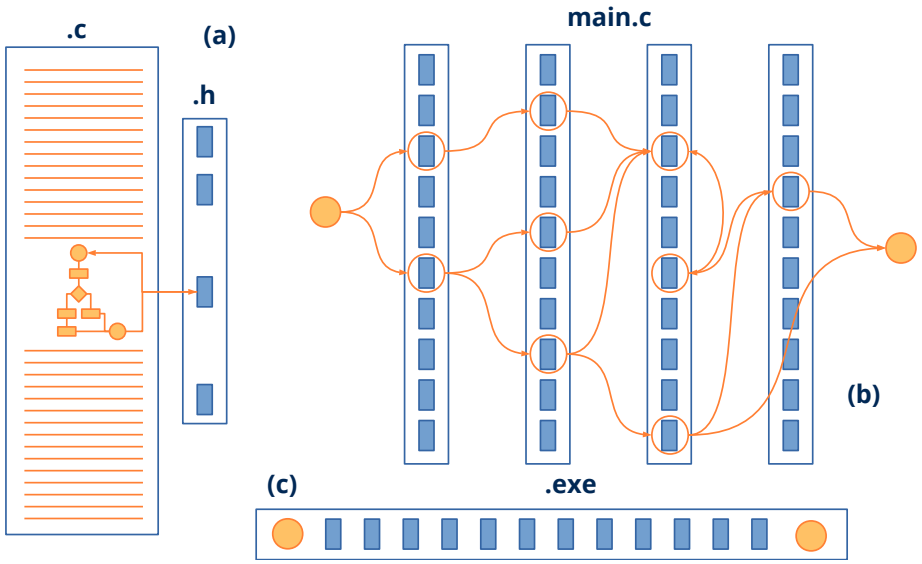


Figura 6.4: Las cabeceras `*.h` suponen un gran nivel de abstracción ocultando la complejidad de la solución **(a)**. Facilitan un desarrollo horizontal, basado en el problema, frente al aprendizaje vertical basado en APIs **(b)**. Ayudan al enlazador a reducir el tamaño del ejecutable **(c)**.

larán con funciones públicas que aceptan punteros a los mismos, ocultando siempre los campos que los componen. Este punto, junto con el anterior de las interfaces, delimita perfectamente las barreras entre módulos, marcando claramente cuando acaba un problema y empieza otro.

Las dos primeras reglas ayudan a reducir la complejidad interna de un módulo haciéndolo lo mas legible y menos críptico posible. Podríamos enriquecerlas con otras sobre indentación, estilo, nombrado de variables, etc. Nosotros seguimos con mas o menos rigor los consejos del genial libro *The Practice of Programming* (Figura 6.5).

6.2.2. Avisos del compilador

El compilador es nuestro gran aliado a la hora de examinar el código en busca de posibles fallos (Figura 6.6). Activar el mayor nivel posible de *warnings* es esencial para reducir errores derivados de la conversión de tipos, variables sin inicializar, código no alcanzable, etc. Todos los proyectos creados con NAppGUI se activará el mayor nivel de avisos posible, equivalente a `-Wall -Wpedantic` en todas las plataformas (Figura 6.7).

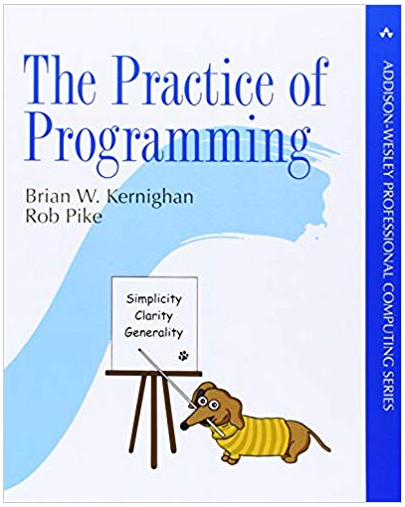


Figura 6.5: *The Practice of Programming* de Brian W. Kernighan y Rob Pike es una buena fuente de inspiración para definir tu propio estilo de programación.

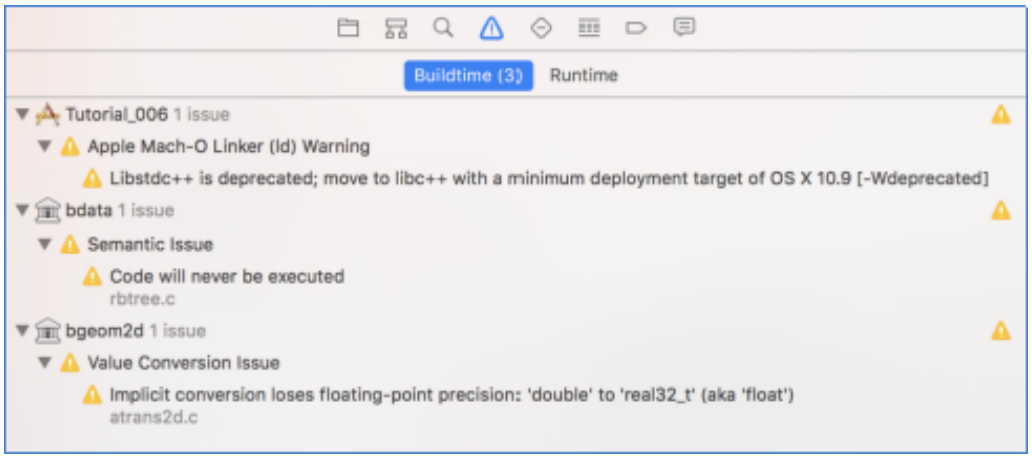


Figura 6.6: Corregir todos los *warnings* del compilador debe ser una prioridad.

6.3. Análisis dinámico

El análisis dinámico se realiza una vez el programa está en ejecución. Aquí nuestra principal arma son las auto-validaciones, implementadas como sentencias *"Asserts"* (Página 155). Los asserts son comprobaciones distribuidas a lo largo y ancho del código fuente, que son evaluadas en tiempo de ejecución cada vez que el programa pasa por ellas. Si una sentencia se resuelve como `FALSE`, el proceso se parará y se mostrará una ventana informativa (Figura 6.8).

```
void layout_set_row_margin(Layout *layout, const uint32_t row, const real32_t
    ↪ margin)
{
    cassert_no_null(layout);
```

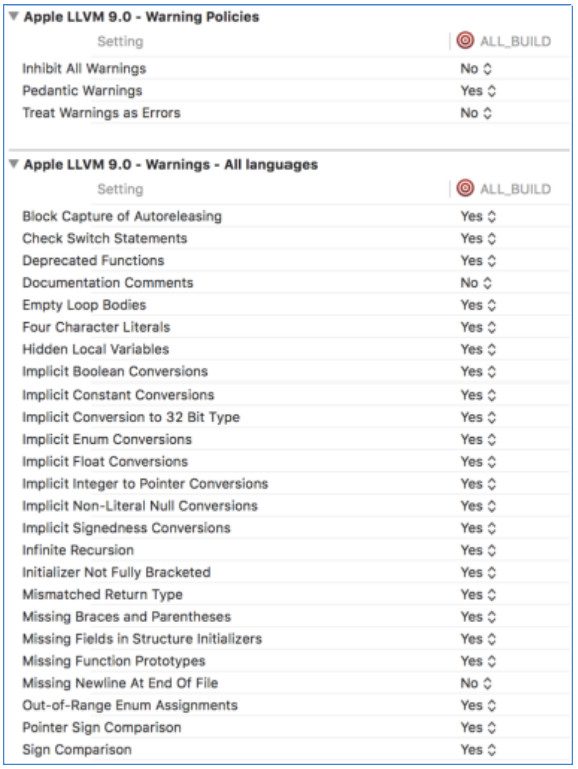


Figura 6.7: NAppGUI activa el mayor nivel de *warnings* posible.

```
    cassert_msg(row < layout->num_rows, "'row' out of range");
    ...
}
```

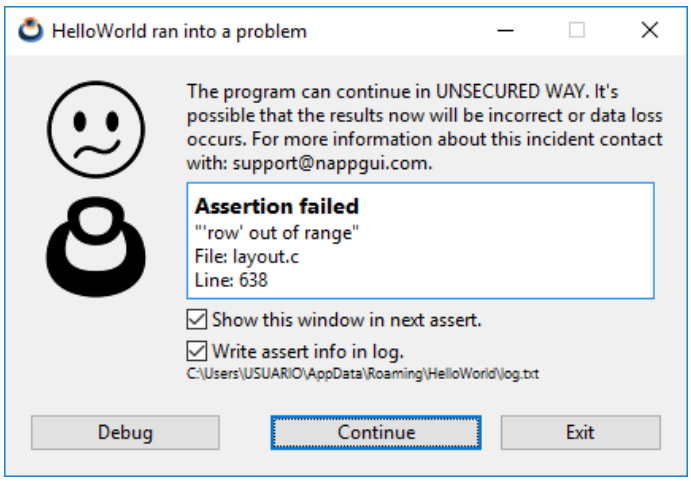


Figura 6.8: Ventana mostrada tras la activación de un assert.

*Es posible también redirigir las sentencias **assert** a la salida estándar o al fichero de **Log**.*

6.3.1. Deshabilitar Asserts

Dentro del código del SDK de NAppGUI se hayan repartidas más de 5000 sentencias asserts, localizadas en puntos estratégicos, que constantemente evalúan la coherencia e integridad del software. Evidentemente, este número irá creciendo tras cada revisión, a medida que vaya integrándose mayor funcionalidad. Esto convierte al SDK en un auténtico campo de minas, donde cualquier error en el uso de las funciones del API será automáticamente notificado al programador. Dependiendo de la configuración que estemos utilizando, los asserts serán activados o desactivados:

- `Debug`: Las sentencias *assert* están activadas.
- `Release`: Las sentencias *assert* están desactivadas.
- `ReleaseWithAssert`: Como su nombre indica, activa todas los optimizaciones de Release, pero deja activadas las sentencias *assert*.

6.3.2. Depurando el programa

Cuando se activa un assert, el programa se detiene justo en el punto de la comprobación, mostrando la ventana de confirmación del assert. Si presionamos el botón `[Debug]`, accederemos al *call stack* (Figura 6.9), que es la actual pila de llamadas a función, desde el propio `main()` hasta el punto de parada actual *"Segmento StackSegmento Stack"* (Página 164). Navegando por la pila podemos chequear los valores de variables y objetos en cualquier nivel de llamada. Esto nos ayudará a identificar el origen del error, ya que la causa puede estar algunos niveles por debajo de la detección.

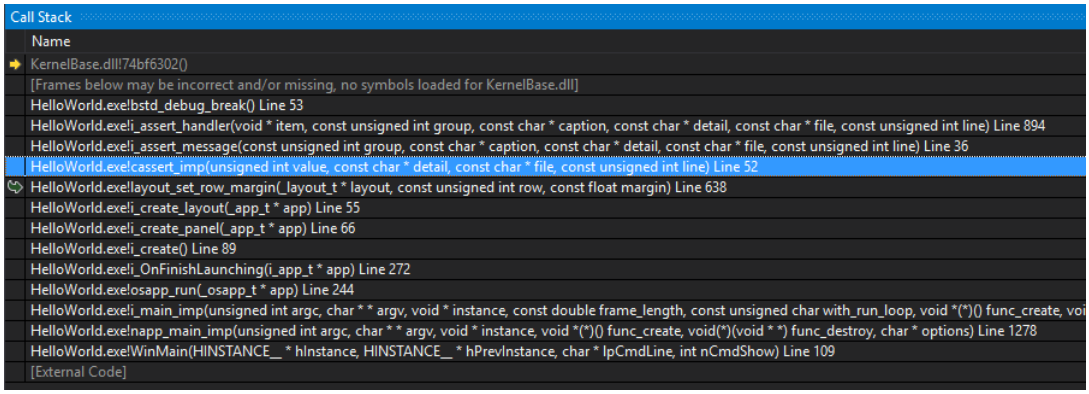


Figura 6.9: *Call stack* mientras depuramos el assert del ejemplo anterior.

6.3.3. Registro de fallos

Un *"Log"* (Página 186) de ejecución es un archivo donde el programa va volcando información acerca de su estado o anomalías detectadas. Puede ser muy útil para conocer la causa de un fallo cuando el software ya ha sido distribuido y no es posible depurarlo. NAppGUI automáticamente crea un archivo de log para cada aplicación ubicado en el directorio de datos de la aplicación `APP_DATA\APP_NAME\log.txt`, por ejemplo `C:\Users\USUARIO\AppData\Roaming\HelloWorld\log.txt`.

```
[15:42:29] Starting log for 'HelloWorld'
[15:42:29] TextView created: [0x6FFC7A30]
[15:42:32] Assertion failed (c:\\nappgui_1_0\\src\\gui\\layout.c:638): "'row'
    ↪ out of range"
[15:42:32] Assertion failed (c:\\nappgui_1_0\\src\\core\\array.c:512): "Array
    ↪ invalid index"
[15:42:34] You have an execution log in: 'C:\\Users\\USUARIO\\AppData\\Roaming
    ↪ \\HelloWorld\\log.txt'
```

Como puedes ver, los asserts se redirigen automáticamente al archivo *log*. Es posible deshabilitar esta escritura desmarcando el check `'Write assert info in log'` de la ventana de información. También puedes añadir tus propios mensajes utilizando el método `log_printf`.

```
log_printf("TextView created: [0x%X]", view);
```

6.3.4. Auditor de memoria

El gestor de memoria de NAppGUI *"Heap - Gestor de memoria"* (Página 192), tiene asociado un auditor que comprueba que no haya fugas de memoria (*leaks*) tras cada ejecución de cada aplicación que utilice el SDK. Esto supone una gran ventaja con respecto al uso de utilidades externas, ya que las comprobaciones de memoria dinámica se están llevando a cabo **siempre** y no en fases aisladas del desarrollo.

```
[18:57:33] [OK] Heap Memory Staticstics
[18:57:33] ============================
[18:57:33] Total a/dellocations: 652962, 652962
[18:57:33] Total bytes a/dellocated: 18085221250, 18085221250
[18:57:33] Max bytes allocated: 238229150
[18:57:33] Effective reallocations: (0/1169761)
[18:57:33] Real allocations: 32776 pages of 65536 bytes
[18:57:33]                   13271 pages greater than 65536 bytes
[18:57:33] ============================
[18:57:33] Config: Debug
[18:57:33] You have an execution log in: 'C:\Users\USUARIO\AppData\Roaming\
    ↪ EuroPlane\log.txt'code.
```

Capítulo 7

Generar binarios NAppGUI

En *"Inicio rápido"* (Página 5) ya vimos como descargar, compilar y ejecutar los ejemplos a partir del código fuente. En este capítulo y el siguiente profundizaremos en el proceso de compilación y en la portabilidad entre plataformas. Todo el *build system* de NAppGUI se centra en un único script `src/CMakeLists.txt` (Listado 7.1). En él se definirá una solución con varios proyectos relacionados, así como sus dependencias.

Listado 7.1: `src/CMakeLists.txt`

```
cmake_minimum_required(VERSION 2.8.12)
project(NAppGUI)

# NAppGUI Build Scripts
get_filename_component(ROOT_PATH ${CMAKE_CURRENT_SOURCE_DIR} PATH)
include(${ROOT_PATH}/prj/CMakeNAppGUI.cmake)

# Static libraries
staticLib("sewer" "sewer" "" NRC_NONE)
staticLib("osbs" "osbs" "sewer" NRC_NONE)
staticLib("core" "core" "osbs" NRC_NONE)
staticLib("geom2d" "geom2d" "core" NRC_NONE)
staticLib("draw2d" "draw2d" "geom2d" NRC_NONE)
staticLib("osgui" "osgui" "draw2d" NRC_NONE)
staticLib("gui" "gui" "draw2d" NRC_EMBEDDED)
staticLib("inet" "inet" "core" NRC_NONE)
staticLib("osapp" "osapp" "osgui;gui" NRC_NONE)
```

```
# Executables
desktopApp("Fractals" "demo/fractals" "osapp" NRC_EMBEDDED)
desktopApp("HelloWorld" "demo/hello" "osapp" NRC_EMBEDDED)
desktopApp("HelloCpp" "demo/hellocpp" "osapp" NRC_EMBEDDED)
...

generateSolution()
```

7.1. Generar librerías estáticas

Por defecto, el `CMakeLists.txt` creará la versión estática de las librerías de NAppGUI. Si quieres utilizar NAppGUI de forma externa en tus proyectos, tan solo tienes que seguir los siguientes pasos:

```
git clone --depth 1 https://github.com/frang75/nappgui_src.git
cd nappgui_src

// Windows
cmake -S ./src -B ./build
cmake --build ./build --config Debug
cmake --install ./build --prefix ./install --config Debug

// macOS
cmake -G Xcode -S ./src -B ./build
cmake --build ./build --config Debug
cmake --install ./build --prefix ./install --config Debug

// Linux
cmake -S ./src -B ./build -DCMAKE_BUILD_CONFIG=Debug
cmake --build ./build -j 4
cmake --install ./build --prefix ./install --config Debug
```

En la carpeta `install` tendrás los binarios y las cabeceras:

```
install
  |
  +-- inc
  |    |
  |    +-- core
  |   ...   |
  |         +-- array.h
  |        ...
  +-- lib
  |    |
  |    +-- v143_x64
  |         |
  |         +-- Debug
  |              |
  |              +-- core.lib
```

```
|              ...
+-- bin
     |
     +-- v143_x64
          |
          +-- Debug
               |
               +-- Bode.exe
              ...
```

- En `/install/inc` encontrarás los ficheros de cabecera de cada librería.
- En `/install/lib` encontrarás las librerías estáticas (.lib, .a).
- En `/install/bin` encontrarás las ejecutables de ejemplo.
- `v143_x64` identifica al compilador y arquitectura. En *"Compiladores e IDEs"* (Página 73) tienes más información acerca de compiladores, plataformas y arquitecturas soportadas.
- `Debug` es una de las tres configuraciones posibles: `Debug, Release, ReleaseWithAssert` . *"ConfiguracionesConfiguraciones"* (Página 97)

Si no quieres compilar las aplicaciones de ejemplo, elimina las líneas ***desktopApp*** *del script.*

7.2. Generar librerías dinámicas

Para generar las versiones de enlace dinámico (.dll, .so, .dylib) de NAppGUI, edita `CMakeLists.txt`, sustituyendo los comandos `staticLib` por `dynamicLib`. Una vez hecho esto, compila e instala utilizando `cmake` de la misma forma que en el caso estático.

```
cmake_minimum_required(VERSION 2.8.12)
project(NAppGUI)

# NAppGUI Build Scripts
get_filename_component(ROOT_PATH ${CMAKE_CURRENT_SOURCE_DIR} PATH)
include(${ROOT_PATH}/prj/CMakeNAppGUI.cmake)

# Dynamic libraries
dynamicLib("sewer" "sewer" "" NRC_NONE)
dynamicLib("osbs" "osbs" "sewer" NRC_NONE)
dynamicLib("core" "core" "osbs" NRC_NONE)
dynamicLib("geom2d" "geom2d" "core" NRC_NONE)
dynamicLib("draw2d" "draw2d" "geom2d" NRC_NONE)
dynamicLib("osgui" "osgui" "draw2d" NRC_NONE)
dynamicLib("gui" "gui" "draw2d" NRC_EMBEDDED)
dynamicLib("inet" "inet" "core" NRC_NONE)
```

```
dynamicLib("osapp" "osapp" "osgui;gui" NRC_NONE)

# Executables
desktopApp("Fractals" "demo/fractals" "osapp" NRC_EMBEDDED)
desktopApp("HelloWorld" "demo/hello" "osapp" NRC_EMBEDDED)
desktopApp("HelloCpp" "demo/hellocpp" "osapp" NRC_EMBEDDED)
...

generateSolution()
```

Tras la instalación:

- En `/install/inc` encontrarás los ficheros de cabecera de cada librería.
- En `/install/lib` se guardarán las librerías de importación de símbolos de las `.dll` (`.lib`), solo en Windows.
- En `/install/bin` se guardarán las librerías dinámicas `.dll`, `.so`, `.dylib` junto con los ejecutables de ejemplo.

```
11-Dec-22  19:42    <DIR>          .
11-Dec-22  19:42    <DIR>          ..
11-Dec-22  19:42    <DIR>          res
11-Dec-22  19:42           217,088 osgui.dll
11-Dec-22  19:42            93,184 casino.dll
11-Dec-22  19:41           119,808 osbs.dll
11-Dec-22  19:42           241,152 core.dll
11-Dec-22  19:42           100,864 osapp.dll
11-Dec-22  19:42           118,784 inet.dll
11-Dec-22  19:42           222,208 draw2d.dll
11-Dec-22  19:42           250,880 gui.dll
11-Dec-22  19:42           329,728 geom2d.dll
11-Dec-22  19:41           229,376 sewer.dll
11-Dec-22  19:42           462,336 DrawImg.exe
11-Dec-22  19:42           188,416 DrawHello.exe
11-Dec-22  19:42           129,024 DrawBig.exe
11-Dec-22  19:42           483,840 GuiHello.exe
11-Dec-22  19:42           138,240 HelloCpp.exe
11-Dec-22  19:42           123,904 HelloWorld.exe
11-Dec-22  19:42           132,096 Die.exe
11-Dec-22  19:42           124,928 Dice.exe
11-Dec-22  19:42           152,576 Col2dHello.exe
11-Dec-22  19:42           126,464 Bricks.exe
11-Dec-22  19:42           153,088 Products.exe
11-Dec-22  19:42           159,744 Bode.exe
11-Dec-22  19:42           127,488 Fractals.exe
11-Dec-22  19:42           128,000 UrlImg.exe
```

Si vas a utilizar estas librerías en proyectos de terceros, no generados mediante `CMakeLists.txt`, deberás previamente definir estas macros con el fin de que la importación de símbolos

se realice correctamente.

```
#define OSAPP_IMPORT
#define OSGUI_IMPORT
#define DRAW2D_IMPORT
#define GEOM2D_IMPORT
#define CORE_IMPORT
#define OSBS_IMPORT
#define SEWER_IMPORT
#define GUI_IMPORT
#define INET_IMPORT
```

7.3. Más sobre CMakeLists.txt

NAppGUI simplifica el uso de CMake, proporcionando funciones de alto nivel, ubicadas en la carpeta `/prj` de la distribución. El `CMakeLists.txt` define una solución donde coexisten diferentes librerías y ejecutables relacionados a través de dependencias. Tras ser procesado por CMake este script creará, en la carpeta `/build`, los proyectos de compilación de cada plataforma (VisualStudio, Xcode, Make). Dentro del script trabajaremos con cuatro comandos esencialmente:

- `staticLib`: Para crear *"Librerías estáticasLibrerías estáticas"* (Página 109).
- `dynamicLib`: Para crear *"Librerías dinámicasLibrerías dinámicas"* (Página 116).
- `desktopApp`: Para crear *"Aplicaciones de escritorioAplicaciones de escritorio"* (Página 101).
- `commandApp`: Para crear *"Aplicaciones por línea de comandosAplicaciones por línea de comandos"* (Página 105).

En el ejemplo siguiente definimos una solución que contiene una librería dinámica y dos aplicaciones, una de escritorio y otra por línea de comandos. Ambas hacen uso (dependen) de dicha librería dinámica y de NAppGUI (librerías estáticas) para la interfaz gráfica y el soporte multiplataforma.

CMakeLists.txt

```
#-------------------------------------------------------------------------------
# CMake build script
# Copyright (C) 2023 - PhysicsLab
# See LICENSE.txt for details
#-------------------------------------------------------------------------------
cmake_minimum_required(VERSION 2.8.12)
project(PhysicsSimulator)

# NAppGUI Build Scripts
get_filename_component(ROOT_PATH ${CMAKE_CURRENT_SOURCE_DIR} PATH)
include(${ROOT_PATH}/prj/CMakeNAppGUI.cmake)
```

```
# NAppGUI static libraries
staticLib("sewer" "sewer" "" NRC_NONE)
staticLib("osbs" "osbs" "sewer" NRC_NONE)
staticLib("core" "core" "osbs" NRC_NONE)
staticLib("geom2d" "geom2d" "core" NRC_NONE)
staticLib("draw2d" "draw2d" "geom2d" NRC_NONE)
staticLib("osgui" "osgui" "draw2d" NRC_NONE)
staticLib("gui" "gui" "draw2d" NRC_EMBEDDED)
staticLib("osapp" "osapp" "osgui;gui" NRC_NONE)
staticLib("inet" "inet" "core" NRC_NONE)

# User dynamic library
dynamicLib("physics" "physics" "geom2d" NRC_NONE)

# Exes
desktopApp("PhysicsSim" "phsim" "osapp;physics" NRC_EMBEDDED)
commandApp("PhysicsTest" "phtest" "core;physics" NRC_NONE)

generateSolution()
```

- Línea 6: Versión mínima requerida de CMake.
- Línea 7: Nombre del proyecto o solución.
- Líneas 10-11: Incluye los scripts CMake de NAppGUI, ubicados en `/prj`.
- Líneas 14-22: Generan las librerías estáticas que componen NAppGUI.
- Línea 25: Genera una librería dinámica con funciones propias del usuario.
- Línea 28: Genera una aplicación de escritorio.
- Línea 29: Genera una aplicación por línea de comandos.
- Línea 31: Procesa la solución. Este comando deberá ser el último del script.

7.4. ¿Por qué nueve librerías independientes?

NAppGUI proporciona un soporte completo multiplataforma a varios niveles. No es necesario crear una aplicación con interfaz gráfica para aprovechar las ventajas que nos ofrece en cuanto a portabilidad del código. Podemos desarrollar potentes aplicaciones *back-end* por línea de comandos orientadas a servidores. En función del nivel de asistencia que requiera cada proyecto, podemos optar por enlazar estas librerías. Más información en *"NAppGUI API"* (Página 147).

- *"Sewer"* (Página 151): Tipos básicos, aserciones, Unicode, funciones matemáticas, wrapper sobre la librería estándar de C.

- *"Osbs"* (Página 168): Servicios del sistema operativo. API portable sobre archivos, directorios, procesos, hebras, memoria, etc.
- *"Core"* (Página 191): Utilidades no gráficas de uso común. Auditor de memoria, estructuras de datos, strings, streams, expresiones regulares, recursos, etc.
- *"Geom2D"* (Página 241): Geometría 2D. Transformaciones, vectores, polígonos, colisiones, etc.
- *"Draw2D"* (Página 262): API portable de dibujo vectorial, imágenes y fuentes. Se puede utilizar en **aplicaciones por línea de comandos**, ya que es posible dibujar en memoria y exportar a un archivo o transmitir por la red.
- **osgui**: Acceso de bajo a nivel a los elementos de interfaz de usuario (widgets o controles) de cada sistema operativo. No está documentada y no se recomienda utilizarla directamente.
- *"Gui"* (Página 303): Compositor de interfaces de usuario. Utiliza `osgui` para renderizar.
- *"OSApp"* (Página 373): Implementa el ciclo de mensajes de una aplicación de escritorio. Utilízala únicamente para **crear aplicaciones desde cero**. Si solo necesitas crear ventanas en una aplicación ya existente, `gui` será la dependencia de más alto nivel.
- *"INet"* (Página 381): Utilízala si tu aplicación va a utilizar protocolos de Internet como HTTP. Válida para aplicaciones por línea de comandos o escritorio.

Capítulo 8

Compiladores e IDEs

Es muy difícil escribir software que funcione correctamente y de manera eficiente. Entonces, una vez que un programa funciona en un determinado entorno, no queremos repetir parte del trabajo al moverlo a un compilador, procesador o sistema operativo diferente. ***Idealmente, no debería ser necesario realizar ningún cambio en absoluto****.*

Kernighan & Pike - The Practice of Programming.

Entendemos por **portabilidad** la capacidad de compilar y depurar nuestros programas en otras plataformas diferentes a las que fueron escritos, sin que por ello tengamos que tocar ni una sola línea de código. Entendemos por **plataforma** a la combinación de un compilador y de una arquitectura CPU. Por ejemplo, `v143_x64` hace referencia a Visual Studio 2022 e Intel 64bits. En (Figura 8.1) vemos los diferentes pasos en el proceso de migración de código.

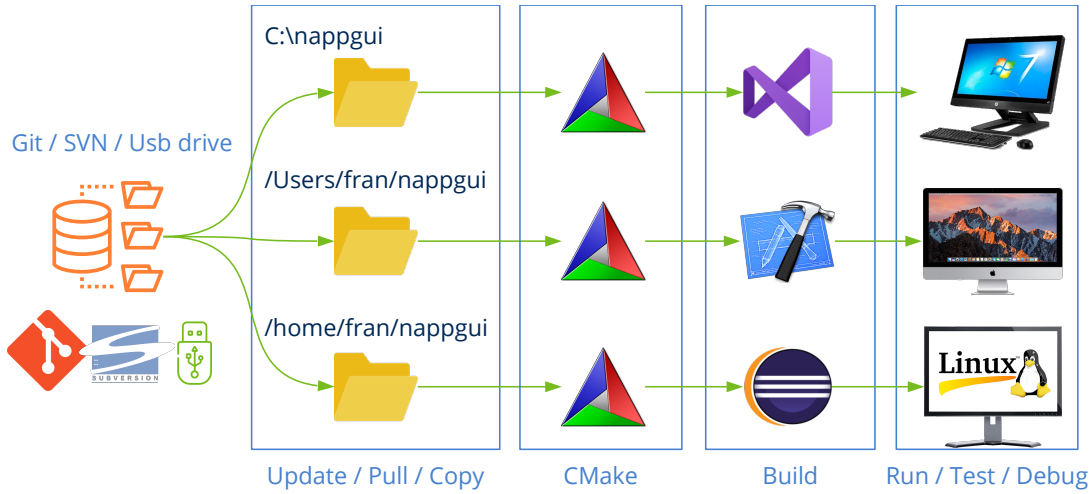


Figura 8.1: Etapas en la portabilidad del código entre plataformas.

- **Copia de trabajo:** En cada máquina deberá existir una copia del código fuente del proyecto. Normalmente esto se llevará a cabo mediante un sistema de control de versiones (SVN, Git, etc).
- **CMake:** creará o actualizará un proyecto de compilación a partir del código fuente utilizando `/src/CMakeLists.txt` y los scripts del directorio `/prj`. Esto se realizará de forma totalmente automática.
- **Compilar:** Utilizando Visual Studio, Xcode o GCC se compilará la solución y se generarán las librerías y ejecutables incluidos en la misma.
- **Ejecutar/Depurar:** Los binarios ya se podrán ejecutar y depurar en la plataforma de destino.

8.1. Compiladores Windows

Podemos utilizar cualquier versión de Visual Studio a partir de 2005 para compilar bajo Windows (Tabla 8.1). Como ya vimos en *"Inicio rápido"* (Página 5) lo primero que

tenemos que hacer es lanzar CMake sobre el código fuente:

	Compilador	Plataforma	S.O.Mínimo
	Visual Studio 2022	v143_x64 (x86)	Vista
	Visual Studio 2019	v142_x64 (x86)	Vista
	Visual Studio 2017	v141_x64 (x86)	Vista
	Visual Studio 2015	v140_x64 (x86)	Vista
	Visual Studio 2013	v120_x64 (x86)	Vista
	Visual Studio 2012	v110_x64 (x86)	Vista
	Visual Studio 2010	v100_x64 (x86)	XP
	Visual Studio 2008	v90_x64 (x86)	XP
	Visual Studio 2005	v80_x64 (x86)	XP

Tabla 8.1: Versiones de Visual Studio soportadas por NAppGUI.

```
cmake -G "Visual Studio 16 2019" -A x64 -T v120 -S ./src -B ./build
```

- `-G` es la versión del compilador (o generador en el argot de CMake).

```
-G "Visual Studio 17 2022"
-G "Visual Studio 16 2019"
-G "Visual Studio 15 2017"
-G "Visual Studio 14 2015"
-G "Visual Studio 12 2013"
-G "Visual Studio 11 2012"
-G "Visual Studio 10 2010"
-G "Visual Studio 9 2008"
-G "Visual Studio 8 2005"
```

- `-A` es la arquitectura Intel 32 o 64 bits:

```
-A x64
-A Win32
```

- `-T` es el *Platform Toolset*. Si omites este parámetro se tomará el último soportado por el compilador.

```
-T v143
-T v142
-T v141
-T v140
```

```
-T v120
-T v110

// For XP compatibility
-T v141_xp
-T v140_xp
-T v120_xp
-T v110_xp
-T v100
-T v90
-T v80
```

- `-S`: Ruta donde se encuentra el `CMakeLists.txt`. Normalmente en el directorio `/src` del SDK.
- `-B`: Ruta donde se generarán los proyectos de compilación, binarios y archivos temporales.

El soporte para Visual Studio 8 2005 fue eliminado en CMake 3.12. Debes utilizar una versión anterior de CMake si aún sigues utilizando VS2005. NAppGUI NO funciona con versiones anteriores a VS2005.

NAppGUI no ofrece soporte para arquitecturas no x86, x64 en Windows: ARM, Itanium, etc.

Tras ejecutar CMake, en la carpeta `/build` aparecerá una solución de VisualStudio, `NAppGUI.sln` o el nombre que se haya configurado en `project(NAppGUI)` del `CMakeLists.txt`. Abre dicha solución y, desde Visual Studio, `Build->Build Solution` para compilar `Debug->Start Debugging` para depurar (Figura 8.2).

Para cambiar la versión de Visual Studio, selecciona otro generador en CMake ***-G "Visual Studio 15 2017"****, cierra y vuelve a abrir la solución.*

8.1.1. Platform toolset

A partir de Visual Studio 2010, se produce una disociación entre el editor y el compilador. El término *Plaform Toolset* identifica al propio compilador, que se podrá seguir utilizando con IDEs mas modernos. Si no indicamos nada, CMake utilizará el toolset incluido por defecto en cada versión de VS, pero se puede cambiar mediante el parámetro `-T` de CMake (Tabla 8.2). Por ejemplo, podemos combinar `Visual Studio 15 2017` con el toolset de VS2013 para Windows XP `v120_xp`:

```
cmake -G "Visual Studio 16 2019" -A Win32 -T v120_xp -S ./src -B ./build
```

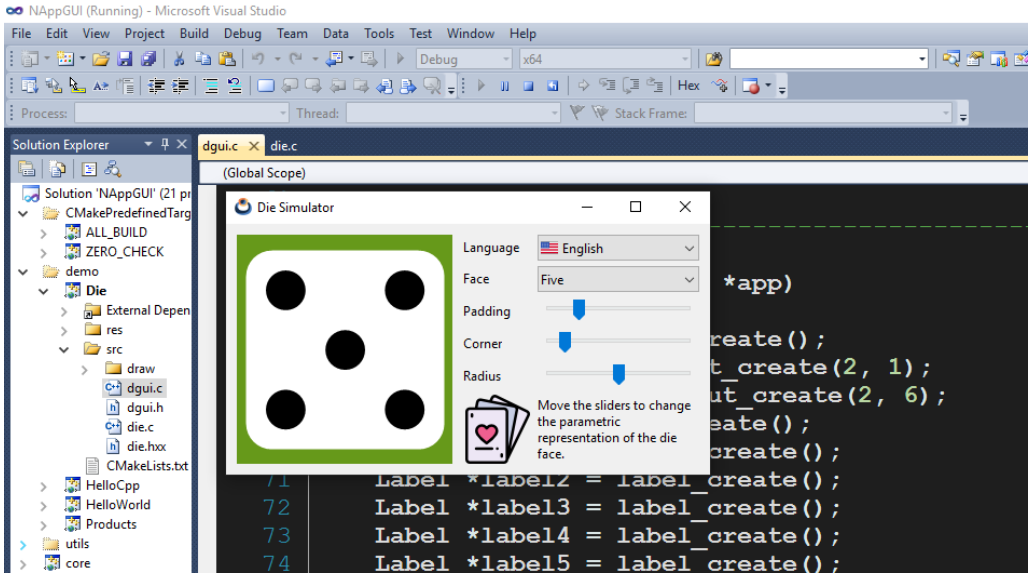


Figura 8.2: Depurando la aplicación *Die* en Visual Studio 2010.

Toolset (-T)	Versión de VS
v143	Visual Studio 2022
v142	Visual Studio 2019
v141	Visual Studio 2017
v141_xp	Visual Studio 2017 (con soporte XP)
v140	Visual Studio 2015
v140_xp	Visual Studio 2015 (con soporte XP)
v120	Visual Studio 2013
v120_xp	Visual Studio 2013 (con soporte XP)
v110	Visual Studio 2012
v110_xp	Visual Studio 2012 (con soporte XP)
v100	Visual Studio 2010
v90	Visual Studio 2008
v80	Visual Studio 2005

Tabla 8.2: Toolset incluido en cada versión de VS.

Es necesario que tengas instalada cada versión de Visual Studio para utilizar su toolset. Existen versiones "ligeras" que instalan las ***build tools*** *sin el entorno de desarrollo.*

8.1.2. Visual C++ Redistributable

Por defecto, Visual Studio enlaza dinámicamente las funciones de la librería estándar de C, lo que provoca que los `.exe` puedan no funcionar en máquinas que no dispongan de las DLL de `VC++` (Figura 8.3). Esto obliga a las aplicaciones a incluir una copia de `MSVCRT.dll`, `VCRUNTIME.dll`, ... o a instalar los famosos paquetes *Visual C++ Redistributable* para asegurar que la aplicación pueda correr sin problemas.

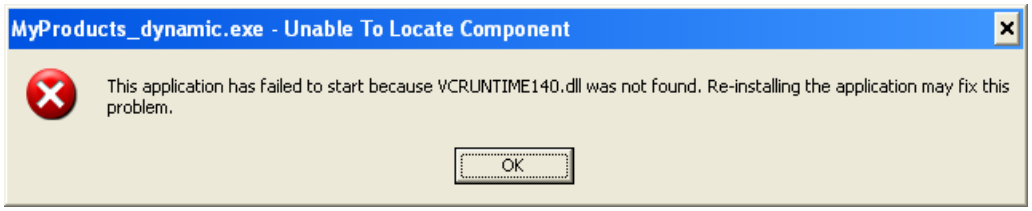


Figura 8.3: Error por la falta de las .dll de VC++.

NAppGUI utiliza un reducido conjunto de la librería C, ya que accede directamente al API de Windows siempre que sea posible. Por esta razón, todas las aplicaciones creadas con NAppGUI realizan un enlace estático (opción `/MT`) de las funciones necesarias de la stdlib, evitando dependencias a costa de aumentar ligeramente (unos pocos Kb) el tamaño del ejecutable final. Esto garantiza que las aplicaciones correrán sin problemas en todas las máquinas Windows sin necesidad de DLLs adicionales y sin tener que instalar los *VC++ Redistributable*.

> *Las aplicaciones NAppGUI no requieren los Visual C++ Redistributable. Tampoco utilizan las MFC "Microsoft Foundation Classes" ni la plataforma .NET.*

8.1.3. Soporte WindowsXP

A partir de VS2012, el *Platform Toolset* genera ejecutables no compatibles con WindowsXP. Si queremos que nuestras aplicaciones corran en este sistema, deberemos seleccionar el toolset alternativo acabado en `_xp`: `v141_xp`, `v140_xp`, `v120_xp`, `v110_xp`. O bien los `v100`, `v90` ó `v80` (VS2010, 2008, 2005), que sí soportan directamente XP (Figura 8.4).

> *El soporte para WindowsXP se ha eliminado definitivamente en Visual Studio 2019.* ***No existe el Platform Toolset v142_xp.***

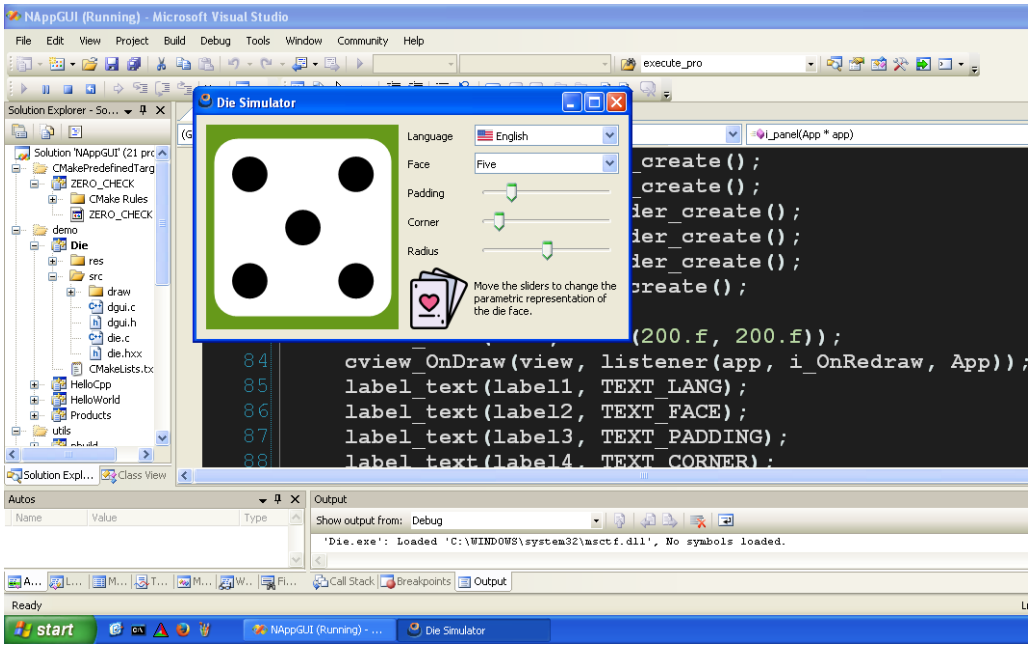


Figura 8.4: Depurando la aplicación *Die* en WindowsXP con VS2005 (toolset v80).

No se pueden crear aplicaciones con NAppGUI que funcionen en Windows anteriores a XP.

8.1.4. Soporte SSE

Con el Pentium III, Intel incorporó un juego de instrucciones adicional para operaciones en coma flotante denominado SSE *Streaming SIMD Extensions*. Esto permite optimizar los cálculos matemáticos a costa de perder la compatibilidad, ya que las aplicaciones que utilicen SSE no funcionarán en los modelos Pentium II o anteriores. En NAppGUI se han reservado los toolset `v80_x86` y `v90_x86` para crear aplicaciones compatibles con los procesadores más antiguos (Tabla 8.3). A partir de `v100_x86`, se utilizará SSE2 en todos los toolset.

Toolset	SSE	CPU Mínima
v80_x86	x87 (no SSE)	Pentium II/AMD K6
v90_x86	SSE	Pentium III/AMD Duron
v100_x86	SSE2	Pentium IV/AMD Sempron
v110_x86	SSE2	Pentium IV/AMD Sempron
...	SSE2	...

Tabla 8.3: Soporte SSE

El soporte SSE solo se deshabilita en arquitecturas de 32 bits (x86). Todas las CPU de 64 bits (x64) incorporan SSE2.

8.2. Compiladores macOS

Para compilar para los iMac, macBook y macMini de Apple necesitaremos CMake y Xcode[1] a partir de la versión 3.2.6 (Tabla 8.4). NAppGUI permite generar aplicaciones que funcionen en MacOSX 10.6 Snow Leopard y posteriores:

[1] `https://developer.apple.com/xcode/`

	Compilador	S.O.Mínimo	Plataforma
	Xcode 14.1	Ventura	sdk13_1_x64 (arm)
	Xcode 13.4.1	Monterey	sdk12_3_x64 (arm)
	Xcode 12.5.1	Big Sur	sdk11_5_x64 (arm)
	Xcode 11.7	Catalina	sdk10_15_x64

	Compilador	S.O.Mínimo	Plataforma
	Xcode 10.3	Mojave	sdk10_14_x64
	Xcode 9.4.1	High Sierra	sdk10_13_x64
	Xcode 8.3.3	Sierra	sdk10_12_x64
	Xcode 7.3.1	El Capitan	sdk10_11_x64
	Xcode 6.4	Yosemite	sdk10_10_x64
	Xcode 6.2	Mavericks	sdk10_9_x64
	Xcode 5.1.1	Mountain Lion	sdk10_8_x64
	Xcode 4.6.3	Lion	sdk10_7_x64
	Xcode 3.2.6	Snow Leopard	sdk10_6_x64 (x86)

Tabla 8.4: Versiones de Xcode soportadas por NAppGUI.

```
cmake -G "Xcode" -DCMAKE_DEPLOYMENT_TARGET="11.0" -DCMAKE_ARCHITECTURE="arm64"
    ↪ -S ./src -B ./build
```

- `-G` siempre `"Xcode"`. Utilizar `xcode-select` para alternar si tienes varias versiones instaladas.
- `-DCMAKE_DEPLOYMENT_TARGET`. Sistema operativo mínimo que será soportado. Si se omite se establecerá el **Base SDK** incluido en la versión de Xcode.

```
-DCMAKE_DEPLOYMENT_TARGET="13.1"    // Ventura
-DCMAKE_DEPLOYMENT_TARGET="13.0"    // Ventura
-DCMAKE_DEPLOYMENT_TARGET="12.4"    // Monterey
-DCMAKE_DEPLOYMENT_TARGET="12.3"    // Monterey
-DCMAKE_DEPLOYMENT_TARGET="12.2"    // Monterey
-DCMAKE_DEPLOYMENT_TARGET="12.0"    // Monterey
-DCMAKE_DEPLOYMENT_TARGET="11.5"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="11.4"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="11.3"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="11.2"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="11.1"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="11.0"    // Big Sur
-DCMAKE_DEPLOYMENT_TARGET="10.15"   // Catalina
-DCMAKE_DEPLOYMENT_TARGET="10.14"   // Mojave
-DCMAKE_DEPLOYMENT_TARGET="10.13"   // High Sierra
-DCMAKE_DEPLOYMENT_TARGET="10.12"   // Sierra
-DCMAKE_DEPLOYMENT_TARGET="10.11"   // El Capitan
-DCMAKE_DEPLOYMENT_TARGET="10.10"   // Yosemite
-DCMAKE_DEPLOYMENT_TARGET="10.9"    // Mavericks
-DCMAKE_DEPLOYMENT_TARGET="10.8"    // Mountain Lion
```

```
-DCMAKE_DEPLOYMENT_TARGET="10.7"    // Lion
-DCMAKE_DEPLOYMENT_TARGET="10.6"    // Snow Leopard
```

- `-DCMAKE_ARCHITECTURE`. `arm64`, `x64`, `i386`. La arquitectura `arm64` se incluye a partir del SDK 11.0 Big Sur. La `i386` se declaró obsoleta en macOS 10.13 High Sierra.

```
-DCMAKE_ARCHITECTURE="arm64"
-DCMAKE_ARCHITECTURE="x64"
-DCMAKE_ARCHITECTURE="i386"
```

*NAppGUI no soporta la creación de **Apple's Fat binaries**. Debes indicar un solo valor en este campo.*

- `-S`: Ruta donde se encuentra el `CMakeLists.txt`. Normalmente en el directorio `/src` del SDK.
- `-B`: Ruta donde se generarán los proyectos de compilación, binarios y archivos temporales.

Tras ejecutar CMake, en la carpeta `/build` aparecerá una solución de Xcode, `NAppGUI.xcodeproj` o el nombre que se haya configurado en `project(NAppGUI)` del `CMakeLists.txt`. Al abrir la solución Xcode, vemos los diferentes proyectos que la forman, incluidos *Die* y *Dice*. Seleccionamos *Die* en el desplegable superior izquierdo y después pulsamos `Play` o `Product->Run` (Figura 8.5). Esto compilará el programa y lo lanzará en modo depuración, donde podremos establecer puntos de ruptura para inspeccionar la pila y el valor de las variables.

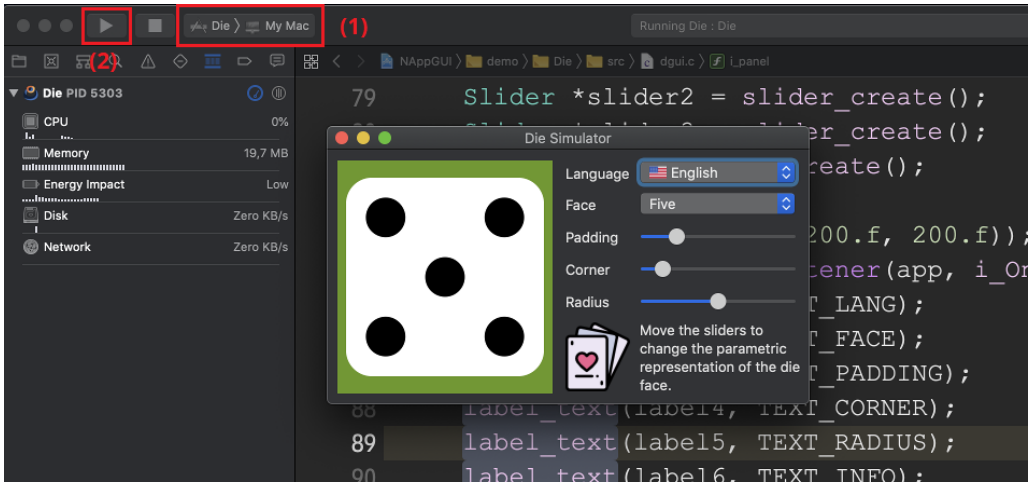


Figura 8.5: Depurando la aplicación *Die* en Xcode.

8.2.1. Base SDK y Deployment Target

Cada año, Apple lanza una nueva versión de macOS, que viene acompañada de un nuevo SDK y de la actualización de Xcode que incluye dicho SDK. A esta se denomina **Base SDK**.

> ***Base SDK*** *es la versión incluida en cada nueva versión mayor de Xcode, que coincide con la última versión del sistema macOS aparecida en el mercado.*

Apple tiene una política mucho más restrictiva que Microsoft en lo referente a la compatibilidad de las aplicaciones con versiones anteriores del sistema operativo. Por defecto, un programa compilado con SDK 10.14 (macOS Mojave) no funcionará en el inmediatamente anterior macOS High Sierra (Figura 8.6).

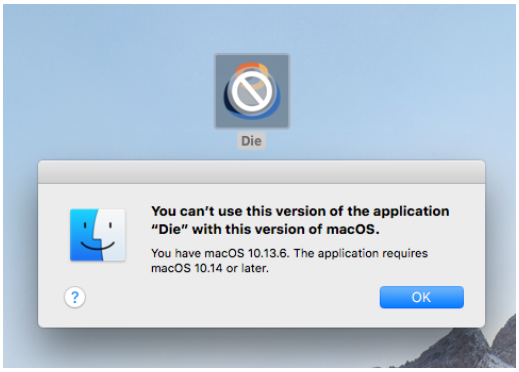


Figura 8.6: *Die* con *Base SDK* 10.14 no funcionará en High Sierra.

Para evitar este problema, y que las aplicaciones funcionen en macOS más antiguos, existe el parámetro **Deployment Target**. Al utilizarlo, se activará una macro que anulará las nuevas características del Base SDK. Esto permitirá que el programa corra en versiones antiguas a costa, claro está, de no tener acceso a las últimas funcionalidades de los iMac. Podrás seleccionar el Deployment Target requerido por tu proyecto a través del parámetro `-DCMAKE_DEPLOYMENT_TARGET`, como ya hemos visto en el apartado anterior.

> *Xcode 14 considera obsoletos los Deployment Target inferiores a 10.13 (Figura 8.7). Utiliza Xcode 13 si quieres compatibilidad con Mac OSX 10.12 Sierra y anteriores.*

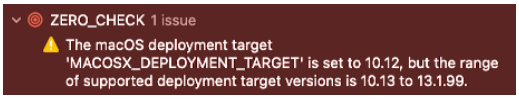


Figura 8.7: Deployment Target 10.12 obsoleto a partir de Xcode 14.

> *Xcode 8 considera obsoletos los Deployment Target inferiores a 10.9 (Figura 8.8). Utiliza Xcode 7 si quieres compatibilidad con Mac OSX 10.8 Mountain Lion y anteriores.*

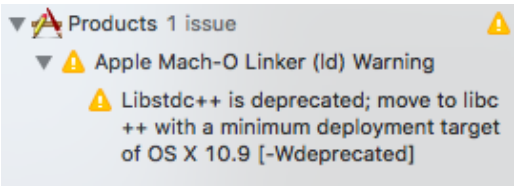


Figura 8.8: Deployment Target 10.8 obsoleto a partir de Xcode 8.

8.2.2. xcode-select

Ya hemos visto que CMake solo ofrece un generador para Xcode (`-G "Xcode"`), aunque es posible tener varias versiones instaladas en la misma máquina, cada una dentro de su propio *bundle* `Xcode.app`. Siempre existirá un Xcode por defecto en el sistema (el más reciente) pero se puede cambiar mediante la utilidad `xcode-select`:

Consulta de la versión actual de Xcode.

```
xcode-select -p
/Applications/Xcode.app/Contents/Developer
```

Cambio de la versión activa de Xcode.

```
sudo xcode-select -s /Applications/Xcode8.app/Contents/Developer
```

Establecer la versión por defecto de Xcode.

```
sudo xcode-select -r
```

*Deberás ejecutar de nuevo **cmake -G "Xcode"...** cada vez que utilices `xcode-select` para que tu proyecto actualice el cambio de compilador.*

8.2.3. macOS ARM

En Noviembre de 2020 Apple lanza su nueva línea de ordenadores de sobremesa y portátiles (iMac, macBook y macMini) basados en el procesador Apple M1 con arquitectura ARM (Figura 8.9). A pesar que son capaces de ejecutar programas compilados para Intel x64 mediante el programa Rosetta 2 (Figura 8.10) lo ideal sería compilar nuestras aplicaciones para la nueva arquitectura con el fin de optimizar al máximo los ejecutables.

NAppGUI soporta la compilación para la arquitectura Apple ARM. Tan solo deberás incluir la opción `-DCMAKE_ARCHITECTURE="arm64"` en CMake, como ya vimos en la sección anterior.

Puedes compilar la arquitectura M1 desde máquinas Intel x64, pero no podrás depurar los ejecutables.

Figura 8.9: Procesadores M1 de Apple.

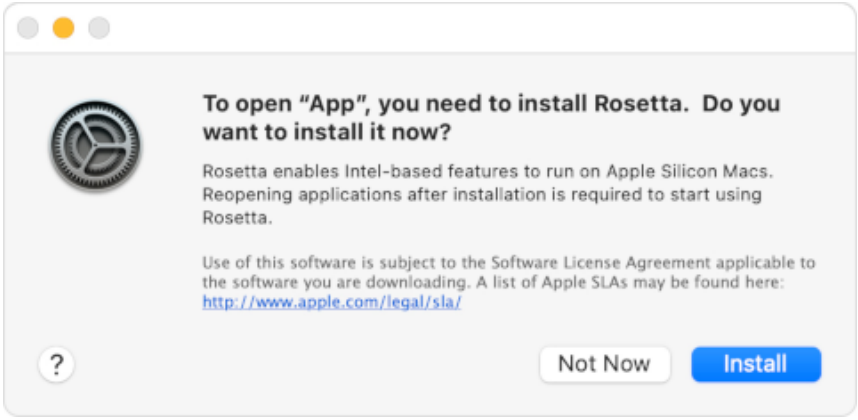


Figura 8.10: Aviso Rosetta 2 al intentar ejecutar código x64 en un Apple M1.

La arquitectura M1 solo está disponible para el sistema Big Sur (macOS 11.0) y posteriores.

8.2.4. macOS 32bits

Desde la versión macOS High Sierra, Apple ha declarado obsoleta[2] la arquitectura de 32 bits, emitiendo avisos a los usuarios en el caso de detectar ejecutables `i386` (Figura 8.11). A partir de Xcode 10, no se puede compilar en esta arquitectura (Figura 8.12).

El soporte para aplicaciones 32bits ha desaparecido definitivamente en ***macOS Catalina****, que solo permite ejecutar aplicaciones de 64bits.*

Esto tiene cierto sentido ya que todos los modelos de iMac basados en Intel incorporan

[2] `https://support.apple.com/en-us/HT208436`

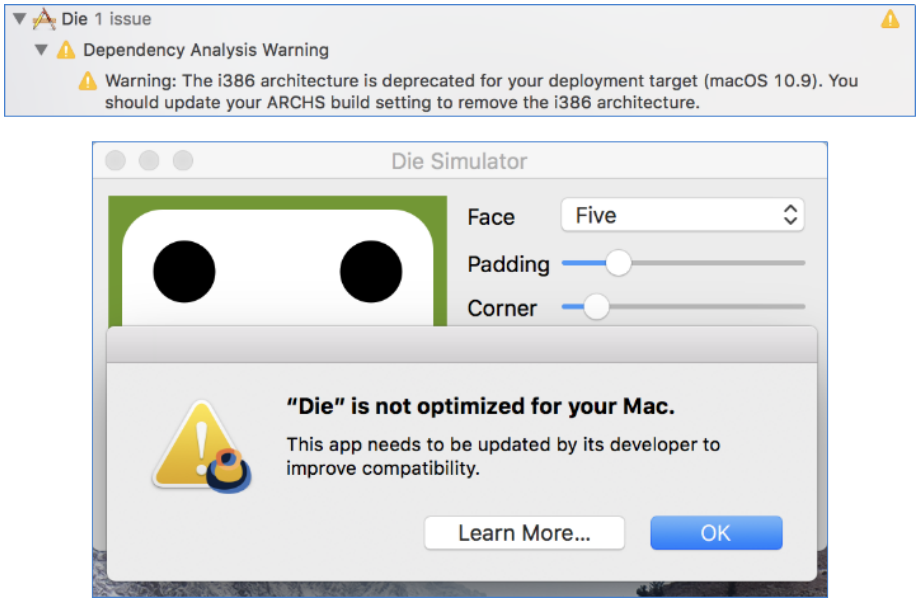


Figura 8.11: Avisos de macOS en aplicaciones de 32bits.

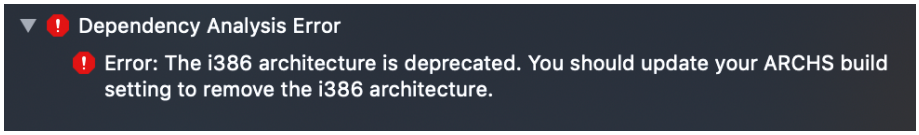


Figura 8.12: Error en Xcode 10 al intentar compilar en 32bits.

procesadores de 64 bits, a excepción de unos pocos modelos de 2006 en policarbonato blanco que montaban el Intel Core Duo de 32 bits (Figura 8.13). Estos iMac admitían como máximo el Mac OSX 10.6 Snow Leopard, siendo requisito fundamental a partir de 10.7 Lion, el disponer de una CPU de 64 bits. Para compilar sin problemas en 32bits hay que utilizar, como máximo, Xcode 6 (Figura 8.14).

Figura 8.13: Únicos modelos de Apple con procesador Intel de 32bits.

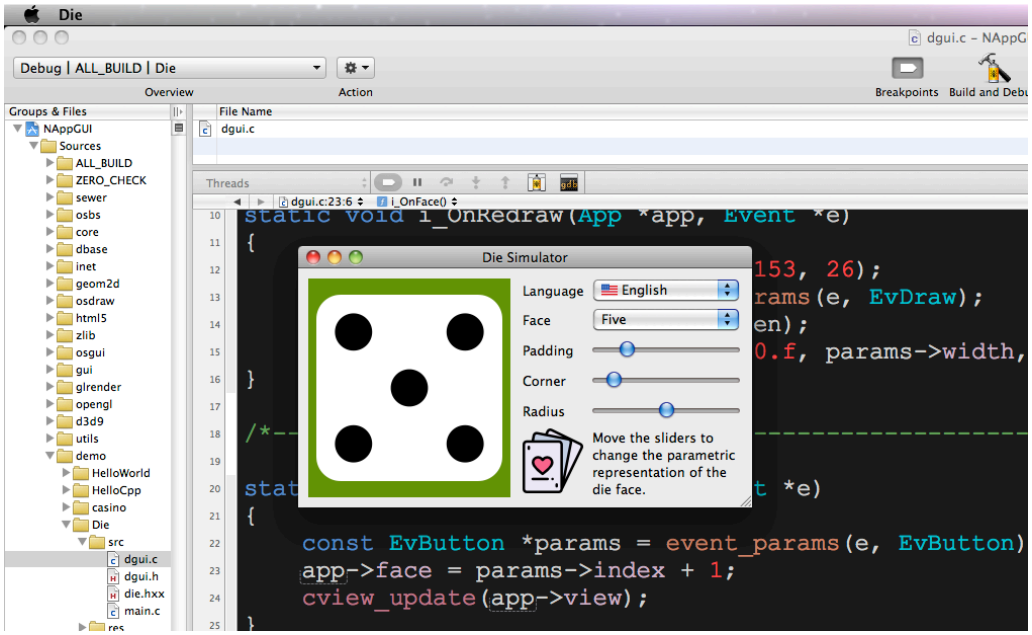


Figura 8.14: Compilación en 32bits con Xcode 3.2.6 (Snow Leopard).

8.3. Compiladores Linux

Para las versiones Linux, utilizaremos el compilador `gcc` (Tabla 8.5) y la herramienta `make` para generar los binarios, pero no existe un entorno de desarrollo "oficial" tal como ocurre en Windows y macOS. Para realizar una configuración elemental de nuestro equipo teclear los siguientes comandos en un terminal:

```
// Development tools
sudo apt-get install build-essential
sudo apt-get install git
sudo apt-get install cmake

// Development libraries
sudo apt-get install libgtk-3-dev
sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
sudo apt-get install libcurl4-openssl-dev

// GTK Inspector (Ctrl+D when debugging)
gsettings set org.gtk.Settings.Debug enable-inspector-keybinding true

// Check system libraries version
pkg-config --modversion gtk+-3.0
3.24.20

pkg-config --modversion libcurl
7.68.0
```

	S.O.Mínimo	Compilador	Toolkit	Plataforma
	Ubuntu 22.04 LTS	GCC 11.2.0	GTK 3.24.33	gcc11_gtk3_x64
	Ubuntu 20.04 LTS	GCC 9.3	GTK 3.24.20	gcc9_gtk3_x64
	Ubuntu 18.04 LTS	GCC 7.5	GTK 3.22.30	gcc7_gtk3_x64
	Ubuntu 16.04 LTS	GCC 5.4	GTK 3.18.9	gcc5_gtk3_x64 (x86)
	Ubuntu 14.04 LTS	GCC 4.8.4	GTK 3.10.8	gcc4_8_gtk3_x64 (x86)
	Ubuntu 12.04 LTS	GCC 4.6.3	GTK 3.4.2	gcc4_6_gtk3_x64 (x86)
	Raspbian 11 Bullseye	GCC 10.2.1	GTK 3.24.24	gcc10_gtk3_arm64
	Raspbian 10 Buster	GCC 8.3.0	GTK 3.24.5	gcc8_gtk3_arm
	Raspbian 9.1 Strech	GCC 6.3.0	GTK 3.22.11	gcc6_gtk3_arm
	Raspbian 8.0 Jessie	GCC 4.9.2	GTK 3.14.5	gcc4_9_gtk3_arm

Tabla 8.5: Versiones de GCC soportadas por NAppGUI.

Al igual que hicimos en Windows y macOS, ejecutamos `cmake` para generar el proyecto de compilación:

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_CONFIG="Debug" -DCMAKE_ARCHITECTURE="
    ↪ x64" -DCMAKE_TOOLKIT="GTK3" -S ./src -B ./build
```

- `-G` siempre `"Unix Makefiles"`. Adicionalmente se pueden crear proyectos para los principales IDEs disponibles en Linux:

```
-G "Unix Makefiles"
-G "CodeBlocks - Unix Makefiles"
-G "CodeLite - Unix Makefiles"
-G "Sublime Text 2 - Unix Makefiles"
-G "Kate - Unix Makefiles"
-G "Eclipse CDT4 - Unix Makefiles"
```

- `-DCMAKE_BUILD_CONFIG`. A diferencia de Visual Studio y Xcode, Make no permite la creación de proyectos multi-configuración. Hay que indicarla en el momento de la generación:

```
-DCMAKE_BUILD_CONFIG="Debug"
-DCMAKE_BUILD_CONFIG="Release"
```

```
-DCMAKE_BUILD_CONFIG="ReleaseWithAssert"
```

- `-DCMAKE_ARCHITECTURE`. `x64`, `i386`, `arm`, `arm64`. En Linux no se permite la compilación cruzada. Deberemos seleccionar la misma arquitectura que la máquina anfitrión. Este parámetro puede omitirse, se configurará automáticamente.

```
-DCMAKE_ARCHITECTURE="x64"      // Only in Linux Intel 64bits hosts
-DCMAKE_ARCHITECTURE="i386"     // Only in Linux Intel 32bits hosts
-DCMAKE_ARCHITECTURE="arm"      // Only in Linux ARM 32bits hosts
-DCMAKE_ARCHITECTURE="arm64"    // Only in Linux ARM 64bits hosts
```

- `-DCMAKE_TOOLKIT`. A día de hoy, la única opción disponible es `"GTK3"`, ya que NAppGUI no soporta otros toolkits gráficos. Este parámetro puede omitirse, se configurará automáticamente.

```
-DCMAKE_TOOLKIT="GTK3"
```

- `-S`: Ruta donde se encuentra el `CMakeLists.txt`. Normalmente en el directorio `/src` del SDK.
- `-B`: Ruta donde se generarán los proyectos de compilación, binarios y archivos temporales.

Tras ejecutar `cmake` tendremos, en la carpeta `/build` una serie de `Makefiles` preparados para compilar el proyecto.

```
cmake --build ./build -j 4

...
[ 93%] Linking CXX executable ../../Debug/bin/DrawBig
[ 93%] Linking CXX executable ../../Debug/bin/GuiHello
[ 93%] Built target DrawBig
[ 94%] Building C object howto/drawhello/CMakeFiles/DrawHello.dir/resgen/
    ↪ res_drawhello.c.o
[ 94%] Linking CXX executable ../../Debug/bin/Col2dHello
[ 98%] Built target GuiHello
[ 98%] Building C object howto/drawimg/CMakeFiles/DrawImg.dir/resgen/
    ↪ res_drawimg.c.o
[ 98%] Linking CXX executable ../../Debug/bin/UrlImg
[ 98%] Linking CXX executable ../../Debug/bin/DrawHello
[ 98%] Built target Col2dHello
[ 98%] Linking CXX executable ../../Debug/bin/ColorView
[ 98%] Built target UrlImg
[ 98%] Built target DrawHello
[ 99%] Linking CXX executable ../../Debug/bin/DrawImg
[100%] Built target ColorView
[100%] Built target DrawImg
```

Una vez terminada la compilación, podemos lanzar los ejecutables directamente desde el terminal:

Lanzar la aplicación *Die*.

```
./build/demo/die/Debug/Die
```

Si tienes cierta soltura con `gdb`, puedes intentar depurar el código directamente desde el terminal (Figura 8.15). Más adelante veremos como hacerlo mediante Eclipse y Visual Studio Code.

Depurando *Die* con *gdb*

```
gdb ./build/demo/die/Debug/Die
(gdb) run
...
```

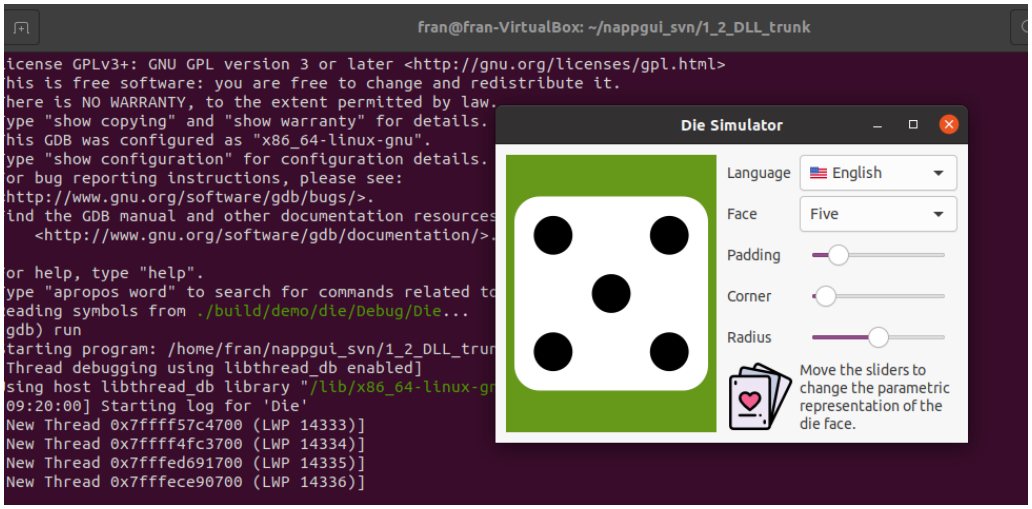


Figura 8.15: Depurando *Die* con GDB desde el terminal.

8.3.1. GTK+3

A diferencia de Windows y macOS, en Linux soporta multitud de entornos de escritorio basados en diferentes librerías (o *toolkits*) siendo GTK y Qt las dos más famosas. NAppGUI utiliza GTK+3 para la parte gráfica ya que es la base del entornos Gnome, Xfce, Lxde, etc, (Tabla 8.6) presentes en muchas de las distribuciones mas extendidas. GTK+3 estará presente de forma natural en todas ellas, no siendo necesarias otras dependencias adicionales. Eso sí, para compilar bajo GTK+3 tendremos que instalar la versión de desarrollador, como vimos al inicio de esta sección.

	Entorno	Distribuciones
	Gnome	Ubuntu, Debian, Fedora, Red Hat, CentOS, Manjaro, Suse, Arch, ...
	Xfce	Xubuntu, Debian, Fedora, Manjaro, ...
	Lxde	Lubuntu, Raspbian, Debian, Fedora, Mandriva, ...
	Cinnamon	Mint, Debian, Ubuntu, Fedora, OpenSuse, ...
	Mate	Ubuntu Mate, Mint, Debian, Fedora, OpenSuse, ...
	Pantheon	Elementary OS
	Sugar	

Tabla 8.6: Entornos de escritorio basados en GTK.

8.3.2. Múltiples versiones de GCC

Aunque cada distribución de Linux incorpora una versión "canónica" de GCC, es posible tener varias instaladas en la misma máquina y alternar entre ellas de forma similar a como hacíamos en macOS con `xcode-select`. Para ello utilizaremos el comando `update -alternatives` de Linux. Suponemos que estamos en Ubuntu 18.04 LTS:

Versión de gcc instalada.

```
gcc --version
gcc 7.5.0
```

Instalar gcc-6

```
sudo apt-get install gcc-6 g++-6
```

Registrar gcc-7 y gcc-6

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 60 --slave /
    ↪ usr/bin/g++ g++ /usr/bin/g++-7
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-6 50 --slave /
    ↪ usr/bin/g++ g++ /usr/bin/g++-6
```

Cambiar a gcc-6.

```
sudo update-alternatives --set gcc /usr/bin/gcc-6
gcc --version
gcc 6.5.0
g++ --version
g++ 6.5.0
```

Volver a la versión por defecto de gcc.

```
sudo update-alternatives --auto gcc
gcc --version
gcc 7.5.0
g++ --version
g++ 7.5.0
```

8.3.3. Linux 32bits

Para compilar aplicaciones 32bits desde un sistema Ubuntu 64bits es necesario instalar el paquete `multilib`:

```
sudo apt-get install gcc-multilib
```

Pero actualmente existen problemas[3] para realizar compilación cruzada que incluya la librería GTK+, por lo que no será posible utilizar la misma máquina de desarrollo para generar en ambas arquitecturas, como ocurre en Windows. Las aplicaciones de consola o librerías que no accedan a GTK sí que se pueden compilar en 32bits desde un ordenador de 64bits.

No es posible compilar en 32bits desde un sistema Ubuntu de 64bits aplicaciones que utilicen GTK+3. Debes utilizar para ello un sistema Linux de 32bits.

8.3.4. Linux ARM

La arquitectura ARM[4] *Advanced RISC Machine* es la predominante en el mercado de los dispositivos embebidos como teléfonos inteligentes y tablets. Actualmente, NAppGUI no ofrece soporte para el desarrollo de aplicaciones móviles iOS/Android, pero sí para otro tipo de placas que soporten versiones de Linux ARM "de escritorio", como la Raspberry PI. Para portar nuestro código a Raspberry Pi hay que seguir los mismos pasos que en Ubuntu Linux (Figura 8.16). Ambas distribuciones están basadas en Debian, por lo que disponemos de GCC, CMake y Make de forma directa a través de `apt-get`.

8.3.5. Eclipse CDT

Trabajar directamente con el terminal nos brinda una gran flexibilidad a la hora de configurar nuestras propies herramientas. Con volver a la consola y teclear `cmake --build ./build -j 4` se re-compilará todo lo necesario. Ahora bien, utilizar `GDB` directamente resultará bastante tedioso, por lo que el uso de un depurador integrado (o IDE) se hace casi imprescindible. Para el desarrollo de NAppGUI utilizamos de forma intensiva Eclipse

[3] https://ubuntuforums.org/showthread.php?t=2038875
[4] https://en.wikipedia.org/wiki/ARM_architecture

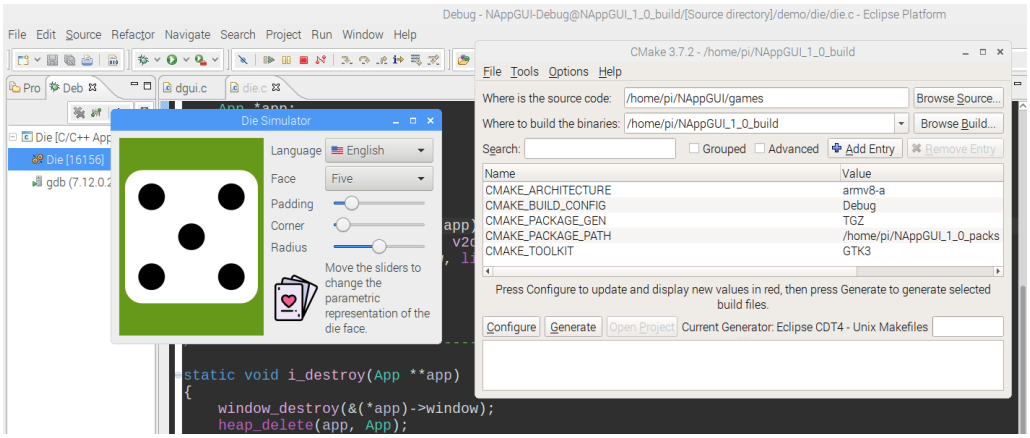


Figura 8.16: Depuración de la aplicación *Die* en una Raspberry Pi.

CDT[5]. Este entorno nos permitirá programar con una metodología similar a la de Visual Studio y Xcode: Situar puntos de ruptura, inspeccionar la pila y variables, buscar archivos dentro del directorio de código, ediciones múltiples, búsquedas masivas, etc.

La única diferencia es que deberemos utilizar el generador `-G "Eclipse CDT4 - Unix Makefiles"` en CMake que, adicionalmente a los `Makefile`, creará los archivos `.cproject` y `.project` necesarios para importar el proyecto dentro de Eclipse.

Abrimos Eclipse y hacemos `File->Import->Existing Projects into Workspace`. Aparecerá un cuadro de diálogo donde indicamos el directorio *build* que hayamos configurado en CMake (`/build`). Eclipse abrirá el proyecto situando a la izquierda un árbol con todos los archivos y compilaremos con `Project->Build All`. A la hora de depurar (Die en este caso) crearemos un perfil desde `Run->Debug Configurations->C/C++ Application`. Pulsamos `[Search Project...]` y seleccionamos *Die* en la lista desplegable. Finalmente pulsamos `[Debug]` para depurar la aplicación de forma interactiva (Figura 8.17).

Algunas opciones interesantes de Eclipse CDT bajo `Window->Preferences`.

- Run/Debug->Launching->Terminate and Relaunch while launching.

El uso de Eclipse es solo una recomendación. Tienes total libertad para utilizar las herramientas que mejor consideres.

[5] `https://www.eclipse.org/cdt/`

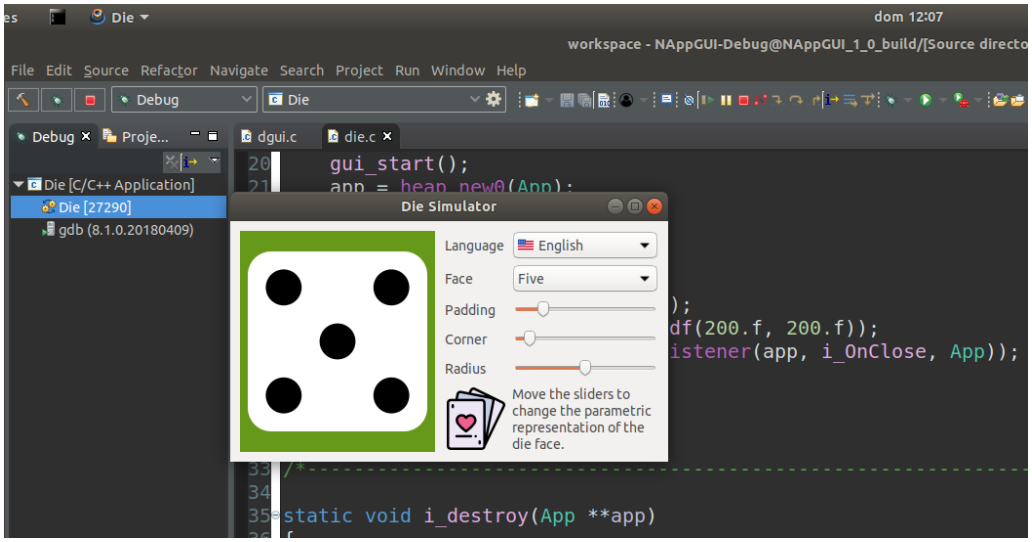


Figura 8.17: Depurando la aplicación *Die* con Eclipse.

8.3.6. Visual Studio Code

Otro entorno interesante para desarrollar en Linux es Visual Studio Code. Con las extensiones apropiadas, es posible trabajar en C/C++ con CMake de una forma muy cómoda y fluida. Para instalarlo:

```
sudo apt-get install code
```

Añadimos, como mínimo, **C/C++ Extension Pack** que incluirá también soporte para CMake (Figura 8.18).

Abrimos nuestro proyecto con `Open Folder`. Posteriormente, ejecutamos CMake desde el propio entorno: `[F1]->CMake:Configure`. La primera vez, VSCode preguntará por la ubicación del `CMakeLists.txt` principal (Figura 8.19) (`/src/CMakeLists.txt`).

Tras la configuración ya podemos compilar con `[F1]->CMake:Build`. En la pestaña **Output** de VSCode veremos la evolución del proceso:

```
[build] [ 97 %] Building C object demo/die/CMakeFiles/Die.dir/resgen/res_die.c.o
[build] [ 98 %] Built target Bode
[build] [ 98 %] Building C object demo/products/CMakeFiles/Products.dir/products
    ↪ .c.o
[build] [ 98 %] Built target Fractals
[build] [ 98 %] Building C object demo/products/CMakeFiles/Products.dir/prview.c
    ↪ .o
[build] [ 99 %] Linking CXX executable ../../Debug/bin/Die
[build] [100 %] Building C object demo/products/CMakeFiles/Products.dir/resgen/
    ↪ res_products.c.o
[build] [100 %] Built target Die
```

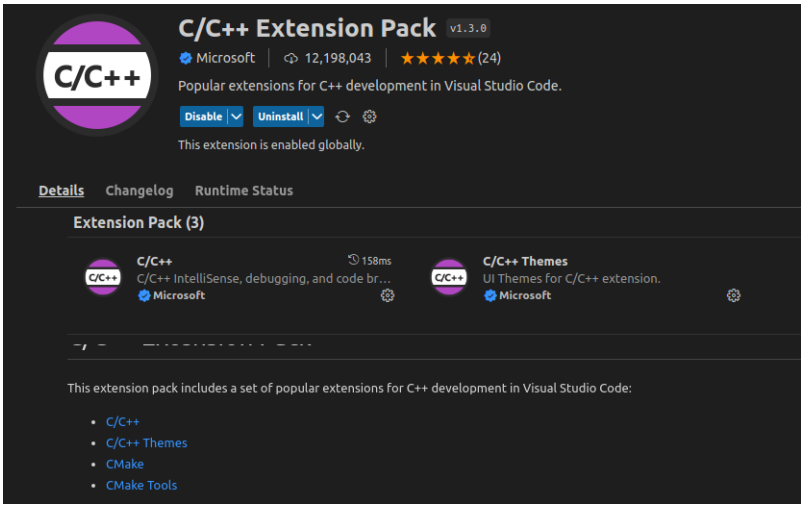


Figura 8.18: C/C++ Extension Pack.

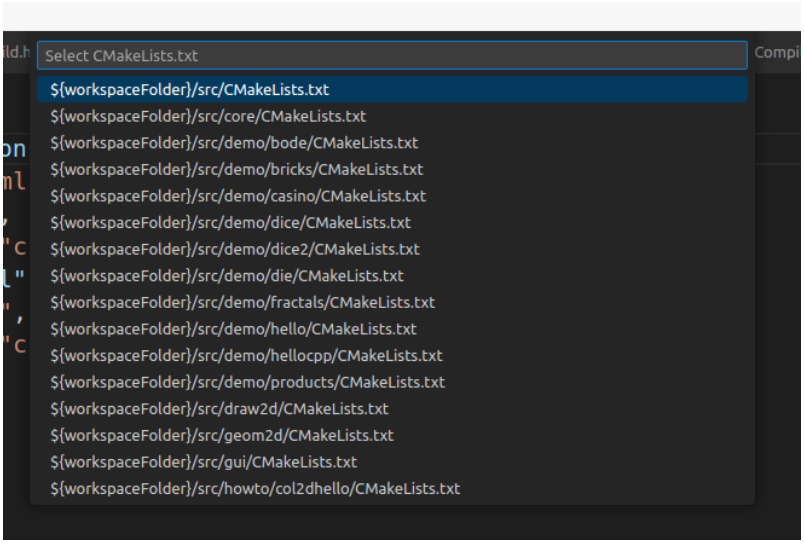


Figura 8.19: Selección del CMakeLists.txt principal del proyecto.

```
[build] [100 %] Linking CXX executable ../../Debug/bin/Products
[build] [100 %] Built target Products
```

Para depurar, lo primero es seleccionar el target (o ejecutable) con `[F1]->CMake:Set Debug Target` (Figura 8.20).

Y lanzamos el depurador con `[F1]->CMake:Debug` (Figura 8.21).

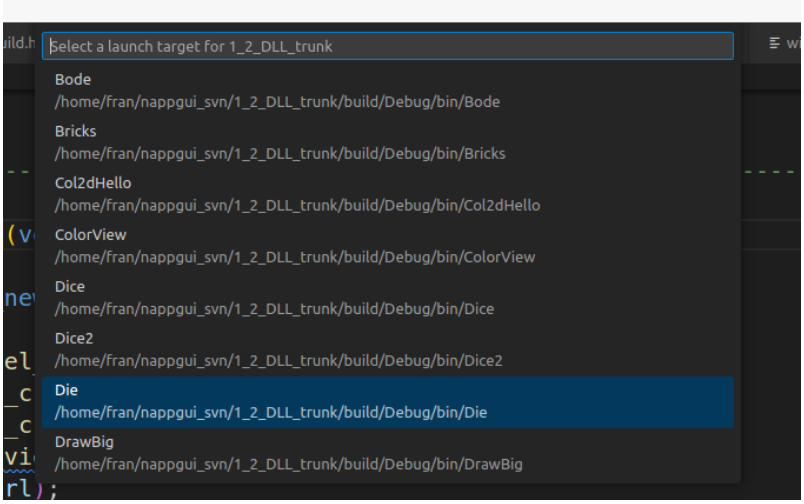


Figura 8.20: Selección del ejecutable a depurar.

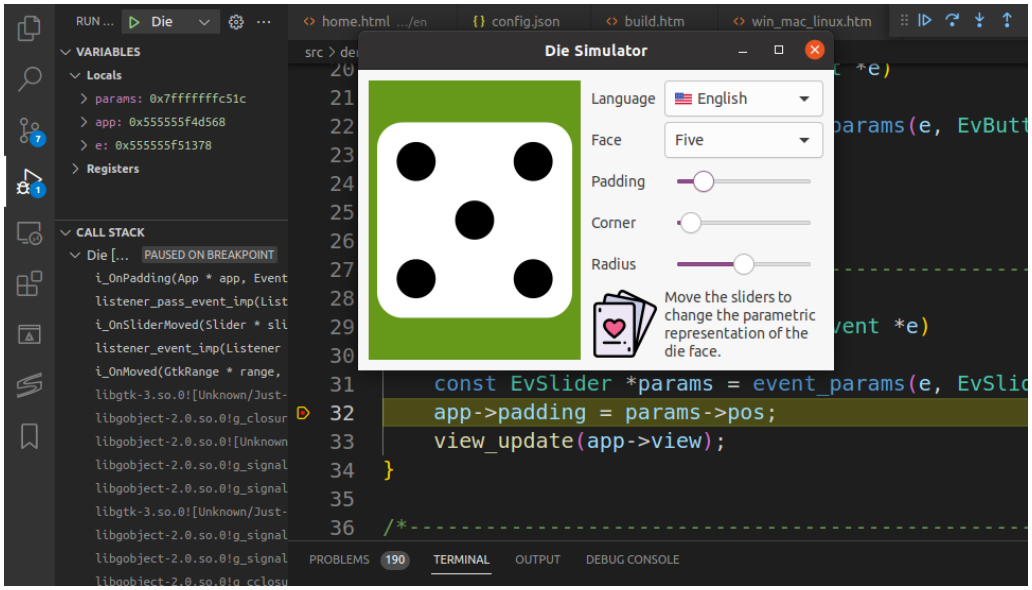


Figura 8.21: Depurando la aplicación *Die* desde Visual Studio Code.

8.4. Configuraciones

Una aplicación NAppGUI se puede compilar en tres configuraciones diferentes, en función del nivel de depuración que necesitemos.

- *Debug*: Incluye información de depuración en los binarios y no se realizan optimizaciones del código. Es la versión para el desarrollador.

- *Release*: Se elimina la información de depuración y se realizan todas las optimizaciones posibles. Es la versión para el usuario.
- *ReleaseWithAssert*: Es la versión de Release, pero dejando activas las sentencias *"Asserts"* (Página 155). Está dirigida al usuario final, pero en casos donde sea necesario obtener información detallada de posibles anomalías, a costa de una bajada del rendimiento global del programa.

Tanto Visual Studio como Xcode son entornos multi-configuración, es decir, podemos alternar entre una y otra directamente desde el propio editor. En Visual Studio tenemos un desplegable en la parte superior del editor (Figura 8.22).

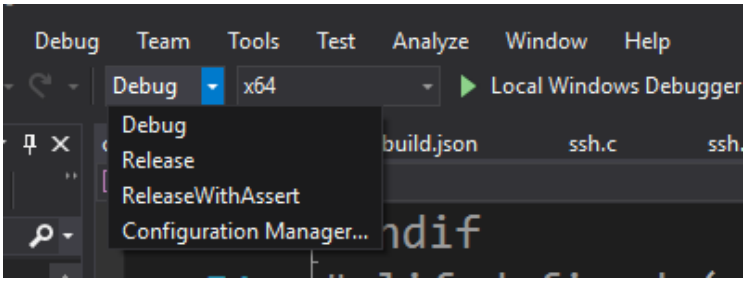


Figura 8.22: Cambio de configuración en Visual Studio.

En Xcode está un poco más escondido. Hacemos `Product->Scheme->Edit Scheme`. Aparecerá una ventana emergente. Seleccionamos `Run->Info->Build Configuration` (Figura 8.23).

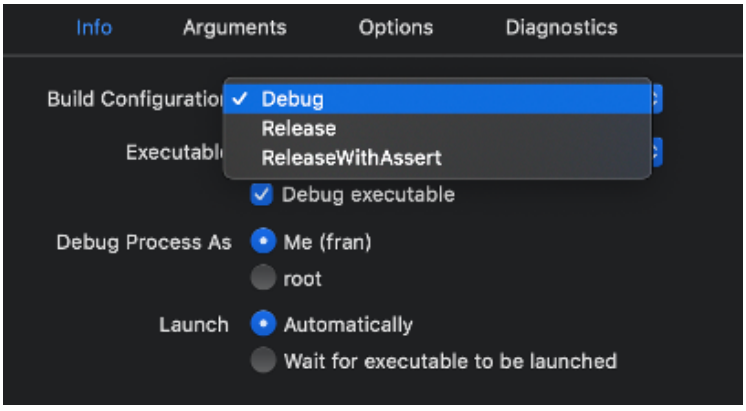


Figura 8.23: Cambio de configuración en Xcode.

Desafortunadamente, `Unix make` no soporta múltiples configuraciones. Esto nos obliga a introducir la propiedad `CMAKE_BUILD_CONFIG` (Figura 8.24) para establecer la configuración en CMake antes de generar los `Makefile`. Deberemos volver a lanzar `cmake -S ./src -B ./build` si cambiamos la configuración, para que la nueva surja efecto.

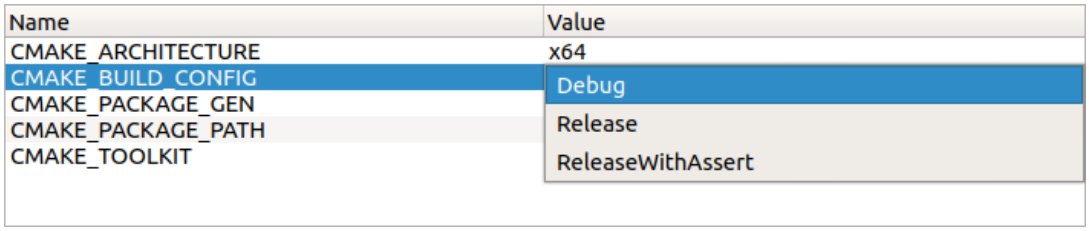


Figura 8.24: Cambio de configuración en CMake (Unix Makefile).

Capítulo 9

Crear nueva aplicación

Me considero una persona técnica que eligió un gran proyecto y una excelente manera para llevarlo a cabo.

Linus Torvalds.

En *"Inicio rápido"* (Página 5) y *"Generar binarios NAppGUI"* (Página 65) hemos visto como obtener el SDK, así como compilar y ejecutar las aplicaciones de ejemplo. También, en *"¡Hola Mundo!"* (Página 23), aprendimos la estructura básica de una aplicación basada en NAppGUI. Ha llegado el momento de crear nuestras propias aplicaciones, aprovechando los scripts CMake incluidos en la carpeta `/prj` de la distribución.

Si tu objetivo es utilizar NAppGUI como librería externa en tus proyectos, puedes saltar este capítulo.

9.1. Aplicaciones de escritorio

Para crear una nueva aplicación de escritorio, abre el archivo `/src/CMakeLists.txt` y añade la siguiente línea después de `# Your projects here!`:

src/CMakeLists.txt

```
# Your projects here!
desktopApp("MyNewApp" "myapp" "osapp" NRC_EMBEDDED)
```

Acto seguido, vuelve a generar la solución con CMake y ábrela con el IDE correspondiente:

```
cmake -S ./src -B ./build
cmake --open ./build
```

> *El comando **cmake –open** solo funciona con los generadores de Visual Studio y Xcode. En Linux tendrás que abrirlo manualmente con el editor que prefieras.*

En caso de que la solución ya estuviese abierta, es posible que el IDE te avise que han habido cambios, por ejemplo Visual Studio (Figura 9.1).

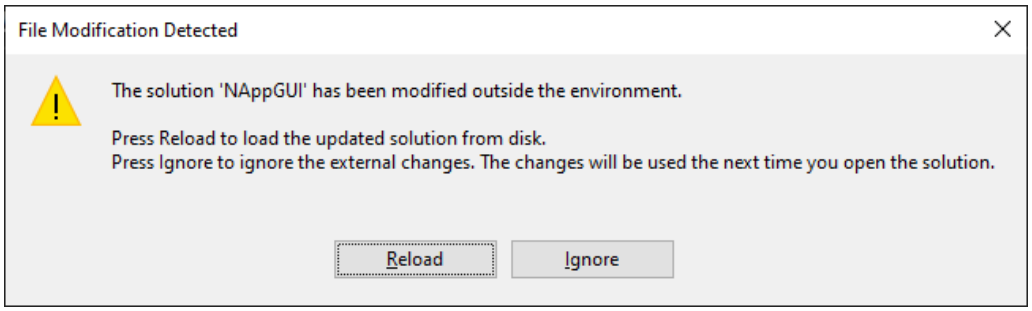


Figura 9.1: Aviso de Visual Studio de que debe recargar la solución.

Verás que CMake ha creado un nuevo proyecto llamado `MyNewApp` dentro de la solución (Figura 9.2).

Si compilas y ejecutas `MyNewApp`, te darás cuenta de que no es otra cosa que el *Hello, World!* (Figura 9.3), ya que esta es la plantilla predeterminada para cada nueva aplicación de escritorio. También se ha asignado un *"Icono de aplicaciónIcono de aplicación"* (Página 143) por defecto, pero lo podremos personalizar posteriormente.

Viendo en detalle la sintaxis del comando `desktopApp` que acabamos de añadir al `CMakeLists.txt`, tenemos:

```
desktopApp("MyNewApp" "myapp" "osapp" NRC_EMBEDDED)

desktopApp(appName path depends nrcMode)
```

- `appName`: El nombre de la aplicación.
- `path`: Ruta relativa a `/src` donde se ubicará el proyecto (en este caso `nappgui_src/src/myapp`). Se admite cualquier profundidad de ruta. Por ejemplo, `"games/myapp"` creará el proyecto en `nappgui_src/src/games/myapp` y `"demo/games/myapp"` en `nappgui_src/src/demo/games/myapp`.

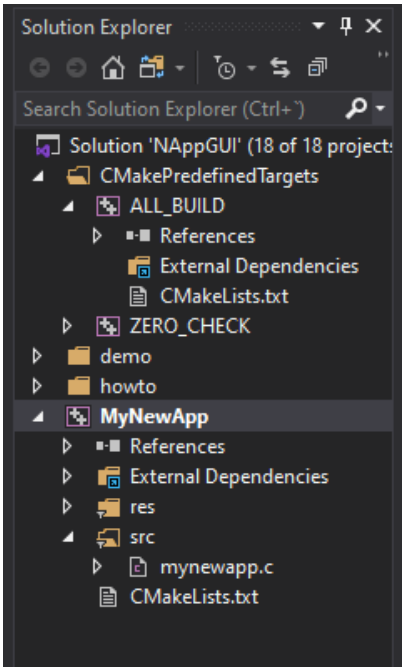


Figura 9.2: Proyecto `MyNewApp` recién añadido a la solución.

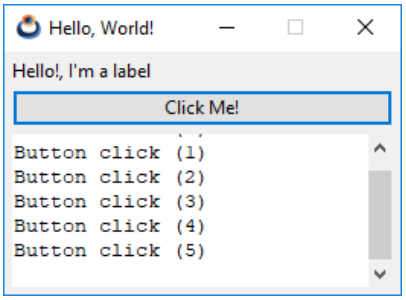


Figura 9.3: Resultado de compilar y ejecutar `MyNewApp`.

- `depends`: Librerías de las que depende la aplicación. Como mínimo tendrás que incluir `osapp` ya que estamos ante una aplicación de escritorio. Si la aplicación necesitara dependencias adicionales, como librerías creadas por ti mismo, las escribiremos a continuación separadas por punto y coma (pe. `"osapp;physics;render"`). En *"Crear nueva librería"* (Página 109) tienes un ejemplo de aplicaciones con dependencias. **Importante:** Solo necesitas indicar las dependencias de primer nivel. CMake añadirá recursivamente las dependencias de las dependencias.

- `nrcMode`: Cómo se administrarán los recursos de la aplicación. Por el momento, indicamos `NRC_EMBEDDED`. Profundizaremos en ellos en el capítulo *"Recursos"* (Página 131).

Puedes crear tantas aplicaciones dentro de la misma solución como quieras. Solo debes que repetir el proceso, añadiendo nuevos `desktopApp()` al script `CMakeLists.txt`.

9.2. Añadir archivos

Volviendo al proyecto `MyNewApp`, vemos que por defecto solo se crea un archivo de código fuente (`mynewapp.c`) que contiene toda la aplicación. Es muy probable que quieras dividir el código entre diferentes archivos. Crea un par de nuevos archivos `myfunc.c` y `myfunc.h` dentro de `nappgui_src/src/myapp` desde el IDE o directamente desde el explorador. Ábrelos y añade estas líneas:

myfunc.h

```
// Example of new header

#include "core.hxx"

real32_t myadd_func(real32_t a, real32_t b);
```

myfunc.c

```
// Example of new c file

#include "myfunc.h"

real32_t myadd_func(real32_t a, real32_t b)
{
    return a + b;
}
```

Abre `mynewapp.c` y edita la función `i_OnButton`.

mynewapp.c

```
...
static void i_OnButton(App *app, Event *e)
{
    real32_t res = myadd_func(56.4f, 23.3f);
    textview_printf(app->text, "Button click (%d-%.2f)\n", app->clicks, res);
    app->clicks += 1;
    unref(e);
}
...
```

Vuelve a re-generar la solución con `cmake -S ./src -B ./build`. El IDE, Visual Studio en este caso, nos vuelve a informar que han habido cambios en el proyecto `MyNewApp` (Figura 9.4). Simplemente presiona `[Reload All]`.

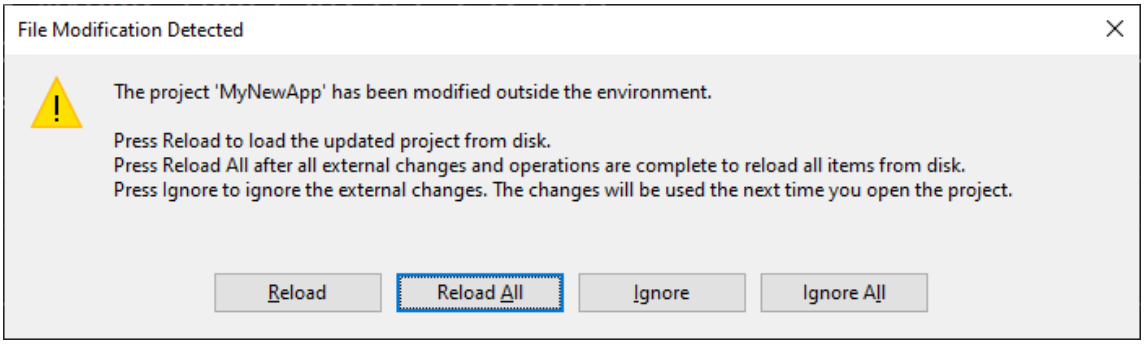


Figura 9.4: Visual Studio advierte sobre nuevos archivos. Pulsa `[Reload All]`.

Vuelve a compilar y ejecutar `MyNewApp` para ver los cambios que acabas de realizar. Puedes crear tantos archivos y subcarpetas dentro del directorio `src/myapp` como necesites para organizar mejor tu código. Recuerda siempre ejecutar `cmake -S ./src -B ./build` cada vez que añadas o elimines archivos del proyecto. CMake actualizará la solución "clonando" la estructura de directorios dentro del proyecto (`MyNewApp` en este caso).

Llegados a este punto te recomendamos que dediques algo de tiempo a investigar, compilar y probar los ejemplos de las carpetas `demo` y `howto`.

9.3. Aplicaciones por línea de comandos

De forma similar a las aplicaciones de escritorio vistas anteriormente, podrás crear aplicaciones de consola. Abre `/src/CMakeLists.txt` y añade esta línea después de `# Your projects here!`:

src/CMakeLists.txt

```
# Your projects here!
commandApp("myutil" "utils/myutil" "core" NRC_NONE)
```

Al regenerar la solución con `cmake -S ./src -B ./build`, Visual Studio te alertará que debes recargar la solución, de la misma forma que ocurrió con nuestra aplicación anterior. Se habrá creado un nuevo proyecto en `nappgui_src/src/utils/myutil`, pero esta vez si lo compilas y ejecutas no aparecerá ninguna ventana. Tan solo verás un mensaje en la consola de Visual Studio:

```
Hello world!
```

Si abres `myutil.c` encontrarás el código que ha generado la salida anterior:

```
/* NAppGUI Console Application */
```

```c
#include "coreall.h"

int main(int argc, char *argv[])
{
    unref(argc);
    unref(argv);
    core_start();
    bstd_printf("Hello world!\n");
    core_finish();
    return 0;
}
```

Que es la típica plantilla de un programa en C, a la que se le ha incluido el soporte de la librería *core*. A partir de aquí, ya podemos modificar el código y compilar. CMake ya lo configuró todo por nosotros. Volvamos al `src/CMakeLists.txt` para repasar la línea anterior:

src/CMakeLists.txt

```
commandApp("myutil" "utils/myutil" "core" NRC_NONE)

commandApp(appName path depends nrcMode)
```

- `appName`: El nombre de la aplicación.
- `path`: Ruta relativa a `/src` donde se ubicará el proyecto (en este caso `nappgui_src /src/utils/myutil`).
- `depends`: Dependencias. Una aplicación de línea de comandos no requiere ninguna dependencia "mínima", como ocurría con las aplicaciones de escritorio. Recomendamos incluir una dependencia con *"Core"* (Página 191) ("core") ya que contiene gran variedad de funciones que nos pueden facilitar la tarea. No obstante, puedes establecer *"Osbs"* (Página 168) ("osbs") o incluso *"Sewer"* (Página 151) ("sewer") como requisitos mínimos.
- `nrcMode`: `NRC_NONE`, `NRC_EMBEDDED` o `NRC_PACKED`. *"Distribución de recursosDistribución de recursos"* (Página 140).

Ni que decir tiene que podemos añadir nuevos archivos y subcarpetas al proyecto de forma similar a como lo hicimos en aplicaciones de escritorio.

9.4. Estándar C/C++

NAppGUI se ha creado, en su práctica totalidad, en lenguaje C90, añadiendo los enteros de tipo fijo `uint32_t`, `int16_t`, ... (`<stdint.h>`) de C99. Para ciertas partes del proyecto se ha utilizado C++98, pero siempre encapsulado bajo una interfaz en C90. Por

lo tanto, **se puede crear una aplicación o librería utilizando únicamente C90**, lo que proporciona gran portabilidad entre plataformas y compiladores.

Por lo general, los compiladores permiten comprobar que el código se ajuste a ciertos estándares de C/C++, emitiendo advertencias o errores cuando no sea así. Por defecto, cada nuevo proyecto creado con `desktopApp()` o `commandApp()` establecerá `C90` y `C++98` como estándares. De esta forma, serán compatibles con toda la lista de *"Compiladores e IDEs"* (Página 73) soportados por NAppGUI.

No obstante, es posible que quieras utilizar un estándar más moderno para tus nuevos proyectos. En este caso, deberás indicarlo al final de los comandos `desktopApp` o `commandApp`:

```
desktopApp("MyNewApp" "myapp" "osapp" NRC_EMBEDDED "C17;C++17")
```

En este caso indicaremos que la aplicación `MyNewApp` utilizará los estándares `C17` y `C++17` en lugar de `C90` y `C++98`, que son los valores por defecto.

- Para el compilador de C, las opciones serán: `C90`, `C99`, `C11`, `C17` y `C23`.
- Para el compilador de C++, las opciones serán: `C++98`, `C++11`, `C++14`, `C++17`, `C++20` y `C++23`.

Si el compilador no soporta la versión del lenguaje indicada, se establecerá la mayor soportada. Es responsabilidad del programador utilizar los compiladores apropiados al estándar elegido.

Capítulo 10

Crear nueva librería

Lo único que debes saber absolutamente es donde está ubicada la librería.

Albert Einstein

El uso de librerías nos va a permitir compartir código común entre varios proyectos. Sirva como ejemplo el SDK de NAppGUI, que se ha organizado en varias librerías de enlace estático o dinámico. Por ejemplo *"Core"* (Página 191) implementa funciones relacionadas con strings, streams y estructuras de datos que pueden ser reutilizadas en diferentes aplicaciones.

10.1. Librerías estáticas

Para ilustrar el uso de librerías, vamos a utilizar dos aplicaciones incluidas en los ejemplos de NAppGUI: `Die` (Figura 10.1) y `Dice` (Figura 10.2). En ambas se debe poder dibujar la silueta de un dado.

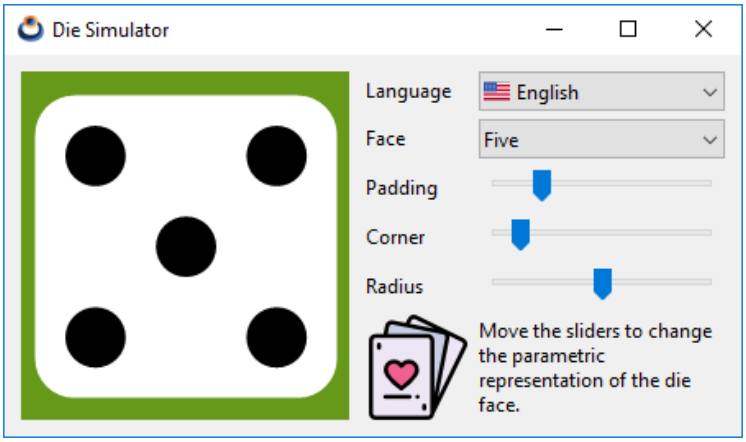


Figura 10.1: Aplicación *Die*.

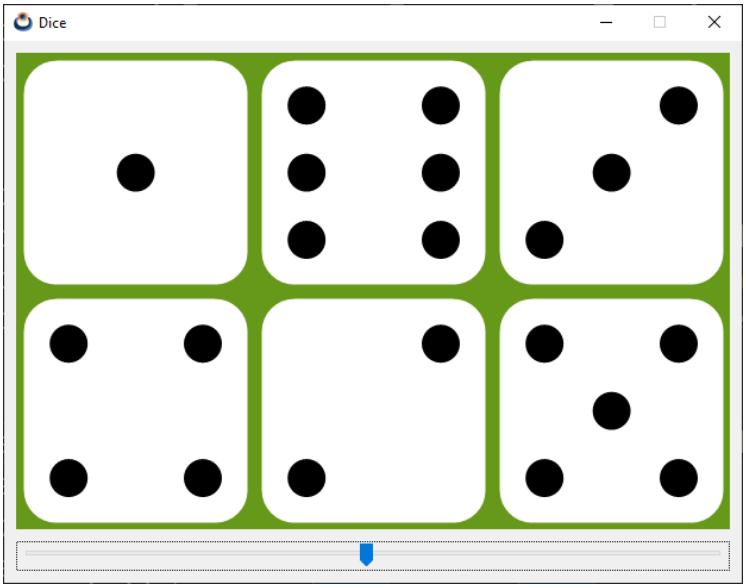


Figura 10.2: Aplicación *Dice*.

El código fuente ambas aplicaciones lo tienes en src/demo/die[a] *y src/demo/dice*[b].

[a] https://github.com/frang75/nappgui_src/tree/main/src/demo/die
[b] https://github.com/frang75/nappgui_src/tree/main/src/demo/dice

No es muy complicado intuir que podríamos reutilizar la rutina de dibujo paramétrico en ambos proyectos. Una forma de hacerlo sería copiando dicha rutina desde *Die* a *Dice*, pero esto no es lo más aconsejable ya que tendríamos dos versiones del mismo código

que mantener. Otra opción, la más sensata, es mover la función de dibujo a una librería y vincularla en ambas aplicaciones. Esto es muy sencillo de realizar gracias, de nuevo, a CMake. Si abrimos el `src/CMakeLists.txt` veremos estas tres líneas:

```
staticLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
desktopApp("Die" "demo/die" "osapp;casino" NRC_EMBEDDED)
desktopApp("Dice" "demo/dice" "osapp;casino" NRC_EMBEDDED)
```

Donde hemos utilizado el comando `staticLib()`, que es análogo a `desktopApp()`.

```
staticLib(libName path depends nrcMode)
```

- `libName`: El nombre de la librería.
- `path`: Ruta relativa a `/src` donde se ubicará el proyecto (en este caso `nappgui_src /src/demo/casino`). Al igual que vimos al crear nuevas aplicaciones, se admite cualquier profundidad de ruta.
- `depends`: Dependencias de la librería. Al igual que en aplicaciones, solo es necesario indicar las de más alto nivel (*draw2d* en este caso). Cada librería se encarga de enlazar con las que tiene por debajo. *draw2d* incluirá *geom2d* y así sucesivamente. En *"NAppGUI API"* (Página 147) tienes el grafo completo de dependencias.
- `nrcMode`: Cómo se administrarán los recursos de la librería. Por el momento, indicamos `NRC_EMBEDDED`. Profundizaremos en ellos en el capítulo *"Recursos"* (Página 131).
- `standard`: Opcionalmente se puede indicar el *"Estándar C/C++Estándar C/C++"* (Página 106).

Tanto *Die* como *Dice* han añadido una dependencia con *casino* (Figura 10.3) por medio del parámetro `depends` del comando `desktopApp()`. De esta forma CMake sabe que debe enlazar, además de *osapp*, la librería *casino* que es donde se encuentra código común de ambos proyectos.

Al re-generar con `cmake -S ./src -B ./build`, se añade la librería *casino* a nuestra solución, así como un vínculo a ella en ambas aplicaciones (Figura 10.4).

Al igual que ocurría al crear una nueva aplicación, cuando se crea una librería aparecen varios archivos por defecto, que son:

`casino.def`: Fichero que definirá la macro `_casino_api` necesaria para la exportación de símbolos. Más información en *"Símbolos y visibilidadSímbolos y visibilidad"* (Página 126).

Listado 10.1: demo/casino/casino.def

```
/* casino library import/export */

#if defined(NAPPGUI_SHARED)
```

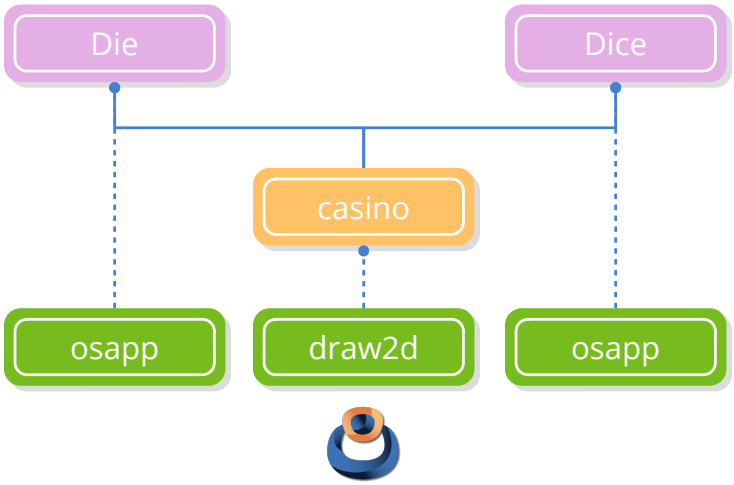


Figura 10.3: Árbol de dependencias de las aplicaciones, centrado en la librería *casino*.

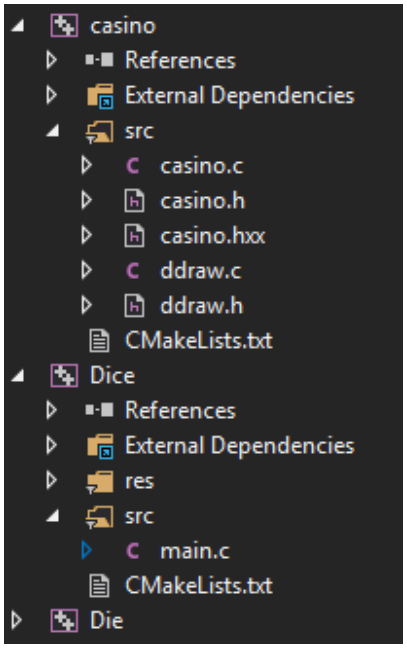


Figura 10.4: Librería estática *casino*, integrada en la solución.

```
    #if defined(NAPPGUI_BUILD_CASINO_LIB)
        #define NAPPGUI_CASINO_EXPORT_DLL
    #else
        #define NAPPGUI_CASINO_IMPORT_DLL
    #endif
#endif

#if defined(__GNUC__)
    #if defined(NAPPGUI_CASINO_EXPORT_DLL)
        #define _casino_api __attribute__((visibility("default")))
```

```
    #else
        #define _casino_api
    #endif
#elif defined(_MSC_VER)
    #if defined(NAPPGUI_CASINO_IMPORT_DLL)
        #define _casino_api __declspec(dllimport)
    #elif defined(NAPPGUI_CASINO_EXPORT_DLL)
        #define _casino_api __declspec(dllexport)
    #else
        #define _casino_api
    #endif
#else
    #error Unknown compiler
#endif
```

`casino.hxx`: Aquí definiremos tipos públicos, como `enum` o `struct`. Por el momento `casino` no contiene tipos públicos.

Listado 10.2: demo/casino/casino.hxx

```
/* casino */

#ifndef __CASINO_HXX__
#define __CASINO_HXX__

#include <draw2d/draw2d.hxx>
#include "casino.def"

/* TODO: Define data types here */

#endif
```

`casino.h`: Fichero de cabecera. Aquí escribiremos la declaración de funciones generales. Por defecto, CMake crea dos: `casino_start()` y `casino_finish()`, donde implementaríamos código global de inicio y finalización de la librería, si fuera necesario.

Listado 10.3: demo/casino/casino.h

```
/* casino */

#include "casino.hxx"

__EXTERN_C

_casino_api void casino_start(void);

_casino_api void casino_finish(void);

__END_C
```

`casino.c`: Implementación de las funciones generales.

Listado 10.4: demo/casino/casino.c

```
/* casino */

#include "casino.h"

/*----------------------------------------------------------------------------*/

void casino_start(void)
{
    /*TODO: Implement library initialization code here */
}

/*----------------------------------------------------------------------------*/

void casino_finish(void)
{
    /*TODO: Implement library ending code here */
}
```

Posteriormente creamos dos nuevos archivos dentro de `src/demo/casino`, `ddraw.c` y `ddraw.h` donde implementaremos la función de dibujo a compartir. Ya vimos como *"Añadir archivosAñadir archivos"* (Página 104).

Listado 10.5: demo/casino/ddraw.h

```
/* Die drawing */

#include "casino.hxx"

_casino_api void die_draw(
                        DCtx *ctx,
                        const real32_t x,
                        const real32_t y,
                        const real32_t width,
                        const real32_t height,
                        const real32_t padding,
                        const real32_t corner,
                        const real32_t radius,
                        const uint32_t face);

_casino_api extern const real32_t kDEF_PADDING;

_casino_api extern const real32_t kDEF_CORNER;

_casino_api extern const real32_t kDEF_RADIUS;
```

Listado 10.6: demo/casino/ddraw.c

```c
/* Die drawing */

#include "ddraw.h"
#include <draw2d/draw2dall.h>

/*----------------------------------------------------------------------------*/

static const real32_t i_MAX_PADDING = 0.2f;
const real32_t kDEF_PADDING = .15f;
const real32_t kDEF_CORNER = .15f;
const real32_t kDEF_RADIUS = .35f;

/*----------------------------------------------------------------------------*/

void die_draw(DCtx *ctx, const real32_t x, const real32_t y, const real32_t
    ↪ width, const real32_t height, const real32_t padding, const real32_t
    ↪ corner, const real32_t radius, const uint32_t face)
{
    color_t white = color_rgb(255, 255, 255);
    color_t black = color_rgb(0, 0, 0);
    real32_t dsize, dx, dy;
    real32_t rc, rr;
    real32_t p1, p2, p3;

    dsize = width < height ? width : height;
    dsize -= bmath_floorf(2.f * dsize * padding * i_MAX_PADDING);
    dx = x + .5f * (width - dsize);
    dy = y + .5f * (height - dsize);
    rc = dsize * (.1f + .3f * corner);
    rr = dsize * (.05f + .1f * radius);
    p1 = 0.5f * dsize;
    p2 = 0.2f * dsize;
    p3 = 0.8f * dsize;

    draw_fill_color(ctx, white);
    draw_rndrect(ctx, ekFILL, dx, dy, dsize, dsize, rc);
    draw_fill_color(ctx, black);

    if (face == 1 || face == 3 || face == 5)
        draw_circle(ctx, ekFILL, dx + p1, dy + p1, rr);

    if (face != 1)
    {
        draw_circle(ctx, ekFILL, dx + p3, dy + p2, rr);
        draw_circle(ctx, ekFILL, dx + p2, dy + p3, rr);
    }

    if (face == 4 || face == 5 || face == 6)
    {
        draw_circle(ctx, ekFILL, dx + p2, dy + p2, rr);
        draw_circle(ctx, ekFILL, dx + p3, dy + p3, rr);
```

```
    }

    if (face == 6)
    {
        draw_circle(ctx, ekFILL, dx + p2, dy + p1, rr);
        draw_circle(ctx, ekFILL, dx + p3, dy + p1, rr);
    }
}
```

¿Que significa realmente que *Die* y *Dice* tengan una dependencia con *casino*? Que a partir de ahora ninguna de ellas se podrá compilar si hay algún error en el código de *casino*, ya que es un módulo fundamental para ambas. Dentro del proyecto de compilación (Visual Studio, Xcode, Makefile, etc) han ocurrido varias cosas:

- Las dos aplicaciones saben donde está ubicada *casino*, por lo que pueden hacer `# include "casino.h"` sin preocuparse de su ubicación.
- El código binario de las funciones de *casino* se incluirá en cada ejecutable en el proceso de enlazado. CMake ya se encargó de vincular la librería con los ejecutables.
- Cualquier cambio realizado en *casino* obligará a recompilar las aplicaciones debido al punto anterior. De nuevo, el proyecto de compilación sabrá como hacerlo de la forma más eficiente posible. Tan solo deberemos volver a lanzar `cmake --build ./build` para actualizar todos los binarios.

Como ya indicamos antes, *casino* también tiene una dependencia con *"Draw2D"* (Página 262), la librería de dibujo vectorial de NAppGUI. A su vez *draw2d* depende de *geom2d* y así sucesivamente, hasta llegar a *sewer*, el paquete más bajo del SDK. Cuando desarrolles una nueva librería deberías vincularla con el menor número posible de dependencias o, dicho de otra forma, con las librerías de menor nivel dentro de la jerarquía que incluyan la funcionalidad necesaria. Esto mejorará la compilación y distribución, además de constituir una muy buena práctica de trabajo.

10.2. Librerías dinámicas

Las librerías dinámicas son, en esencia, lo mismo que las estáticas. Lo único que cambia es la forma con la que se vinculan al ejecutable (Figura 10.5). En el enlace estático, el código de la librería se añade al propio ejecutable, por lo que el tamaño de este último crecerá. En el enlace dinámico el código de la librería se distribuye en su propio archivo (`.dll`, `.so`, `.dylib`) y se carga justamente antes que el programa ejecutable.

El proceso para crear librerías dinámicas es exactamente igual que las estáticas. Lo único que tenemos que hacer es sustituir el comando `staticLib()` por `dynamicLib()` en `/src/CMakeLists.txt`.

```
dynamicLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
```

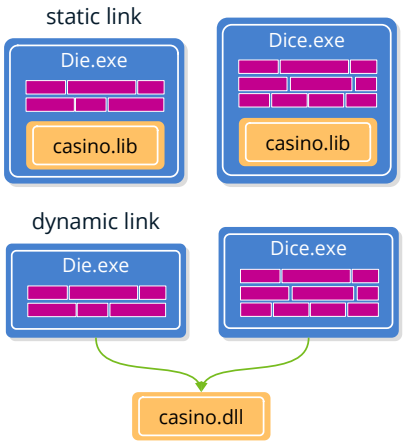


Figura 10.5: Enlace estático o dinámico de *casino*.

```
desktopApp("Die" "demo/die" "osapp;casino" NRC_EMBEDDED)
desktopApp("Dice" "demo/dice" "osapp;casino" NRC_EMBEDDED)
```

```
dynamicLib(libName path depends nrcMode)
```

Los parámetros son exactamente los mismos que en `staticLib`:

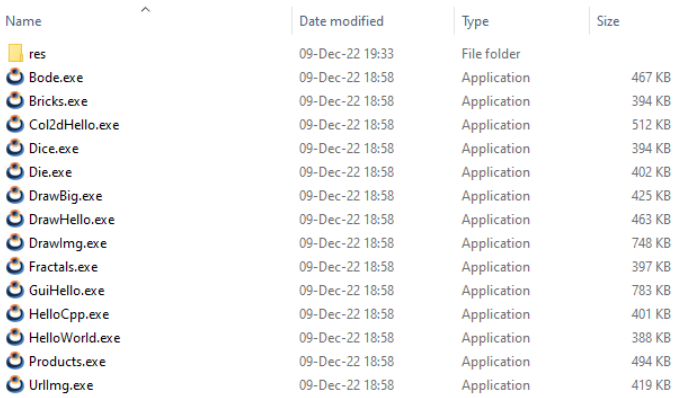
- `libName`: El nombre de la librería.
- `path`: Ruta relativa a `/src` donde se ubicará el proyecto.
- `depends`: Dependencias de la librería.
- `nrcMode`: Cómo se administrarán los recursos de la librería.
- `standard`: Opcionalmente se puede indicar el *"Estándar C/C++Estándar C/C++"* (Página 106).

Es totalmente válido crear la versión estática y dinámica de una librería. La única condición es la de renombrar una de ellas, ya que **no es posible tener dos proyectos con el mismo nombre en la misma solución**. A continuación, hemos creado dos versiones de `casino`, vinculando cada una de ellas con un ejecutable.

```
staticLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
dynamicLib("casino_d" "demo/casino" "draw2d" NRC_EMBEDDED)

// Use the static version of 'casino'
desktopApp("Die" "demo/die" "osapp;casino" NRC_EMBEDDED)

// Use the dynamic version of 'casino'
desktopApp("Dice" "demo/dice" "osapp;casino_d" NRC_EMBEDDED)
```

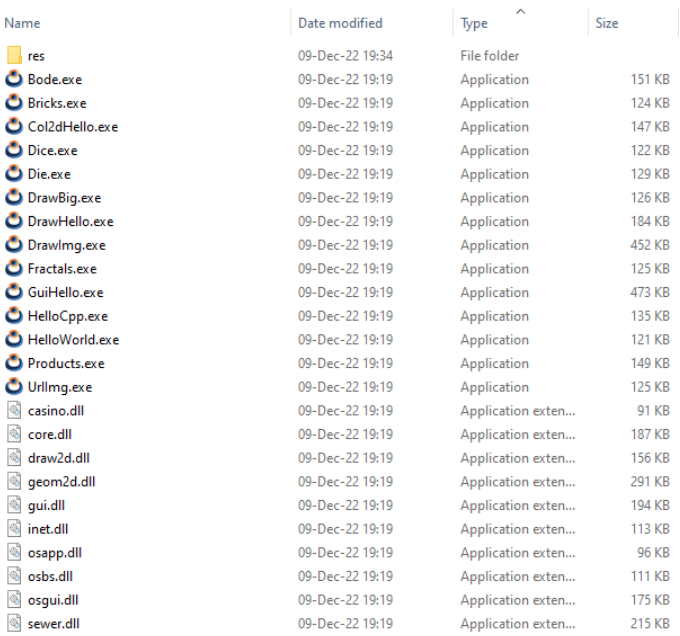
10.2.1. Ventajas de las DLLs

Como hemos podido intuir en el ejemplo anterior, utilizando DLLs reduciremos el tamaño de los ejecutables, agrupando el código binario común (Figura 10.6), (Figura 10.7). Esto es precisamente lo que hacen los sistemas operativos. Por ejemplo, `Die.exe` necesitará acceder, en última instancia, a las funciones del API de Windows. Si todas las aplicaciones tuviesen que enlazar de forma estática los binarios de Windows, su tamaño crecería desmesuradamente y se desperdiciaría mucho espacio dentro del sistema de archivos.



Name	Date modified	Type	Size
res	09-Dec-22 19:33	File folder	
Bode.exe	09-Dec-22 18:58	Application	467 KB
Bricks.exe	09-Dec-22 18:58	Application	394 KB
Col2dHello.exe	09-Dec-22 18:58	Application	512 KB
Dice.exe	09-Dec-22 18:58	Application	394 KB
Die.exe	09-Dec-22 18:58	Application	402 KB
DrawBig.exe	09-Dec-22 18:58	Application	425 KB
DrawHello.exe	09-Dec-22 18:58	Application	463 KB
DrawImg.exe	09-Dec-22 18:58	Application	748 KB
Fractals.exe	09-Dec-22 18:58	Application	397 KB
GuiHello.exe	09-Dec-22 18:58	Application	783 KB
HelloCpp.exe	09-Dec-22 18:58	Application	401 KB
HelloWorld.exe	09-Dec-22 18:58	Application	388 KB
Products.exe	09-Dec-22 18:58	Application	494 KB
UrlImg.exe	09-Dec-22 18:58	Application	419 KB

Figura 10.6: Los ejemplos de programación ocupan **6.52 Mb** en su versión estática.



Name	Date modified	Type	Size
res	09-Dec-22 19:34	File folder	
Bode.exe	09-Dec-22 19:19	Application	151 KB
Bricks.exe	09-Dec-22 19:19	Application	124 KB
Col2dHello.exe	09-Dec-22 19:19	Application	147 KB
Dice.exe	09-Dec-22 19:19	Application	122 KB
Die.exe	09-Dec-22 19:19	Application	129 KB
DrawBig.exe	09-Dec-22 19:19	Application	126 KB
DrawHello.exe	09-Dec-22 19:19	Application	184 KB
DrawImg.exe	09-Dec-22 19:19	Application	452 KB
Fractals.exe	09-Dec-22 19:19	Application	125 KB
GuiHello.exe	09-Dec-22 19:19	Application	473 KB
HelloCpp.exe	09-Dec-22 19:19	Application	135 KB
HelloWorld.exe	09-Dec-22 19:19	Application	121 KB
Products.exe	09-Dec-22 19:19	Application	149 KB
UrlImg.exe	09-Dec-22 19:19	Application	125 KB
casino.dll	09-Dec-22 19:19	Application exten...	91 KB
core.dll	09-Dec-22 19:19	Application exten...	187 KB
draw2d.dll	09-Dec-22 19:19	Application exten...	156 KB
geom2d.dll	09-Dec-22 19:19	Application exten...	291 KB
gui.dll	09-Dec-22 19:19	Application exten...	194 KB
inet.dll	09-Dec-22 19:19	Application exten...	113 KB
osapp.dll	09-Dec-22 19:19	Application exten...	96 KB
osbs.dll	09-Dec-22 19:19	Application exten...	111 KB
osgui.dll	09-Dec-22 19:19	Application exten...	175 KB
sewer.dll	09-Dec-22 19:19	Application exten...	215 KB

Figura 10.7: Los ejemplos de programación ocupan **4.08 Mb** en su versión dinámica.

Otra gran ventaja de las DLLs es el ahorro de memoria en tiempo de ejecución. Por ejemplo, si cargamos `Die.exe`, se cargará `casino.dll` al mismo tiempo. Pero si después

cargamos `Dice.exe`, ambas compartirán la copia de `casino.dll` existente en memoria. Sin embargo, con enlace estático, existirían dos copias de `casino.lib` en la memoria RAM: Una integrada en `Die.exe` y otra en `Dice.exe`.

10.2.2. Desventajas de las DLLs

El principal inconveniente del uso de DLLs es la incompatibilidad que puede presentarse entre las diferentes versiones de una librería. Supongamos que lanzamos una primera versión de los tres productos:

```
casino.dll        102,127 (v1)
Die.exe            84,100 (v1)
Dice.exe           73,430 (v1)
```

Unos meses después, lanzamos una nueva versión de la aplicación `Dice.exe` que implica cambios en `casino.dll`. En ese caso, la distribución de nuestra *suite* quedaría así:

```
casino.dll        106,386  (v2)*
Die.exe            84,100  (v1)?
Dice.exe           78,491  (v2)*
```

Si no hemos sido muy cuidadosos, es muy probable que `Die.exe` ya no funcione al no ser compatible con la nueva versión de la DLL. Este problema trae de cabeza a muchos desarrolladores y ha sido bautizado como *DLL Hell*[1]. Dado que en este ejemplo trabajamos sobre un entorno "controlado" podríamos solucionarlo sin demasiados problemas, creando una nueva versión de todas la aplicaciones funcionando bajo `casino.dll(v2)`.

```
casino.dll        106,386  (v2)
Die.exe            84,258  (v2)
Dice.exe           78,491  (v2)
```

Esto no siempre será posible. Supongamos ahora que nuestra compañía desarrolla tan solo `casino.dll` y son terceras empresas las que trabajan en los productos finales. Ahora cada producto tendrá sus ciclos de producción y distribución (entorno no controlado) por lo que, para evitar problemas, cada compañía incluirá una copia de la versión concreta de la DLL con la que funciona su producto. Esto podría dar lugar al siguiente escenario:

```
/Apps/Die
casino.dll        114,295  (v5)
Die.exe            86.100  (v8)

/Apps/Dice
casino.dll        106,386  (v2)
Dice.exe           72,105  (v1)
```

[1] `https://en.wikipedia.org/wiki/DLL_Hell`

Viendo esto intuimos de que las bondades del uso de DLLs ya no lo son tanto, sobre todo en lo relativo a la optimización del espacio y tiempos de carga. El caso es que se puede agravar aún más. Normalmente, las librerías se escriben para que sean lo más genéricas posible y puedan dar servicio a muchas aplicaciones. En muchas ocasiones, una aplicación concreta utiliza sólo unas pocas funciones cada librería con las que enlaza. Utilizando librerías estáticas, se puede reducir considerablemente el tamaño del ejecutable (Figura 10.8), ya que el enlazador sabe perfectamente que funciones concretas utiliza la aplicación y añade el código estrictamente necesario. Sin embargo, utilizando DLLs, debemos distribuir la librería completa por muy pocas funciones que utilice el ejecutable (Figura 10.9). En este caso, se está desperdiciando espacio y aumentando innecesariamente los tiempos de carga de la aplicación.

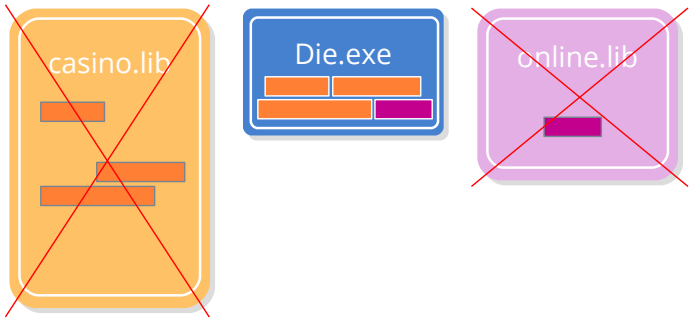


Figura 10.8: Con librerías estáticas se optimiza el espacio y tiempos de carga de esta aplicación.

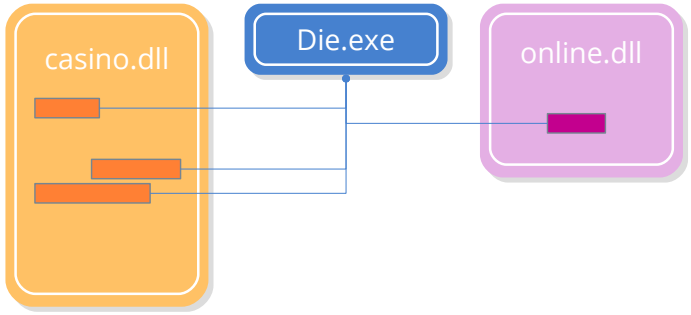


Figura 10.9: Con librerías dinámicas esta aplicación ocupa más de lo que debería y aumentan sus tiempos de carga.

10.2.3. Comprobar vínculos con DLLs

Cuando se lanza un ejecutable, por ejemplo `Die.exe`, se cargan en memoria todas las librerías dinámicas vinculadas con él (en el caso de que no existan previamente). Si hay algún problema durante dicha carga, **el ejecutable no podrá arrancar** y el sistema operativo mostrará algún tipo error.

Vínculos en Windows

Windows mostrará un aviso de error (Figura 10.10) cuando no pueda cargar una DLL asociada a un ejecutable.

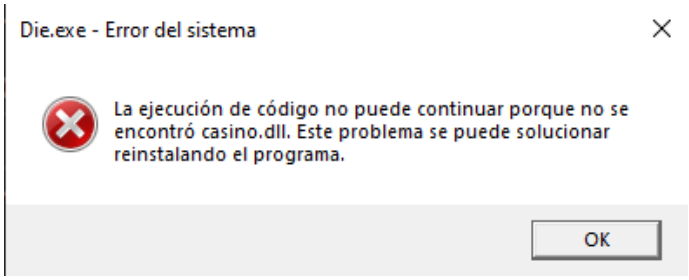


Figura 10.10: Error en la carga de la DLL `casino`.

Si queremos ver que DLLs están vinculadas con un ejecutable, utilizaremos el comando `dumpbin`.

```
dumpbin /dependents Die.exe

Dump of file Die.exe

File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    casino.dll
    KERNEL32.dll
    USER32.dll
    GDI32.dll
    SHELL32.dll
    COMDLG32.dll
    gdiplus.dll
    SHLWAPI.dll
    COMCTL32.dll
    UxTheme.dll
    WS2_32.dll
```

Vemos, al principio, la dependencia con `casino.dll`. Las demás son librerías de Windows relacionadas con el kernel y la interfaz de usuario. En el caso de que hagamos un enlace estático de `casino`:

```
staticLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
desktopApp("Die" "demo/die" "osapp;casino" NRC_EMBEDDED)
```

```
dumpbin /dependents Die.exe

Dump of file Die.exe

File Type: EXECUTABLE IMAGE

  Image has the following dependencies:

    KERNEL32.dll
```

```
    USER32.dll
    GDI32.dll
    SHELL32.dll
    COMDLG32.dll
    gdiplus.dll
    SHLWAPI.dll
    COMCTL32.dll
    UxTheme.dll
    WS2_32.dll
```

Ya no aparece `casino.dll`, al haber sido enlazada de forma estática dentro de `Die.exe`.

Vínculos en Linux

En Linux ocurre algo similar, obtendremos un error si no es posible cargar una librería dinámica (`*.so`).

```
:~/$ ./Die
./Die: error while loading shared libraries: libcasino.so: cannot open shared
    ↪ object file: No such file or directory
```

Para comprobar que librerías están vinculadas con un ejecutable utilizamos el comando `ldd`.

```
~/$ ldd ./Die
linux-vdso.so.1 (0x00007fff58036000)
libcasino.so => libcasino.so (0x00007f6848bf4000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f6848bba000)
libgtk-3.so.0 => /lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007f6848409000)
libgdk-3.so.0 => /lib/x86_64-linux-gnu/libgdk-3.so.0 (0x00007f6848304000)
libpangocairo-1.0.so.0 => /lib/x86_64-linux-gnu/libpangocairo-1.0.so.0 (0
    ↪ x00007f68482f2000)
libpango-1.0.so.0 => /lib/x86_64-linux-gnu/libpango-1.0.so.0 (0
    ↪ x00007f68482a3000)
libcairo.so.2 => /lib/x86_64-linux-gnu/libcairo.so.2 (0x00007f684817e000)
libgdk_pixbuf-2.0.so.0 => /lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0 (0
    ↪ x00007f6848156000)
libgio-2.0.so.0 => /lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f6847f75000)
libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0
    ↪ x00007f6847f15000)
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f6847dec000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f6847c9d000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6847aa9000)
...
```

Donde vemos que `Die` depende de `libcasino.so`. Las demás son dependencias del kernel de Linux, de la librería estándar de C y de GTK.

Vínculos en macOS: Utilizamos el comando `otool`.

```
% otool -L ./Die.app/Contents/MacOS/Die
@rpath/libcasino.dylib
/System/Library/Frameworks/Cocoa.framework/Versions/A/Cocoa
/System/Library/Frameworks/UniformTypeIdentifiers.framework/Versions/A/
    ↪ UniformTypeIdentifiers
/usr/lib/libc++.1.dylib
/usr/lib/libSystem.B.dylib
/System/Library/Frameworks/AppKit.framework/Versions/C/AppKit
/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
/System/Library/Frameworks/CoreGraphics.framework/Versions/A/CoreGraphics
/System/Library/Frameworks/CoreText.framework/Versions/A/CoreText
/System/Library/Frameworks/Foundation.framework/Versions/C/Foundation
/usr/lib/libobjc.A.dylib
```

10.2.4. Carga de DLLs en tiempo de ejecución

Hasta ahora, la importación de los símbolos de las DLLs se resuelven en tiempo de compilación o, mejor dicho, en tiempo de enlace. Esto significa que:

- Los ejecutables pueden acceder directamente a las variables globales y funciones definidas en la DLL. Volviendo al código de `Dice.exe`, tenemos:

```
#include "ddraw.h"
...
static void i_OnRedraw(App *app, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    color_t green = color_rgb(102, 153, 26);
    real32_t w = params->width / 3;
    real32_t h = params->height / 2;
    real32_t p = kDEF_PADDING;
    real32_t c = kDEF_CORNER;
    real32_t r = kDEF_RADIUS;
    draw_clear(params->ctx, green);
    die_draw(params->ctx, 0.f, 0.f, w, h, p, c, r, app->face[0]);
    die_draw(params->ctx, w, 0.f, w, h, p, c, r, app->face[1]);
    die_draw(params->ctx, 2 * w, 0.f, w, h, p, c, r, app->face[2]);
    die_draw(params->ctx, 0.f, h, w, h, p, c, r, app->face[3]);
    die_draw(params->ctx, w, h, w, h, p, c, r, app->face[4]);
    die_draw(params->ctx, 2 * w, h, w, h, p, c, r, app->face[5]);
}
```

  - Se ha realizado un `#include "ddraw.h"`, cabecera pública de `casino`.
  - Se han utilizado `die_draw()`, `kDEF_PADDING`, `kDEF_CORNER`, `kDEF_RADIUS`.

- La librería dinámica `casino.dll` se cargará de forma automática justamente antes de `Dice.exe`.

- El uso de versión estática o dinámica de `casino` no implica cambios en el código de `Dice`. Tan solo tendríamos que cambiar las dependencias dentro de `desktopApp()` y recompilar la aplicación.

```
// Source code in demo/dice has no changes

// Option 1 - Static link of casino
staticLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
desktopApp("Dice" "demo/dice" "osapp;casino" NRC_EMBEDDED)

// Option 2 - Dynamic link of casino
dynamicLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
desktopApp("Dice" "demo/dice" "osapp;casino" NRC_EMBEDDED)
```

No obstante, existe la posibilidad que de sea el programador el encargado de cargar, descargar y acceder a los símbolos de las DLLs en cualquier momento. Esto se conoce como enlace en tiempo de ejecución o enlace sin importación de símbolos. En src/demo/dice2[2] tenemos una nueva versión de `Dice`:

```
typedef void(*FPtr_ddraw)(DCtx*, const real32_t, const real32_t, const real32_t
    ↪ , const real32_t, const real32_t, const real32_t, const real32_t, const
    ↪ uint32_t);

static void i_OnRedraw(App *app, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    DLib *casino = dlib_open(NULL, "casino_d");
    FPtr_ddraw func_draw = dlib_proc(casino, "die_draw", FPtr_ddraw);
    color_t green = color_rgb(102, 153, 26);
    real32_t w = params->width / 3;
    real32_t h = params->height / 2;
    real32_t p = *dlib_var(casino, "kDEF_PADDING", real32_t);
    real32_t c = *dlib_var(casino, "kDEF_CORNER", real32_t);
    real32_t r = *dlib_var(casino, "kDEF_RADIUS", real32_t);
    draw_clear(params->ctx, green);
    func_draw(params->ctx, 0.f, 0.f, w, h, p, c, r, app->face[0]);
    func_draw(params->ctx, w, 0.f, w, h, p, c, r, app->face[1]);
    func_draw(params->ctx, 2 * w, 0.f, w, h, p, c, r, app->face[2]);
    func_draw(params->ctx, 0.f, h, w, h, p, c, r, app->face[3]);
    func_draw(params->ctx, w, h, w, h, p, c, r, app->face[4]);
    func_draw(params->ctx, 2 * w, h, w, h, p, c, r, app->face[5]);
    dlib_close(&casino);
}
```

- La línea 6 carga la librería `casino_d`.
- La línea 7 accede a la función `die_draw` definida en `casino_d`.

[2] https://github.com/frang75/nappgui_src/tree/main/src/demo/dice2

- Las líneas 11-13 acceden a variables públicas de `casino_d`.
- Las líneas 15-20 utilizan `die_draw` a través del puntero `func_draw`.
- La línea 21 descarga la librería `casino_d` de memoria.

Como vemos, esta carga en tiempo de ejecución sí que implica cambios en el código fuente, pero también trae consigo ciertas ventajas de las que podemos sacar partido.

- La librería se carga en el momento que la necesitamos, no al inicio del programa. Por esto es **muy importante** que `casino_d` no aparezca como dependencia de `Dice2`.

```
dynamicLib("casino_d" "demo/casino" "draw2d" NRC_EMBEDDED)
desktopApp("Dice2" "demo/dice2" "osapp" NRC_EMBEDDED)
```

- Podemos tener diferentes versiones de `casino` y elegir cual utilizar en tiempo de ejecución. Este es el mecanismo de funcionamiento de los *plug-ins* utilizados por muchas aplicaciones. Por ejemplo, el programa *Rhinoceros 3D* enriquece su funcionalidad gracias a nuevos comandos implementados por terceros y añadidos en cualquier momento mediante un sistema de plugins (.DLLs) (Figura 10.11).

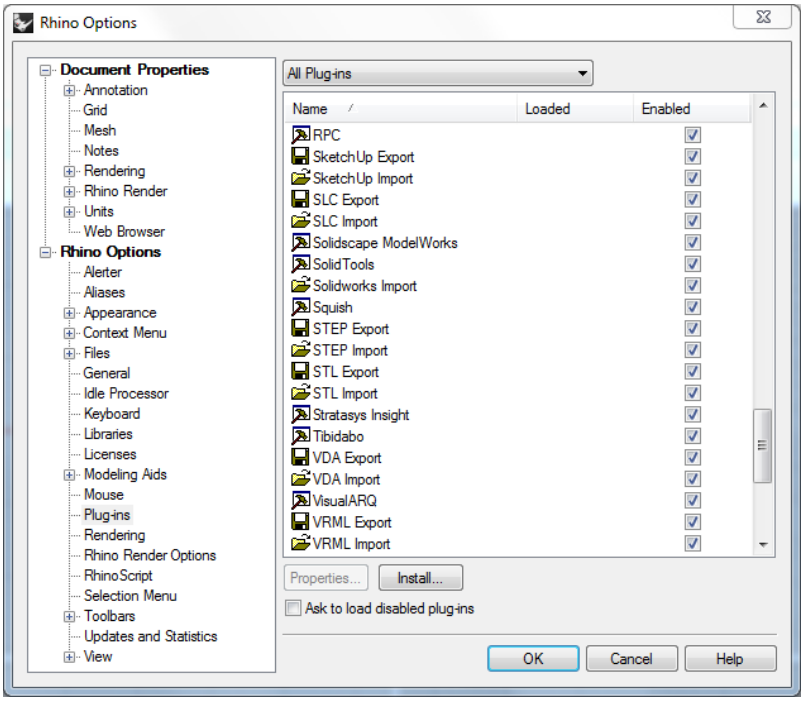


Figura 10.11: Sistema de plug-ins de Rhinoceros 3D, implementado mediante DLLs.

10.2.5. Ubicación de DLLs

Cuando el sistema operativo debe cargar una librería dinámica, sigue cierto orden de búsqueda. En sistemas Windows busca en este orden:

- El mismo directorio que el ejecutable.
- El directorio de trabajo actual.
- El directorio `%SystemRoot%\System32`.
- El directorio `%SystemRoot%`.
- Los directorios especificados en la variable de entorno `PATH`.

Por otro lado, en Linux y macOS:

- Los directorios especificados en la variable de entorno `LD_LIBRARY_PATH` (Linux) o `DYLD_LIBRARY_PATH` (macOS).
- Los directorios especificados en el ejecutable `rpath`.
- Los directorios del sistema `/lib, /usr/lib, etc`.

Aquí tenemos una gran diferencia entre Windows y Unix, ya que en estos últimos es posible añadir dentro del ejecutable directorios de búsqueda de dependencias. Esta variable se conoce como **`RPATH`** y no está disponible en Windows. Para consultar el valor del `RPATH`:

```
// In Linux
~/$ readelf -d ./Die | grep RUNPATH
 0x000000000000001d (RUNPATH)            Library runpath: [${ORIGIN}]

 // In macOS
otool -l ./Die.app/Contents/MacOS/Die
...
Load command 25
          cmd LC_RPATH
      cmdsize 40
         path @executable_path/../../.. (offset 12)
...
```

Los ejecutables generados por el `CMakeLists.txt` de NAppGUI establecen automáticamente el `RPATH` para encontrar las dependencias dinámicas en el mismo directorio que los ejecutables en Linux o los bundles en macOS.

10.3. Símbolos y visibilidad

En el proceso de enlace tras la compilación de la librería, se denomina **símbolo** a aquellos elementos que pueden generar código máquina u ocupar espacio en el binario

final. Estos son métodos, funciones y variables globales. No se consideran símbolos:

- Las definiciones de tipos como `enum`, `struct` o `union`. Estos ayudan al programador a organizar el código y al compilador a validarlo, pero no generan código binario alguno. No existen desde el punto de vista del enlazador.
- Las variables locales. Estas se crean y se destruyen automáticamente en el *"Segmento StackSegmento Stack"* (Página 164) durante la ejecución del programa. No existen en tiempo de enlace.

Por otro lado, todas las funciones y variables globales declaradas como `static` dentro de un módulo `*.c` serán considerados **símbolos privados** no visibles en tiempo de enlace y donde el compilador es libre de realizar las optimizaciones oportunas. Con esto en mente, el código dentro de NAppGUI se organiza de la siguiente forma:

- ***.c:** Fichero de implementación. Definición de símbolos (funciones y variables globales).
- ***.h:** Fichero de cabecera pública. Declaración de funciones y variables globales (`extern`), disponibles para el usuario de la librería.
- ***.hxx:** Declaración de tipos públicos: `struct`, `union` y `enum`.
- ***.inl:** Declaración de funciones y variables privadas. Solo los módulos internos de la librería tendrán acceso a estos símbolos.
- ***.ixx:** Declaración de tipos privados. Aquellos compartidos entre los módulos de la librería, pero no con el exterior.

Si una función solo es necesaria dentro de un módulo `*.c`, no se incluye en un `*.inl`. Se marcará como `static` dentro del mismo `*.c`. De esta forma no estará visible para el enlazador y se permitirá al compilador realizar optimizaciones.

De igual forma, tipos que solo se utilicen dentro de un módulo concreto, se declararán al inicio del `*.c` y no en el `*.ixx`.

En pro de la mantenibilidad y escalibidad del código, se mantendrán las declaraciones de tipos y funciones lo más privado posible.

10.3.1. Exportación en DLLs

Cuando generamos una librería de enlace dinámico, además de incluir los símbolos públicos en una o varias cabeceras `*.h` deberemos marcarlos explícitamente como exportables. La macro de exportación se declara en el archivo `*.def` de cada librería. Por ejemplo en `core.def`, de define la macro `_core_api`.

Listado 10.7: core.def

```
/* Core library import/export */

#if defined(NAPPGUI_SHARED)
    #if defined(NAPPGUI_BUILD_CORE_LIB)
        #define NAPPGUI_CORE_EXPORT_DLL
    #else
        #define NAPPGUI_CORE_IMPORT_DLL
    #endif
#endif

#if defined(__GNUC__)
    #if defined(NAPPGUI_CORE_EXPORT_DLL)
        #define _core_api __attribute__((visibility("default")))
    #else
        #define _core_api
    #endif
#elif defined(_MSC_VER)
    #if defined(NAPPGUI_CORE_IMPORT_DLL)
        #define _core_api __declspec(dllimport)
    #elif defined(NAPPGUI_CORE_EXPORT_DLL)
        #define _core_api __declspec(dllexport)
    #else
        #define _core_api
    #endif
#else
    #error Unknown compiler
#endif
```

Esta macro deberá anteponerse a todas las funciones y variables declaradas en los `*.h`. Los proyectos basados en el `/src/CMakeLists.txt` definirán automáticamente las macros `CORE_IMPORT` y `NAPPGUI_SHARED_LIB` siempre que se vayan generar librerías dinámicas (exportar) o cuando se vayan a utilizar por un ejecutable (importar). En el caso de programas de terceros (no generados por `/src/CMakeLists.txt`) se deberán definir las macros de importación (`CORE_IMPORT`, `GUI_IMPORT`, etc) antes de incluir las cabeceras.

stream.h

```
/* Data streams */

#include "core.hxx"

__EXTERN_C

_core_api Stream *stm_from_block(const byte_t *data, const uint32_t size);

_core_api Stream *stm_memory(const uint32_t size);

_core_api Stream *stm_from_file(const char_t *pathname, ferror_t *error);
```

```
...

_core_api extern Stream *kSTDIN;

_core_api extern Stream *kSTDOUT;

_core_api extern Stream *kSTDERR;

__END_C
```

10.3.2. Comprobación en DLLs

Podemos ver, a partir del binario de una librería dinámica, que símbolos públicos exporta. En Windows utilizaremos `dumpbin /exports dllname`, en Linux `nm -D soname` y en macOS `nm -gU dylibname`.

Símbolos públicos de `core.dll` (Windows).

```
C:\>dumpbin /exports core.dll
2    1 00001000 array_all
3    2 00001010 array_bsearch
4    3 00001090 array_bsearch_ptr
5    4 00001120 array_clear
6    5 000011C0 array_clear_ptr
7    6 00001260 array_copy
8    7 00001340 array_copy_ptr
9    8 00001420 array_create
10    9 00001430 array_delete
11    A 00001530 array_delete_ptr
12    B 00001640 array_destopt
13    C 00001650 array_destopt_ptr
14    D 00001660 array_destroy
15    E 000016F0 array_destroy_ptr
16    F 00001790 array_esize
17   10 000017A0 array_find_ptr
18   11 000017D0 array_get
...
```

Símbolos públicos de `libcore.so` (Linux).

```
$ nm -D ./libcore.so
0000000000011f85 T array_all
000000000001305c T array_bsearch
000000000001316d T array_bsearch_ptr
0000000000011832 T array_clear
00000000000118a1 T array_clear_ptr
0000000000011009 T array_copy
000000000001115d T array_copy_ptr
0000000000010fdd T array_create
0000000000012649 T array_delete
```

```
000000000001276b T array_delete_ptr
0000000000011668 T array_destopt
0000000000011746 T array_destopt_ptr
00000000000115c3 T array_destroy
00000000000116ad T array_destroy_ptr
0000000000011b87 T array_esize
0000000000012dd3 T array_find_ptr
0000000000011e8c T array_get
```

Símbolos públicos de `libcore.dylib` (macOS).

```
% nm -gU ./libcore.dylib
00000000000029f0 T _array_all
0000000000003c90 T _array_bsearch
0000000000003d60 T _array_bsearch_ptr
00000000000024c0 T _array_clear
00000000000025d0 T _array_clear_ptr
0000000000001c20 T _array_copy
0000000000001dd0 T _array_copy_ptr
0000000000001b50 T _array_create
00000000000030f0 T _array_delete
0000000000003350 T _array_delete_ptr
00000000000022f0 T _array_destopt
0000000000002470 T _array_destopt_ptr
0000000000002120 T _array_destroy
0000000000002340 T _array_destroy_ptr
00000000000028b0 T _array_esize
0000000000003980 T _array_find_ptr
00000000000028f0 T _array_get
```

Capítulo 11

Recursos

Si internacionalizamos todo, terminamos con reglas que sofocan la libertad y la innovación.

Myron Scholes

Los recursos son datos necesarios para la aplicación pero que no residen en el área del ejecutable. En otras palabras, no son accesibles directamente mediante las variables del programa, sino que hay que realizar una carga previa para poderlos utilizar. Los más habituales son los textos e imágenes utilizadas en la interfaz de usuario, aunque cualquier tipo de archivo puede convertirse en un recurso (sonidos, tipografías, modelos 3d, páginas html, etc). Para ilustrar su uso con un ejemplo real, vamos a utilizar la aplicación `Die` (Figura 11.1), incluida en `/src/demo/die`.

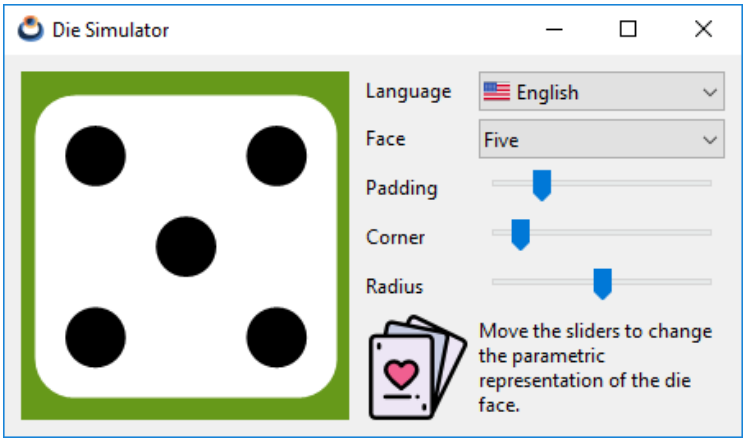


Figura 11.1: Aplicación Die.

11.1. Tipos de recursos

- **Textos**: Si bien es muy sencillo incluir textos en el código como variables de C, en la práctica esto no es aconsejable por dos motivos: El primero es que, normalmente, no son los programadores los que redactan los mensajes que muestra el programa. Separándolos en un archivo a parte, otros miembros del equipo pueden revisarlos y editarlos sin tener que acceder directamente al código. La segunda razón es la internacionalización. Es requisito casi indispensable a día de hoy poder cambiar el idioma del programa y esto puede involucrar a varios miembros del equipo, así como el hecho de que varias cadenas de texto hagan referencia al mismo mensaje. Por tanto, extraerlos del código fuente será casi indispensable.
- **Imágenes**: No es habitual que los iconos del programa cambien en función del idioma, aunque pueda darse el caso. Lo complicado aquí es transformar un archivo .jpg o .png en una variable de C (Listado 11.1). Hay que serializar el archivo y pegarlo en el código, algo muy tedioso y difícil de mantener para el programador. Es preferible tener las imágenes en una carpeta separada y acceder a ellas en tiempo de ejecución.

Listado 11.1: Imagen .png incrustada en el código fuente.

```
const uint32_t IMG_SIZE = 1262;

const byte_t IMG[] = {
            0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A,
            0x00, 0x00, 0x00, 0x0D, 0x49, 0x48, 0x44, 0x52,
            ... };
```

- **Archivos**: Al margen de texto e imágenes, cualquier archivo puede convertirse en un recurso. En este caso, la aplicación recibirá un bloque de bytes con el contenido del mismo, que deberá saber interpretar.

11.2. Crear recursos

Si vamos al directorio fuente de la aplicación (`/src/demo/die`), vemos que hay una carpeta llamada `/res` añadida por CMake al crear el proyecto. Dentro hay varios archivos `logo.*` con el *"Icono de aplicaciónIcono de aplicación"* (Página 143).

También puedes ver una carpeta llamada `/res/res_die` que **no fué creada por CMake**, sino añadida posteriormente al escribir el programa. Esta subcarpeta se considera un **paquete de recursos** y contendrá un conjunto de textos, imágenes o archivos que serán cargados "en bloque" en algún momento de la ejecución. Podemos crear tantos paquetes como sean necesarios en función del tamaño y lógica de nuestro programa.

> *En aplicaciones grandes, organiza tus recursos de tal forma que no sea necesario cargarlos todos al arrancar la aplicación. Es posible que ciertos recursos solo sean necesarios cuando el usuario realice alguna acción.*

Verás que dentro de `/res/res_die` existe un `strings.msg` cuyo contenido mostramos a continuación:

Listado 11.2: Fichero de mensajes de Die.

```
/* Die strings */
TEXT_FACE       Face
TEXT_PADDING    Padding
TEXT_CORNER     Corner
TEXT_RADIUS     Radius
TEXT_ONE        One
TEXT_TWO        Two
TEXT_THREE      Three
TEXT_FOUR       Four
TEXT_FIVE       Five
TEXT_SIX        Six
TEXT_TITLE      Die Simulator
TEXT_INFO       Move the sliders to change the parametric representation of the
    ↪  die face.
TEXT_LANG       Language
TEXT_ENGLISH    English
TEXT_SPANISH    Spanish
```

También contiene la imagen `cards.png` y los iconos `spain.png` y `usa.png` (Figura 11.2).

Cada línea dentro del archivo `strings.msg` define un nuevo mensaje que consta de un identificador (p.e. `TEXT_FACE`) seguido del texto que se mostrará en el programa (*Face* en este caso). Se considera texto desde el primer carácter no blanco después del identificador hasta el final de la línea. No es necesario ponerlo entre comillas (`"Face"`) como ocurre en C:

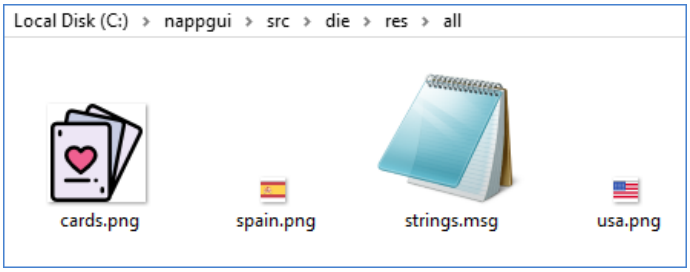


Figura 11.2: Paquete de recursos en `src/die/res/res_die`.

```
BILLY   Billy "the Kid" was an American Old West outlaw.
OTHER   Other text.
```

Tampoco hay que utilizar secuencias de escape ('\\', '\", ...), con la única excepción de `'\n'` para mensajes multilínea:

```
TWO_LINES   This is the first line\nAnd this is the second.
```

El identificador del mensaje sigue las reglas de los identificadores de C, exceptuando que las letras deben estar en mayúscula:

```
_ID1    Ok
0ID2    Wrong!!
id3     Wrong!!
ID3     Ok
```

Los mensajes aceptan cualquier carácter Unicode. Podemos dividir los textos en tantos archivos `*.msg` como sea necesario y deben ser almacenados en **formato UTF8**.

> *Visual Studio no guarda por defecto los archivos en UTF8. Asegurate de hacerlo en cada `*.msg` que contenga caracteres no US-ASCII. `File->Save As->Save with encoding-> Unicode (UTF8 Without Signature)- Codepage 65001`.*

11.3. Internacionalización (i18n)

Hemos utilizado el Inglés como idioma principal en el programa, pero queremos que también esté traducido al Español. Para ello volvemos a la carpeta `/res/res_die`, donde vemos el subdirectorio `/es_es` que contiene otro archivo `strings.msg`. Los identificadores en dicho archivo son los mismos que en `/res_die/strings.msg` pero los textos están en otro idioma. Dependiendo del lenguaje seleccionado, el programa utilizará una versión u otra.

Listado 11.3: Fichero de mensajes de Die, traducido al Español.

```
/* Die strings */
```

```
TEXT_FACE       Cara
TEXT_PADDING    Margen
TEXT_CORNER     Borde
TEXT_RADIUS     Radio
TEXT_ONE        Uno
TEXT_TWO        Dos
TEXT_THREE      Tres
TEXT_FOUR       Cuatro
TEXT_FIVE       Cinco
TEXT_SIX        Seis
TEXT_TITLE      Simulador de dado
TEXT_INFO       Mueve los sliders para cambiar la representación paramétrica de
    ↪  la cara del dado.
TEXT_LANG       Idioma
TEXT_ENGLISH    Inglés
TEXT_SPANISH    Español
```

Debemos tener en cuenta unas sencillas reglas a la hora de localizar recursos:

- Si no existe la versión local de un recurso, se utilizará la versión global del mismo. CMake avisará en el caso que existan **textos sin traducir** *"Avisos de nrcAvisos de nrc"* (Página 141).
- Se ignorarán aquellos recursos solo presentes en carpetas locales. Es imperativo que exista la versión global de cada uno.
- No se permiten "subpaquetes" de recursos. Solo se procesarán dos niveles: `src/res/packname` para los globales y `src/res/packname/local` para los localizados.
- Los paquetes de recursos deberán tener un nombre único dentro de la solución. Una estrategia puede ser anteponer el nombre del proyecto: `/appname_pack1`, `libname_pack2`, etc.
- Se ignorarán los recursos existentes en la carpeta raíz (`/res`). Todos los recursos deben estar incluidos en un paquete `/res/pack1/`, `/res/pack2/`, etc.
- Los textos localizados deben tener el mismo identificador que su correspondiente global. De lo contrario se consideran mensajes diferentes.
- Para crear la versión localizada de una imagen u otro archivo, incluirlo en su correspondiente carpeta local (p.e. `/res/res_die/es_es/cards.png`) utilizando **el mismo nombre de fichero** que la versión global.
- Para nombrar las carpetas localizadas, utilizar el código de idioma de dos letras ISO 639-1[1] (en, es, fr, de, zh, ...) y, opcionalmente, el código de país de dos letras ISO-3166[2] (en_us, en_gb, ...).

[1] `https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes`
[2] `https://en.wikipedia.org/wiki/ISO_3166-1`

11.4. Traducción en ejecución

Para cada paquete de recursos, CMake crea un `*.h` con el mismo nombre que la carpeta: `res_die.h` en este caso (Listado 11.4). Este archivo contiene los identificadores de recursos, así como una función que nos permite acceder a los mismos `res_die_respack()`. En (Listado 11.5) vemos las acciones a realizar para utilizar dichos recursos en nuestro programa.

Listado 11.4: Fichero de cabecera `res_die.h`.

```
/* Automatic generated by NAppGUI Resource Compiler (nrc-r1490) */

#include "core.hxx"

__EXTERN_C

/* Messages */
extern ResId TEXT_FACE;
extern ResId TEXT_PADDING;
extern ResId TEXT_CORNER;
extern ResId TEXT_RADIUS;
extern ResId TEXT_ONE;
extern ResId TEXT_TWO;
extern ResId TEXT_THREE;
extern ResId TEXT_FOUR;
extern ResId TEXT_FIVE;
extern ResId TEXT_SIX;
extern ResId TEXT_TITLE;
extern ResId TEXT_INFO;
extern ResId TEXT_LANG;
extern ResId TEXT_ENGLISH;
extern ResId TEXT_SPANISH;

/* Files */
extern ResId CARDS_PNG;
extern ResId SPAIN_PNG;
extern ResId USA_PNG;

ResPack *res_die_respack(const char_t *local);

__END_C
```

Listado 11.5: Carga y uso de recursos.

```
#include "res_die.h"

gui_respack(res_die_respack);
gui_language("");
...
label_text(label1, TEXT_FACE);
imageview_image(vimg, CARDS_PNG);
```

```
...
static void i_OnLang(App *app, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    const char_t *lang = params->index == 0 ? "en_us" : "es_es";
    gui_language(lang);
    unref(app);
}
```

- La línea 1 incluye la cabecera del paquete de recursos (Listado 11.4), que ha sido generada automáticamente por CMake.
- La línea 3 registra el paquete en *"Gui"* (Página 303), la librería encargada de la interfaz gráfica. Si la aplicación tuviese más paquetes de recursos los añadiríamos de la misma forma.
- La línea 4 establece el lenguaje por defecto (Inglés).
- Las líneas 6 y 7 asignan un texto y una imagen a dos controles respectivamente. Los identificadores están definidos en `"res_die.h"`, como acabamos de ver.
- La línea 13 traduce toda la interfaz como respuesta a un cambio en el control *"PopUp"* (Página 313) (Figura 11.3).

Figura 11.3: Traducción de la aplicación Die, sin destruir la ventana ni reiniciar.

Básicamente, una llamada a `gui_language`, implica coordinar tres acciones:

- Cargar los recursos localizados y sustituirlos por los actuales.
- Asignar los nuevos textos e imágenes a todos los controles y menús del programa.
- Volver a dimensionar las ventanas y los menús, ya que el cambio de textos e imágenes influirá en el tamaño de los controles.

11.5. Editar recursos

Para añadir nuevos archivos de recursos o eliminar alguno de los existentes, tan solo debemos ir a la carpeta `res/res_die` mediante el explorador de archivos y hacerlo allí directamente. Los archivos de mensajes `*.msg` se pueden editar desde Visual Studio, ya que CMake los incluye dentro del IDE (Figura 11.4). Tras realizar cualquier cambio en la carpeta de recursos o editar un archivo `*.msg`, debemos volver a lanzar CMake para que estas modificaciones se vuelvan a integrar en el proyecto. Tras cada actualización, se crearán los identificadores de los nuevos recursos y se eliminarán aquellos cuyo recurso asociado haya desaparecido, lo que producirá errores de compilación que facilitarán la corrección del código.

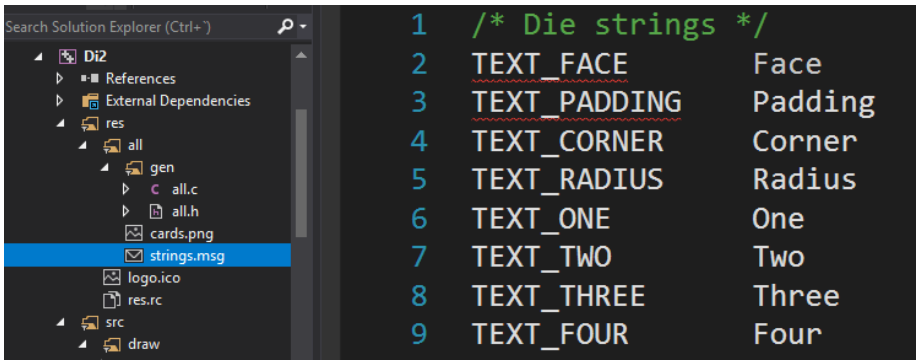


Figura 11.4: Edición de recursos dentro de Visual Studio.

11.6. Gestión manual

Aunque lo habitual será delegar la gestión de recursos en la librería `gui`, es posible acceder al contenido de los paquetes directamente, como vemos en (Listado 11.6).

Listado 11.6: Acceso directo a los recursos.

```
#include "res_die.h"

ResPack *pack = res_die_respack("es_es");
...
label_text(label1, respack_text(pack, TEXT_FACE));
imageview_image(vimg, respack_image(pack, CARDS_PNG));
...
respack_destroy(&pack);
```

- La línea 1 incluye la cabecera del paquete de recursos.
- La línea 3 crea un objeto con el contenido del paquete en idioma Español. Cada paquete de recursos proporcionará su propio constructor, cuyo nombre comenzará

por el de su carpeta xxxx_respack().

- Las líneas 5 y 6 obtienen un texto y una imagen respectivamente para asignarlos a los controles de interfaz.
- La línea 8 destruye el paquete de recursos, al finalizar su uso.

Hay una gran diferencia entre asignar recursos mediante `ResId` o mediante funciones `respack_` (Listado 11.7). En el primer caso, el control label será "sensible" a los cambios de idioma realizados por `gui_language`. Sin embargo, en los casos 2 y 3 se ha asignado un texto constante al control, que no se verá afectado por esta función. Seremos los responsables de cambiar el texto, llegado el caso.

Listado 11.7: Diferentes modos de acceso a recursos.

```
label_text(label1, TEXT_FACE);
label_text(label1, respack_text(pack, TEXT_FACE));
label_text(label1, "Face");
```

La elección de uno u otro modo de acceso dependerá de los requisitos del programa. Recordamos que para poder llevar a cabo las traducciones automáticas, los recursos deben registrarse con `gui_respack`.

11.7. Procesamiento de recursos

Vamos a ver con un poco más de detalle como NAppGUI genera los módulos de recursos. Al establecer `NRC_EMBEDDED` en el comando `desktopApp()`, le indicamos a CMake que debe procesar los recursos del proyecto Die. También podemos elegir la opción `NRC_PACKED` de la que hablaremos a continuación. Cuando lanzamos CMake se recorren las subcarpetas dentro del directorio `res` de cada proyecto, llamando a la utilidad **nrc** *(NAppGUI Resource Compiler)* (Figura 11.5). Este programa se encuentra en la carpeta `prj/scripts` de la distribución del SDK. Para cada paquete de recursos, *nrc* crea dos archivos fuente (un `.c` y un `.h`) y los vincula con el proyecto. El `.h` contiene los identificadores y el constructor que hemos visto en (Listado 11.4). Por su parte, el `.c` realiza la implementación del paquete en función del contenido de cada carpeta y de la modalidad `nrcMode`.

Los archivos creados por nrc se consideran código generado y no se guardan en la carpeta `src` sino en la carpeta `build`. Se actualizarán cada vez que se ejecute CMake, independientemente de la plataforma en la que estemos trabajando. Por el contrario, los archivos originales de recursos (ubicados en la carpeta `res`) sí se consideran parte del código fuente.

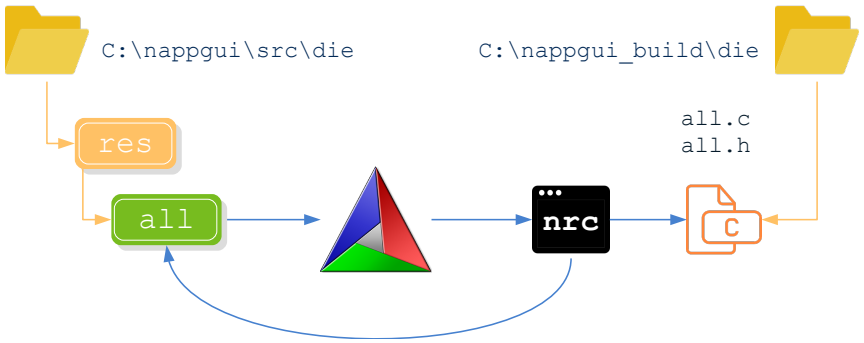


Figura 11.5: Procesamiento de recursos mediante CMake y nrc.

11.8. Distribución de recursos

En el capítulo anterior, al crear la solución de Visual Studio, indicamos que había que utilizar la constante `NRC_EMBEDDED` en la sentencia `desktopApp()` dentro del archivo `CMakeLists.txt`. Existen otras dos modalidades más relacionadas con la gestión de recursos y que pueden ser configuradas por separado dentro de cada comando `desktopApp()`:

- `NRC_NONE`: CMake ignorará el contenido de la carpeta `res`, a excepción del icono de la aplicación. No se generarán paquetes de recursos aunque exista contenido dentro de esta carpeta.
- `NRC_EMBEDDED`: Los recursos, con todas sus traducciones, se integran como parte del ejecutable (Figura 11.6). Es una opción muy interesante para aplicaciones de pequeño o medio tamaño, ya que en un único archivo `*.exe` suministraremos todo el programa. No hará falta un instalador y tendremos la certeza de que el software no fallará por la falta de algún fichero externo. El inconveniente es que, obviamente, el tamaño del ejecutable crecerá considerablemente por lo que no es aconsejable en programas con muchos recursos, muy pesados, o con multitud de traducciones.
- `NRC_PACKED`: Para cada paquete de recursos, se creará un archivo `*.res` externo al ejecutable que será cargado y liberado en tiempo de ejecución a medida que sea necesario (Figura 11.7). Las ventajas de este método son las desventajas del anterior y viceversa: Ejecutables más pequeños, pero con dependencias externas (los propios `.res`) que deberán ser distribuidos conjuntamente. También se optimizará el uso de la memoria, al poder cargar los `*.res` bajo demanda.

CMake gestiona por nosotros la ubicación de los paquetes de recursos. En aplicaciones Windows y Linux copiará todos los `*.res` en el directorio del ejecutable. En macOS los situará en la carpeta `resources` del bundle. Un hecho muy importante es que **no tenemos que modificar el código fuente** al pasar de una modalidad a otra. *nrc* ya se encarga de gestionar la carga según el tipo de paquete. Algo lógico puede ser comenzar por

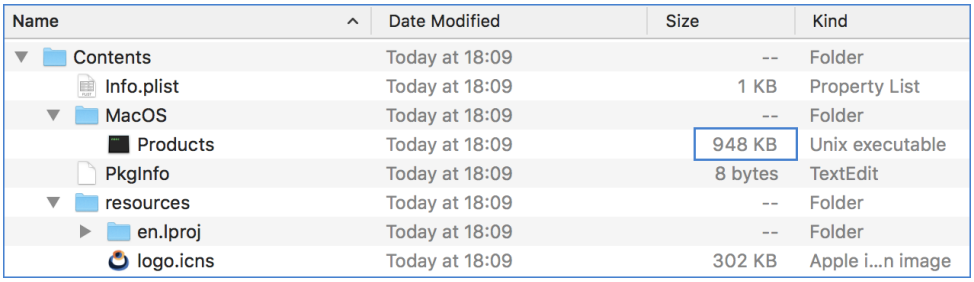


Figura 11.6: Distribución de una aplicación macOS con recursos embebidos.

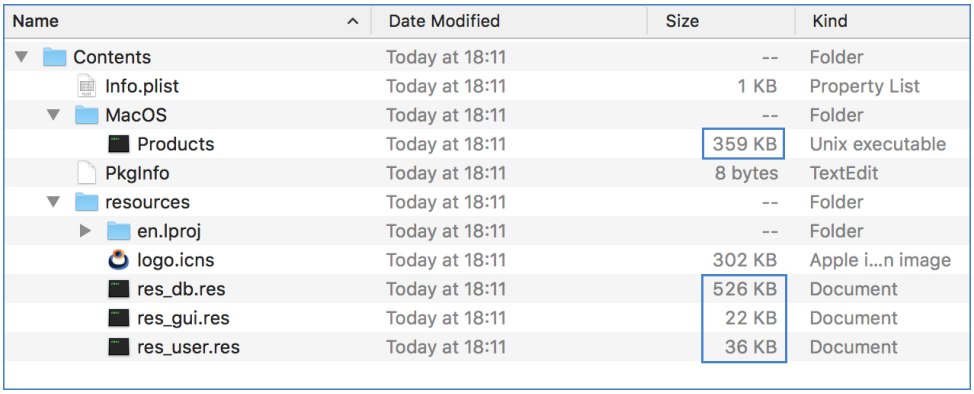


Figura 11.7: Distribución de la misma aplicación macOS con recursos empaquetados.

`NRC_EMBEDDED`, y si el proyecto crece, cambiar a `NRC_PACKED`. Tan solo deberemos volver a lanzar CMake y recompilar el proyecto para que el cambio se haga efectivo.

> *En Windows y Linux los archivos `*.res` deben instalarse siempre en el mismo directorio que el ejecutable. En el caso de macOS, CMake genera un bundle listo para distribución e instala los paquetes de recursos en el directorio `/resources` de dicho bundle.*

11.9. Avisos de nrc

nrc es un script silencioso cuyo trabajo se integra en el *build process* de CMake, pasando desapercibido en la mayoría de ocasiones. Pero hay veces que detecta anomalías en los directorios de recursos y debe informarnos de alguna manera. En estos casos aparecerá una línea roja en la consola de CMake indicando el proyecto y paquete(s) afectado(s) (Figura 11.8). Los detalles los vuelca en el archivo `NRCLog.txt` ubicado en la carpeta de recursos generados (CMake muestra la ruta completa).

Si los fallos son críticos, *nrc* no podrá generar los `*.h` y `*.c` asociados al paquete,

```
- HelloCpp: Starting
- HelloCpp: Completed
- Products: Starting
- nrc 'res_gui' warnings (See C:/NAPPGUI_1_0_build/demo/products/resgen/NRCLog.txt)
- Products: Completed
- BlockBreak: Starting
- BlockBreak: Completed
- Die: Starting
- Die: Completed
```

Figura 11.8: *nrc* ha encontrado anomalías al procesar recursos.

impidiendo que la aplicación se pueda compilar (en esencia no deja de ser un error de compilación). Otras veces son meros *warnings* que convendría arreglar, pero permiten seguir compilando. Concretamente, los **errores críticos** que afectan a *nrc* son los siguientes: (los mostramos en Inglés tal y como son escritos en `NRCLog.txt`).

- MsgError (%s:%d): Comment not closed (%s).
- MsgError (%s:%d): Invalid TEXT_ID (%s).
- MsgError (%s:%d): Unexpected end of file after string ID (%s).
- Duplicate resource id in '%s' (%s).
- Can't load resource file '%s'.
- Error reading '%s' resource directory.
- Error reading '%s' subdirectories.
- Error creating '%s' header file.
- Error creating '%s' source file.
- Error creating '%s' packed file.

Por otro lado, los avisos no-críticos:

- Empty message file '%s'.
- Ignored localized text '%s' in '%s'. Global resource doesn't exists.
- Ignored localized file '%s' in '%s'. Global resource doesn't exists.
- There is no localized version of the text '%s' in '%s'.
- Localized directory '%s' is empty or has invalid resources.

11.10. Icono de aplicación

Cuando creamos un nuevo proyecto, CMake establece un icono por defecto para la aplicación, que ubica en el directorio `/res`, con el nombre `logo*`. Esta imagen quedará "incrustada" en el ejecutable y el sistema operativo la utilizará para representar la aplicación en el escritorio (Figura 11.9). Windows y Linux también la utilizan en la barra de título de la ventana. Disponemos de tres versiones:

- **logo256.ico**: Versión para Windows Vista y posteriores. Deben incluir las resoluciones: 256x256, 48x48, 32x32 y 16x16.
- **logo48.ico**: Versión para Linux y VisualStudio 2008 y 2005, que no admiten resoluciones de 256x256. Esta versión solo incluye: 48x48, 32x32 y 16x16.
- **logo.icns**: Versión para macOS. Resoluciones 512x512, 256x256, 128x128, 32x32 y 16x16 tanto en resolución normal (@1x) como Retina Display (@2x).

Figura 11.9: Iconos de aplicación en la barra de tareas de Windows.

CMake ya se encarga de utilizar la versión apropiada del icono según la plataforma en la estemos compilando. Para cambiar el icono por defecto, abrir los archivos `logo*` con algún editor gráfico (Figura 11.10), realizar los cambios y volver a lanzar CMake. **Muy importante:** no cambiar los nombres de los archivo, siempre deben ser `logo256.ico`, `logo48.ico` y `logo.icns`.

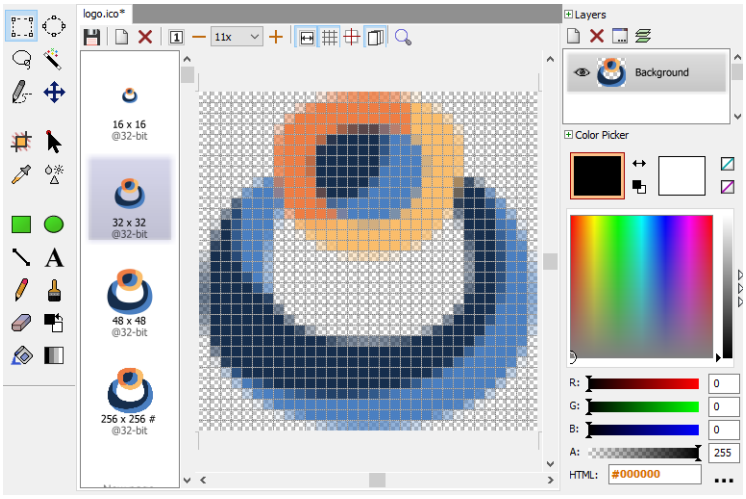


Figura 11.10: Editando `logo.ico`.

Parte 2

Introducción al API

Capítulo 12

NAppGUI SDK

Mientras que los civiles (es decir, los no programadores) a menudo fantasean con ganar la lotería, el equivalente para muchos programadores es la rara oportunidad de crear una nueva librería desde cero, sin las restricciones que a menudo frustran sus deseos de ampliar y mejorar una biblioteca existente.

Philip J. Schneider - Industrial Light + Magic

12.1. NAppGUI API

La implementación de NAppGUI se ha dividido en varias librerías escritas en ANSI-C (C90) con pequeñas partes en C++98 (Figura 12.1). El proyecto compila sin problemas en todas las versiones de Visual Studio (desde VS2005), Xcode (desde 3) y GCC (desde 4). Se puede utilizar para el desarrollo de aplicaciones de alto rendimiento escritas en C en los sistemas Windows, macOS y Linux. Se ha marcado una clara línea que separa los paquetes orientados a cálculo y acceso a datos *(back-end)* de aquellos destinados a las capas de presentación o interfaz *(front-end)*. También hemos seguido ciertos *"EstándaresEstándares"* (Página 58) cuyas bases se centralizan en la librería *"Sewer"* (Página 151), que si bien no incorpora mucha funcionalidad, si que define los tipos básicos y las macros de configuración comunes a todo el proyecto.

- Paquetes que no contienen código dependiente de plataforma.
- Paquetes que contienen código dependiente de plataforma bajo una interfaz común.

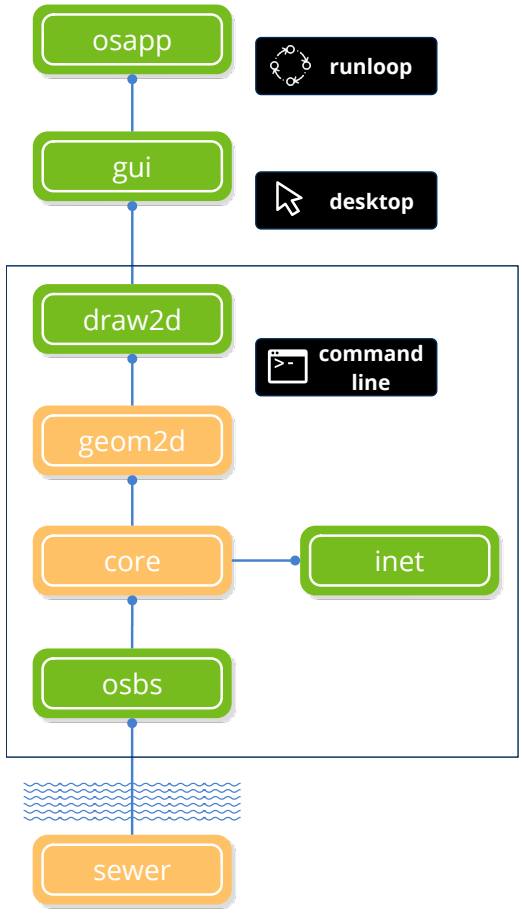


Figura 12.1: Arquitectura de NAppGUI.

- *"Sewer"* (Página 151): Tipos básicos, asserts, Unicode, librería estándar de C, funciones matemáticas.
- *"Osbs"* (Página 168): Servicios del sistema operativo. API portable sobre archivos, directorios, procesos, hebras, memoria, etc.
- *"Core"* (Página 191): Utilidades no gráficas de uso común. Auditor de memoria, estructuras de datos, canales E/S, análisis léxico, etc.
- *"Geom2D"* (Página 241): Geometría 2D. Transformaciones, vectores, polígonos, colisiones, etc.
- *"Draw2D"* (Página 262): API de dibujo vectorial, imágenes y fuentes tipográficas.
- *"Gui"* (Página 303): Compositor de interfaces de usuario de alto nivel.
- *"OSApp"* (Página 373): Aplicaciones de escritorio. Ciclo de mensajes.
- *"INet"* (Página 381): Protocolos y servicios de Internet, como HTTP.

12.2. Recursos online

Por motivos de espacio obvios, en este libro es imposible incluir una referencia completa de todas y cada una de los funciones que forman NAppGUI. En el sitio Web[1] del proyecto encontrarás una guía detallada función a función, así como el código fuente de varias aplicaciones de ejemplo.

Por lo tanto, consulta toda esta sección del libro de forma sosegada, con el único objetivo de tener una idea general de la estructura del software y de las diferentes partes que lo componen.

12.3. Un poco de historia

Empecé a trabajar en este proyecto de manera inconsciente, a mediados del año 2008 cuando estaba finalizando mis estudios de Ingeniería Informática en la Universidad de Alicante. Quería desarrollar un simulador de sistemas físicos que funcionase tanto en ordenadores PC-Windows como en los iMac de Apple sin que por ello tuviese que duplicar todo el trabajo. Las alternativas tecnológicas de la época, como GTK o Qt, no me convencían en absoluto ya que eran demasiado pesadas, complicadas de utilizar y lentas por lo que acabarían empañando la calidad, elegancia y empeño que estaba poniendo en mis algoritmos de cálculo matemático. Después de perder varios meses evaluando diferentes librerías para la programación multiplataforma descargué unos manuales técnicos de Apple para programar directamente en Cocoa, la tecnología base del fabricante de la manzanita para desarrollar software sobre iMac. A mediados de 2010 empecé a ver los primeros resultados y esto fue alentador. Había creado una aplicación con el prototipo de mi simulador en apenas 500Kb (Figura 12.2), en contraste con los más de 30Mb de dependencias exigidos por las soluciones de terceros. El código era compacto y limpio, la aplicación funcionaba a velocidad de vértigo y, sobre todo, tenía una apariencia profesional que recordaba en parte al iMovie, permitía manipular vistas 3D como en un videojuego y aportaba datos técnicos de la simulación en tiempo real. Esto me inspiró a seguir trabajando en trazar una barrera entre la parte de la aplicación reutilizable y aquella dependiente de una tecnología concreta. Esto permitiría adaptar mi simulador a diferentes modelos de ordenador y sistemas operativos.

Paralelamente, en Septiembre de 2008 me reincorporo al mercado laboral después de seis años en la Universidad, mercado en el que sigo en la actualidad (Mayo 2021), aunque los últimos años trabajo como autónomo desde casa lo que permite organizar la agenda y optimizar mi tiempo al máximo. En estos años no he abandonado mi proyecto personal, he seguido trabajando en él en a tiempo parcial sencillamente por pura afición. Su desarrollo me ha permitido investigar y profundizar en áreas interesantes para mí y reciclarme constantemente. En 2013 hago mi primera incursión en el mundo del emprendimiento siendo

[1] `https://www.nappgui.com`

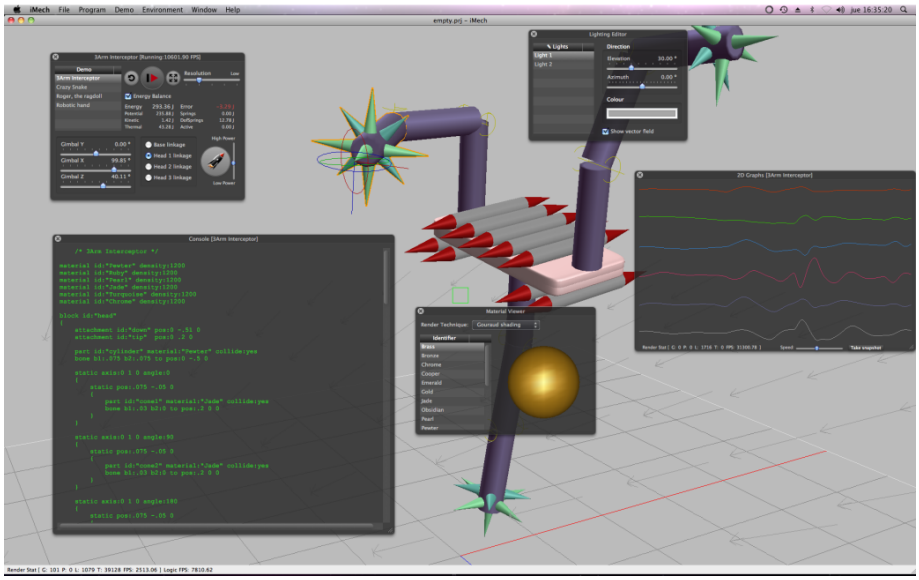


Figura 12.2: Simulador iMech, basado en una versión primitiva de NAppGUI.

co-fundador de iMech Technologies, empresa de software con la que sigo vinculado y cuyo principal objetivo fue la venta del motor de simulación que previamente había creado. Al no plantear una estrategia de marketing sólida, no conseguimos nuestros objetivos iniciales con iMech, pero pudimos reconvertirla incorporando nuevos clientes y, a día de hoy, sigue viva.

A mediados de 2015 empiezo a plantearme el hecho de que todo el esfuerzo técnico realizado durante estos años tiene suficiente entidad para convertirse en un producto por si mismo. Fué entonces cuando creo el proyecto NAppGUI y comienzo a migrar todas las librerías de iMech dedicadas al desarrollo multiplataforma. Durante estos últimos años he completado el soporte para *Cocoa* e incluido el de *Win32* y *Gtk+*. He creado esta documentación en Español y en Inglés, ayudándome de los servicios de traducción de Google.

El 31 de Diciembre de 2019, subo a GitHub la primera versión pública precompilada de NAppGUI.

En Mayo de 2020 comienzo el desarrollo de la primera aplicación comercial programada íntegramente con NAppGUI.

El 8 de Septiembre de 2021, libero el código fuente de NAppGUI en GitHub, convirtiéndolo en un proyecto Open Source bajo licencia MIT.

Capítulo 13

Librería Sewer

Incluso los más grandes palacios necesitaban alcantarillas.

Tom Lehrer

13.1. Sewer

Sewer es la primera librería dentro del SDK de NAppGUI (Figura 13.1). En ella se declaran los tipos básicos, el soporte para Unicode, aserciones, manipulación segura de

punteros, funciones matemáticas elementales, E/S estándar y reserva dinámica de memoria. También es utilizada como "sumidero" donde enterrar las antiestéticas macros del preprocesador necesarias para configurar el compilador, CPU, plataformas, etc. Depende tan solo de la librería estándar de C.

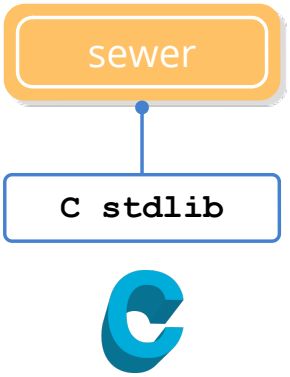


Figura 13.1: Dependencias de *sewer*. Ver *"NAppGUI API"* (Página 147).

13.1.1. La librería estándar de C

La librería estándar de C *(cstdlib)* no forma parte núcleo del lenguaje C, pero implementa funciones de gran utilidad para el desarrollador que resuelven problemas típicos de programación. Cualquier programador de C las ha utilizado en mayor o menor medida y su estudio va normalmente ligado con el aprendizaje del propio lenguaje (Figura 13.2).

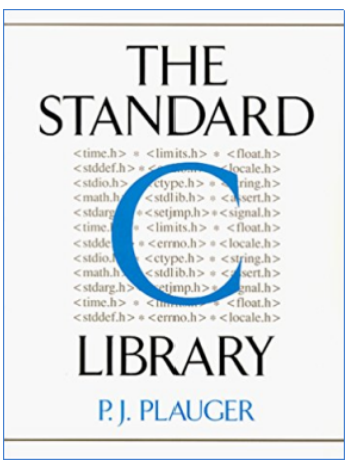


Figura 13.2: Una referencia completa a la librería de C la encontramos en el libro de P.J.Plauger.

Esta librería se sitúa a medio camino entre la aplicación y las llamadas al sistema y proporciona un API portable para acceso a archivos, memoria dinámica, E/S, tiempo, etc (Figura 13.3). También implementa funciones matemáticas, de conversión, búsqueda, manejo de cadenas, etc. De una forma o de otra, NAppGUI integra toda su funcionalidad, por lo que no es necesario (ni recomendable) utilizar *cstdlib* directamente en la capa de aplicación. Las razones que han motivado esta decisión de diseño las podemos resumir en:

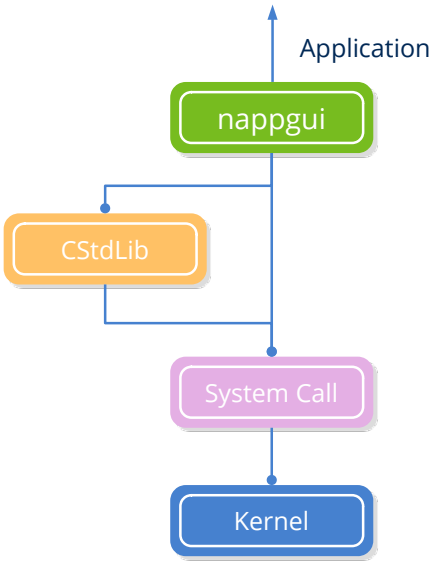


Figura 13.3: La funcionalidad de la librería de C se ha integrado en NAppGUI, evitando su uso directo en aplicaciones.

- **Pequeñas diferencias:** Los sistemas tipo-Unix no soportan las versiones seguras del *cstdlib* implementadas por Microsoft (`strcpy_s()` y otras). El uso de las funciones clásicas (sin el sufijo `_s`) es inseguro y activarán molestos *warnings* en Visual Studio.
- **Seguridad:** Relacionada con la anterior, evita vulnerabilidades del tipo *buffer overflow* en el tratamiento de cadenas y bloques de memoria.
- **Duplicidad:** Mucha de la funcionalidad de *cstdlib* ya se implementa en la librería *osbs* utilizando directamente llamadas al sistema (archivos, memoria dinámica, E/S, tiempo, etc).
- **Completo:** Las funciones de *cstdlib* relacionadas con archivos (`fopen()` y demás) no incorporan soporte para el manejo de directorios. *"Archivos y directorios"* (Página 179) presenta un API completo basado en llamadas al sistema.
- **Rendimiento:** En ciertos casos, sobre todo en funciones matemáticas y gestión de memoria, puede ser interesante cambiar la implementación de *cstdlib* por otra mejorada. Todas las aplicaciones se beneficiarán del cambio, sin tener que modificar su código.
- **Claridad:** El comportamiento de algunas funciones del *cstdlib* no es del todo claro y puede llevar a confusión. Por ejemplo, `strtoul` tiene un funcionamiento muy particular que debemos recordar cada vez que la utilicemos.

```
char *s1 = "-56";
char *s2 = "asCr";
char *s3 = "467Xd";
int v1, v2, v3;
v1 = strtoul(s1, NULL, 10); // v1 = 4294967240, errno = OK
```

```
v2 = strtoul(s2, NULL, 10); // v2 = 0, errno = OK
v3 = strtoul(s3, NULL, 10); // v3 = 467, errno = OK
```

- **Estilo:** El uso de funciones *sewer* no rompe la estética de una aplicación escrita con NAppGUI.

```
real32_t a1 = 1.43f;
real64_t a2 = .38;
real32_t c = (real32_t)cosf((float)a1);
real64_t t = (real64_t)tan((double)a2);
...
real32_t c = bmath_cosf(a1);
real64_t t = bmath_tand(a2);
```

- **Independencia:** NAppGUI utiliza internamente un subconjunto muy pequeño de funciones *cstdlib*. Es posible que en un futuro realicemos nuestras propias implementaciones y desvinculemos completamente el soporte de la librería estándar.
- **Enlace estático**: Si realizamos un enlace estático de la librería estándar, *sewer* contendrá todas las dependencias internamente. Esto evitará posibles incompatibilidades con los runtime instalados en cada máquina (los clásicos VC++ Redistributables de Windows). Con esto tendremos la certeza de que nuestros ejecutables van a funcionar, con independencia de la versión de runtime de C existente en cada caso. Si todas las llamadas a *cstdlib* están dentro de *sewer*, liberamos a las librerías de nivel superior de su gestión y de los posibles errores en tiempo de ejecución relacionados con el runtime de C.

Enlace estático de la *cstdlib* en Sewer. No necesita el runtime de C.

```
RUNTIME_C_LIBRARY "static"

dumpbin /dependents dsewer.dll

  Image has the following dependencies:

    KERNEL32.dll
```

Enlace dinámico de la *cstdlib* en Sewer. Necesita tener instalado un runtime concreto.

```
RUNTIME_C_LIBRARY "dynamic"

dumpbin /dependents dsewer.dll

  Image has the following dependencies:

    KERNEL32.dll
    VCRUNTIME140D.dll
    ucrtbased.dll
```

Para evitar posibles errores o incompatibilidades, no utilices funciones de la Librería Estándar de C directamente en las aplicaciones. Busca siempre una función de NAppGUI equivalente.

13.2. Asserts

Los **asserts** son sentencias distribuidas por el código fuente que realizan un *"Análisis dinámicoAnálisis dinámico"* (Página 61) intensivo, ayudando a detectar errores en tiempo de ejecución. Cuando la condición de un *assert* se vuelve `FALSE`, la ejecución del programa se detiene y se muestra una ventana de aviso (Figura 13.4).

- Utiliza `cassert` para introducir una comprobación dinámica en tu código.
- Utiliza `cassert_no_null` para vez que tengas que acceder al contenido de un puntero.

```
void layout_vmargin(Layout *layout, const uint32_t row, const real32_t
    ↪ margin)
{
    cassert_no_null(layout);
    cassert_msg(row < layout->num_rows, "'row' out of range");
    ...
}
```

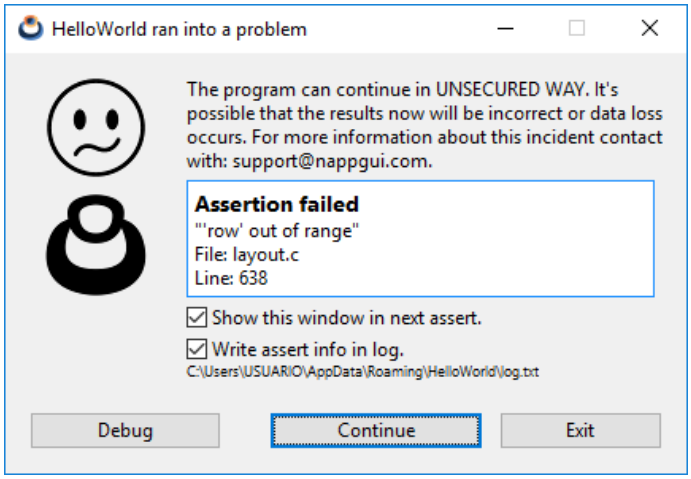


Figura 13.4: Ventana *assert* mostrada tras producirse un error en tiempo de ejecución.

En este momento tenemos tres alternativas:

- **Debug**: Depurar el programa: Acceso a la pila de llamadas, inspeccionar variables, etc. Más en *"Depurando el programaDepurando el programa"* (Página 63).
- **Continue**: Continuar con la ejecución, ignorando el *assert*.

- **Exit**: Salir del programa.

Para no volver a mostrar esta ventana en próximos *asserts*, desactivar el check `'Show this window in next assert'`. Futuras incidencias serán dirigidas a un archivo de *log*. También puedes omitir volcados en este log, desactivando `'Write assert info in log '`.

> *Los **asserts** proveen información muy importante sobre una anomalía del programa y nunca deben ser ignorados.*

En el ejemplo anterior hemos visto un *assert* "continuable", esto es, la ejecución del programa puede seguir si pulsamos `[Continue]`. No obstante, como ya indicamos, no deben ignorarse indefinidamente. Por otro lado tenemos los **asserts críticos** (Figura 13.5). Normalmente están relacionados con problemas de violación de segmento *("Segmentation Fault")*, donde no será posible continuar con la ejecución del programa.

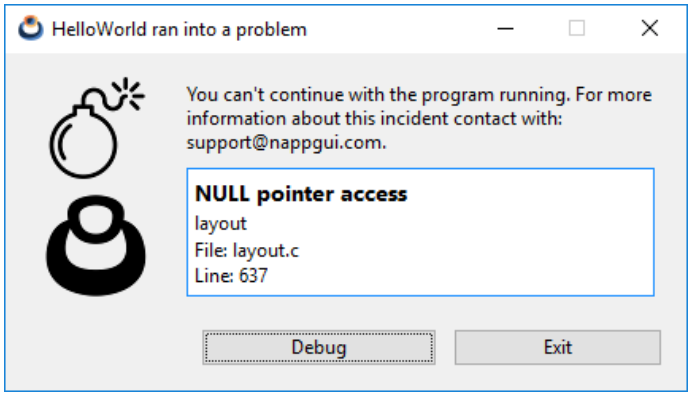


Figura 13.5: *Assert* crítico causado por el acceso a un puntero nulo.

13.3. Punteros

La librería *sewer* proporciona macros y funciones para la manipulación "segura" de punteros. Por "seguro" entendemos el hecho de que el SDK detectará el acceso indebido a un puntero justamente antes de que se produzca una *violación de segmento*. ¿Que sentido tiene detectar accesos indebidos a memoria, si el programa va a fallar de todos modos? La detección previa juega un papel muy importante al ejecutar tests automatizados. Antes del inevitable cierre del proceso, dejará un apunte en el registro de ejecución *log.txt*, indicando el motivo del cierre.

- Utiliza `ptr_get` para obtener el contenido de un puntero.

```
// v2 = NULL
// Segmentation fault
V2Df v1 = *v2;
```

```
// "v2 is NULL in file::line"
// will be record in log.txt
// and then, Segmentation fault
V2Df v1 = ptr_get(v2, V2Df);
```

13.4. Unicode

Unicode es un estándar de la industria informática, en esencia una tabla, que asigna un número único a cada símbolo de cada idioma en el mundo (Figura 13.6). Estos valores generalmente se denominan *codepoints* y se representan escribiendo **U+** seguido de su número en hexadecimal.

- Utiliza `unicode_convers` para convertir una cadena de una codificación a otra.
- Utiliza `unicode_to_u32` para obtener el primer codepoint de una cadena.

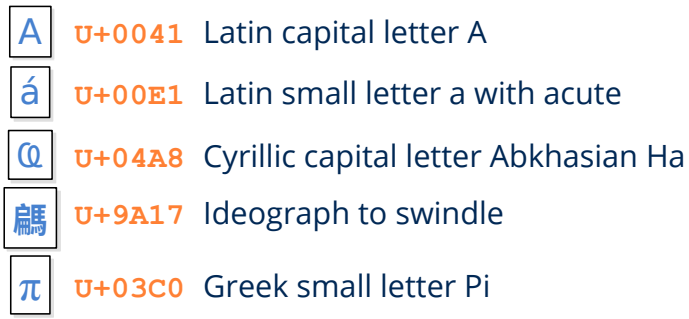


Figura 13.6: Varios *codepoints* Unicode.

Hablando de su estructura, tiene 17 planos de 65536 *codepoints* cada uno (256 bloques de 256 elementos) (Figura 13.7). Esto le da a Unicode un límite teórico de 1114112 caracteres, de los cuales 136755 ya se han ocupado (versión 10.0 de junio de 2017). Para aplicaciones del mundo real, el más importante es el Plano 0 denominado *Plano Multilingüe Básico* (BMP), que incluye los símbolos de todas las idiomas modernos del mundo. Los planos superiores contienen carácteres históricos y símbolos adicionales poco convencionales.

Las primeros computadores utilizaban ASCII *American Standard Code for Information Interchange*, un código de 7 bits que define todos los caracteres del idioma Inglés: 26 letras minúsculas (sin signos diacríticos), 26 letras mayúsculas, 10 dígitos, 32 símbolos de puntuación, 33 códigos de control y un espacio en blanco, para un total de 128 posiciones. Tomando el bit adicional dentro de un byte, tendremos espacio para otros 128 símbolos, pero aún insuficiente para todos. Esto da como resultado numerosas páginas de códigos ASCII extendidos, lo que es un gran problema para compartir textos, ya que el mismo código numérico puede representar diferentes símbolos según la página ASCII utilizada (Figura 13.8).

Ya a principios de los 90, con la llegada de Internet, este problema se agravó, ya que el

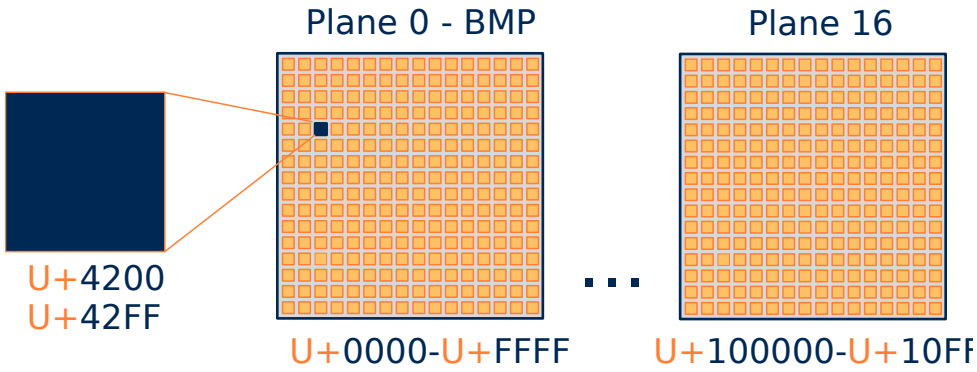


Figura 13.7: Unicode tiene 17 planos de 256x256 *codepoints* cada uno.

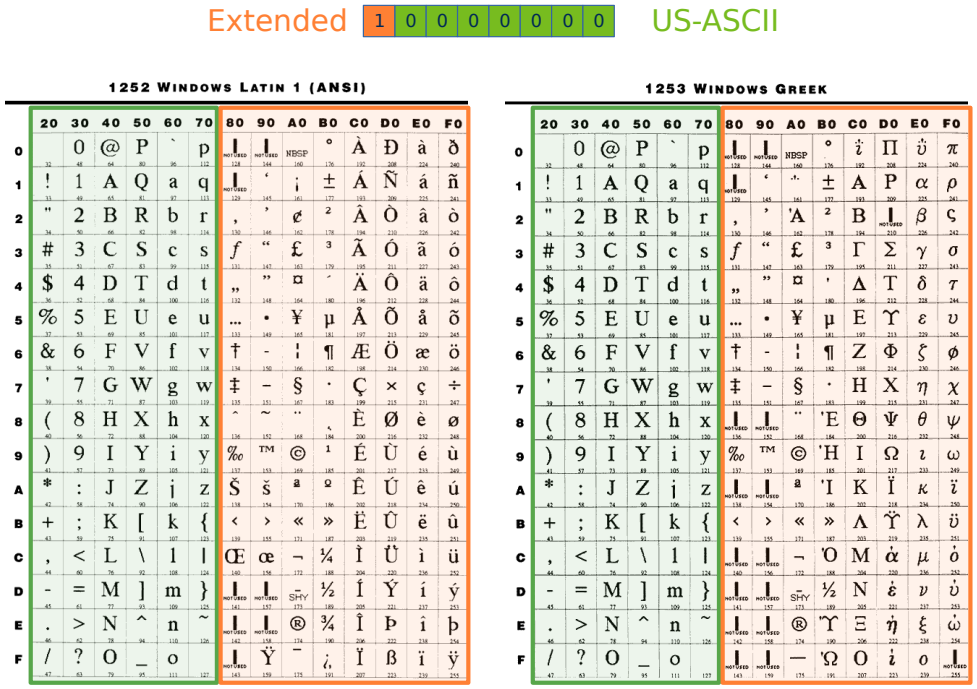


Figura 13.8: En cada página del ASCII Extendido, los 128 códigos superiores representan diferentes caracteres.

intercambio de información entre máquinas de diferente naturaleza y país se convirtió en algo cotidiano. El Consorcio de Unicode (Figura 13.9) se contituyó en California en enero de 1991 y, en octubre del mismo año, se publicó el primer volumen del estándar Unicode.

Figura 13.9: Miembros de pleno derecho del Consorcio Unicode.

13.4.1. Codificaciones UTF

Cada *codepoint* necesita 21 bits para ser representado (5 para el plano y 16 para el desplazamiento). Esto casa muy mal con los tipos básicos en ordenadores (8, 16 o 32 bits). Por este motivo, se han definido tres codificaciones *Unicode Translation Format - UTF* función del tipo de dato que se utilice en la representación (Figura 13.10).

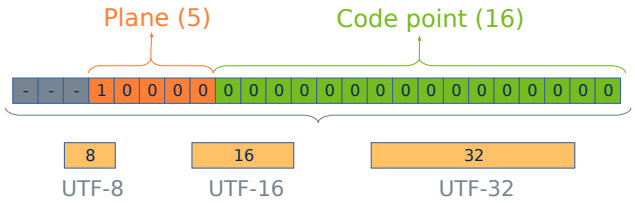


Figura 13.10: Codificaciones para almacenar *codepoints* de 21-bit mediante elementos de 8, 16, o 32.

13.4.2. UTF-32

Sin ningún problema, utilizando 32 bits podemos almacenar cualquier *codepoint*. También podemos acceder aleatoriamente mediante un índice a los elementos de un array, de la misma forma que las cadenas ASCII clásicas de C (char). Las malas noticias son los requisitos de memoria. Una cadena UTF32 necesita cuatro veces más espacio que una ASCII.

```
const char32_t code1[] = U"Hello";
const char32_t code2[] = U"áéíóú";
uint32_t s1 = sizeof(code1); /* s1 == 24 */
uint32_t s2 = sizeof(code2); /* s2 == 24 */
for (i = 0; i < 5; ++i)
{    /* Accessing by index */
     if (code1[i] == 'H')
         return i;
}
```

13.4.3. UTF-16

UTF16 reduce a la mitad el espacio requerido por UTF32. Es posible almacenar un *codepoint* por elemento siempre que no abandonemos el plano 0 (BMP). Para planos superiores, serán necesarios dos elementos UTF16 (32bits). Este mecanismo, que encapsula los planos superiores dentro del BMP, se conoce como **pares subrogados**.

```
const char16_t code1[] = u"Hello";
const char16_t code2[] = u"áéíóú";
uint32_t s1 = sizeof(code1); /* s1 == 12 */
uint32_t s2 = sizeof(code2); /* s2 == 12 */
for (i = 0; i < 5; ++i)
{    /* DANGER! Only BMP */
     if (code1[i] == 'H')
         return i;
}
```

Para iterar sobre una cadena UTF16 que contenga caracteres de cualquier plano debe utilizarse `unicode_next`.

13.4.4. UTF-8

UTF8 es un código de longitud variable donde cada *codepoint* utiliza 1, 2, 3 o 4 bytes.

- 1 byte **(0-7F)**: los 128 símbolos del ASCII original. Esto supone una gran ventaja, ya que las cadenas US-ASCII son cadenas UTF8 válidas, sin necesidad de conversión.
- 2 bytes **(80-7FF)**: Caracteres diacríticos y de lenguaje romance, griego, cirílico, coptos, armenio, hebreo, árabe, siríaco y thaana, entre otros. Un total de 1920 *codepoints*.
- 3 bytes **(800-FFFF)**: Resto del plano 0 (BMP).
- 4 bytes **(10000-10FFFF)**: Planos superiores (1-16).

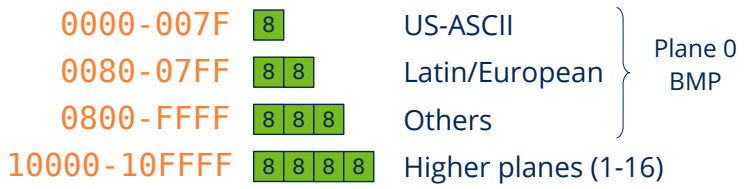


Figura 13.11: Cada carácter en UTF8 emplea 1, 2, 3 o 4 bytes.

Más del 90 % de los sitios web utilizan UTF8 (agosto de 2018[1]), porque es el más óptimo en términos de memoria y velocidad de transmisión en red. Como desventaja, tiene asociado un pequeño coste computacional para codificar/decodificar, ya que es necesario realizar operaciones de nivel de bit para obtener los *codepoints*. Tampoco es posible acceder aleatoriamente mediante índice a un carácter concreto, tenemos que procesar la cadena completa.

```
const char_t code1[] = "Hello";
const char_t code2[] = "áéíóú";
```

[1] https://w3techs.com/technologies/overview/character_encoding/all

```
const char_t *iter = code1;
uint32_t s1 = sizeof(code1); /* s1 == 6 */
uint32_t s2 = sizeof(code2); /* s2 == 11 */
for (i = 0; i < 5; ++i)
{
    if (unicode_to_u32(iter, ekUTF8) == 'H')
        return i;
    iter = unicode_next(iter, ekUTF8);
}
```

13.4.5. Uso de UTF-8

UTF8 es la codificación requerida por todas las funciones del SDK NAppGUI. Las razones por las que hemos elegido UTF-8 sobre otras codificaciones han sido:

- Es la evolución natural del US-ASCII.
- Las aplicaciones serán directamente compatibles con la gran mayoría de servicios de Internet (JSON/XML).
- En entornos multi-lenguaje los textos ocuparán menos espacio. Estadísticamente, los 128 caracteres ASCII son los más utilizados en promedio y solo necesitan un byte en UTF8.
- Como desventaja, en aplicaciones dirigidas exclusivamente al mercado asiático (China, Japón, Corea - CJK), UTF8 es menos eficiente que UTF16.

Dentro de aplicaciones NAppGUI pueden cohexistir diferentes representaciones (`char16_t`, `char32_t`, `wchar_t`). No obstante, **recomendamos encarecidamente el uso de UTF8** en favor de la portabilidad y para evitar las constantes conversiones dentro del API. Para convertir cualquier cadena a UTF8 se utiliza la función `unicode_convers`.

```
wchar_t text[] = L"My label text.";
char_t ctext[128];
unicode_convers((const char_t*)text, ctext, ekUTF16, ekUTF8, 128);
```

NAppGUI no ofrece soporte para convertir páginas del ASCII Extendido a Unicode.

El objeto `Stream` proporciona conversiones automáticas de UTF al leer o escribir en canales de E/S mediante los métodos `stm_set_write_utf` y `stm_set_read_utf`. También es posible trabajar con el tipo `String` (cadenas dinámicas), que incorpora multitud de funciones optimizadas para el tratamiento UTF8. Podemos incluir cadenas de texto constantes directamente en el código fuente (Figura 13.12), aunque lo habitual será escribirlas en archivos de recursos (*"Recursos"* (Página 131)). Evidentemente, deberemos guardar tanto los archivos de código fuente como los de recursos en UTF8. Todos los entornos de desarrollo actuales soportan la opción:

- De forma predeterminada, Visual Studio guarda los archivos fuente en formato ASCII (Windows 1252). Para cambiar a UTF8, ir a `File->Save As->Save with encoding-> Unicode (UTF8 Without Signature)- Codepage 65001`. No hay forma de establecer esta configuración para todo el proyecto :-(.
- En Xcode es posible establecer una configuración global. `Preferences->Text editing ->Default Text Encoding->Unicode (UTF-8)`.
- En Eclipse también permite una configuración global. `Window->Preferences-> General->Workspace->Text file encoding`.

```
static const char_t text[] = {
    "Hello World!",
    "「こんにちは世界」",
    "你好，世界！",
    "Привет мир!",
    "Γειά σου Κόσμε!" };

/* API works with UTF8 */
label_text(label, text[2]);
button_text(button, text[3]);
```

Figura 13.12: Constantes UTF8 en un archivo fuente en C.

13.5. Matemáticas

`BMath` ofrece una interfaz compacta sobre las funciones matemáticas elementales de la librería estándar de C. También define algunas de las constantes más utilizadas, como el número Pi, conversiones entre grados y radianes o raíz de 2.

- Utiliza `bmath_cosf` para calcular el coseno de un ángulo (*wrapper* sobre `cosf()` de la cstdlib).
- Utiliza `bmath_sqrtf` para calcular la raíz cuadrada (*wrapper* sobre `sqrtf()` de la cstdlib).

13.5.1. Números aleatorios

`BMath` incluye un generador de números pseudo-aleatorios basados en semilla. A partir de una misma semilla, la secuencia de números generados será siempre la misma. Las secuencias producidas por dos semillas diferentes serán radicalmente dispares. De ahí que se denominen pseudo-aleatorios.

- Utiliza `bmath_rand_seed` para establecer la semilla de números aleatorios.

- Utiliza `bmath_randf` para obtener un número aleatorio en coma flotante, dentro de un intervalo.

En el caso que de aplicaciones multi-hilo la esta secuencia puede variar en función del orden de ejecución de las hebras, ya que estas funciones **no son re-entrantes**. Deberás utilizar un "entorno" de números aleatorios para cada hilo en cuestión, en el caso que necesites asegurar siempre la misma secuencia (algoritmos deterministas).

- Utiliza `bmath_rand_env` para crear un entorno seguro de números aleatorios.
- Utiliza `bmath_rand_mtf` para obtener un número aleatorio a partir de un entorno.

13.6. Funciones estándar

`BLib` incluye funciones útiles de la librería estándar de C que no encajan en otros módulos como `BMath` o `BMem`. Al igual que en `<stdlib.h>` encontramos funciones de conversión de texto, algoritmos o interacción con el entorno.

- Utiliza `blib_strcmp` para comparar dos cadenas de texto.
- Utiliza `blib_qsort` para ordenar un vector de elementos.
- Utiliza `blib_bsearch` para realizar una búsqueda dicotómica sobre un vector ordenado.
- Utiliza `blib_abort` para acabar la ejecución del programa.

13.7. E/S Estándar

Todos los procesos disponen de canales de entrada y salida por defecto, sin necesidad de crearlos explícitamente. Por canales entendemos *streams* o flujos de datos.

- Utiliza `bstd_printf` para escribir texto en las salida estándar.
- Utiliza `bstd_read` para leer bytes desde la entrada estándar.

Cada proceso en ejecución tiene tres canales de comunicación estándar:

- **stdin:** entrada de datos. El proceso leerá datos que le llegan desde el exterior.
- **stdout:** salida de datos. El proceso escribirá resultados en este canal.
- **stderr:** salida de errores. El proceso escribirá en este canal información referente a errores.

Es como tener tres archivos perpetuamente abiertos donde el programa puede leer y escribir sin límites. Cuando ejecutamos un proceso desde la Consola o el Terminal, `stdin` se conecta automáticamente al teclado y `stdout`/`stderr` a la pantalla (Figura 13.13). No

obstante, estos canales estándar se pueden redirigir para usar archivos como fuentes de entrada o destinos de salida:

```
dir > out.txt
ls > out.txt
sort < out.txt
```

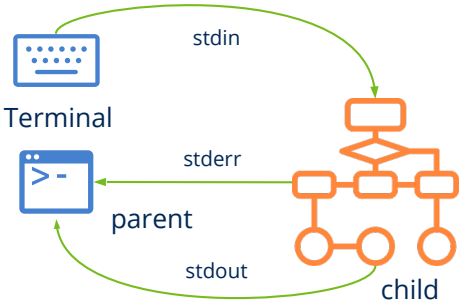


Figura 13.13: Ejecutando un proceso desde el Terminal.

En este fragmento de código, el resultado del comando `dir` (`ls` en Unix) se ha redirigido al archivo `out.txt`, por lo que no veremos nada por pantalla. Por otro lado, el comando `sort` no espera la entrada del usuario por el teclado. Simplemente la toma del archivo `out.txt`, ordenando sus líneas. Por tanto, siempre que escribamos aplicaciones en línea de comandos, deberemos utilizar convenientemente estos canales estándar sin hacer presunciones desde donde viene o hacia donde va la información tratada por la aplicación.

13.8. Memoria

Desde la perspectiva del programador, el acceso a memoria se realiza por medio de variables y se manipula a través de los operadores del lenguaje (+, -, *, =, ...) y siempre de la misma forma, independientemente de como se hayan creado las variables o en que zona de memoria se alojen. Dentro de `bmem.h` disponemos de varias funciones para realizar copias, asignaciones o comprobaciones de bloques genéricos de memoria. En este módulo también se definen funciones para la manipulación de la memoria dinámica *(Heap)*.

- Utiliza `bmem_malloc` para reservar un bloque de memoria dinámica.
- Utiliza `bmem_free` para liberar un bloque de memoria dinámica.
- Utiliza `bmem_copy` para copiar el contenido de dos bloques de memoria, previamente reservados.

13.8.1. Segmento Stack

La memoria de un programa C compilado y en ejecución se divide en varios segmentos. Uno de ellos es el *stack*, un espacio de tamaño variable pero limitado, donde se guardan

las variables locales y las llamadas a función *(call stack)*. Va creciendo y encogiendo a medida que el proceso entra y sale de ámbitos o funciones (Figura 13.14). Es gestionado automáticamente por el compilador como una estructura LIFO *Last-in First-out*, por lo que pasa desapercibido la mayor parte del tiempo, ya que no requiere atención extra por parte del programador. Nos percatamos de su existencia al recibir el error *Stack Overflow*, normalmente provocado por recursividad infinita o la reserva de vectores C muy grandes (Listado 13.1). El depurador nos permite inspeccionar el estado de la pila en cualquier instante de la ejecución *"Depurando el programaDepurando el programa"* (Página 63).

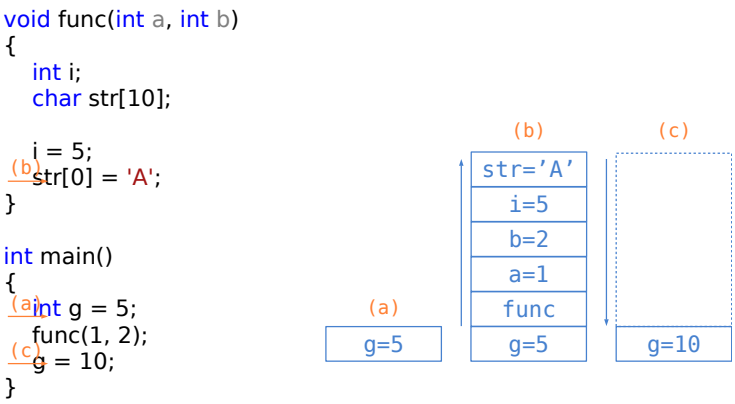


Figura 13.14: Estado del *stack* en diferentes puntos del programa.

Listado 13.1: Dos sencillos casos que provocan el desbordamiento de la pila.

```
int func(int n) { func(n); } // Stack Overflow

float v[2000000]; // Stack Overflow
```

Si bien el uso del *stack* es ideal debido a su gran rendimiento, seguridad y facilidad de uso, en ocasiones se queda corto. Por un lado hay que prever en tiempo de diseño la cantidad de memoria necesaria y definirla estáticamente (pe. `struct Product pr[100];`), algo muy poco flexible a la hora de construir aplicaciones de verdad. Por otro lado, las variables se destruyen al cerrar un ámbito o salir de una función, lo que impide compartir datos de manera global.

13.8.2. Segmento Heap

El *heap* es una zona de memoria que el proceso puede solicitar bajo demanda, a través de llamadas al sistema. Es complementaria al *stack* y se caracteriza por:

- Se puede acceder de forma global, desde cualquier punto del programa a través de un puntero.

- La cantidad de memoria disponible es prácticamente ilimitada.
- Es menos eficiente que el *stack*.
- Requiere gestión. Los sistemas operativos proveen funciones para solicitar bloques de memoria dinámica (`HeapAlloc()`, `sbrk()`), siendo responsabilidad del proceso, o mejor dicho del programador, liberar estos bloques cuando ya no se necesiten.

Como las reservas y liberaciones pueden realizarse en cualquier orden se produce una fragmentación interna a medida que avanza el programa (Figura 13.15). Aquí entraría en juego el denominado **gestor de memoria**, que son algoritmos que permiten optimizar el uso del heap compactándolo y reutilizando los bloques liberados. La librería estándar de C proporciona las conocidas funciones `malloc()/free()`, que implementan un gestor de memoria genérico a través de llamadas al sistema.

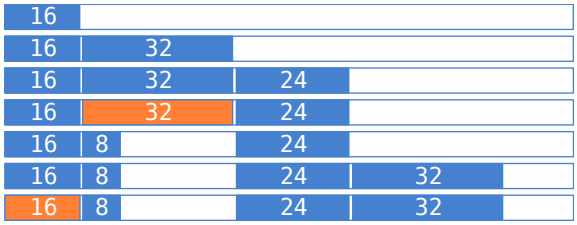


Figura 13.15: Fragmentación del *heap* durante la ejecución del proceso.

NAppGUI implementa su propio gestor/auditor de memoria dinámica "Heap - Gestor de memoria" (Página 192) muy optimizado para servir numerosas de peticiones de pequeño tamaño, que es lo que demandan las aplicaciones en gran medida. `bmem_malloc` */* `bmem_free` *conectan con el sistema operativo a través de llamadas al sistema y no deberían utilizarse directamente.*

Capítulo 14

Librería Osbs

No hay una distinción clara entre el software del sistema operativo y el software que se ejecuta sobre él.

Jim Allchin

14.1. Osbs

osbs *(Operating System Basic Services)* es un *wrapper* portable que permite a las aplicaciones la comunicación con el núcleo del sistema operativo a nivel de procesos, memoria, archivos y redes. Esta comunicación se lleva a cabo a través de una serie de **llamadas al sistema** (Figura 14.1) que varían según el sistema operativo para el que estemos programando. Es el API no gráfico de más bajo nivel para comunicarse con dispositivos de hardware y acceder a los recursos de la máquina. Por debajo se encuentran los controladores de dispositivos *(drivers)* administrados directamente por el *kernel*, a los que las aplicaciones tienen vetado el acceso.

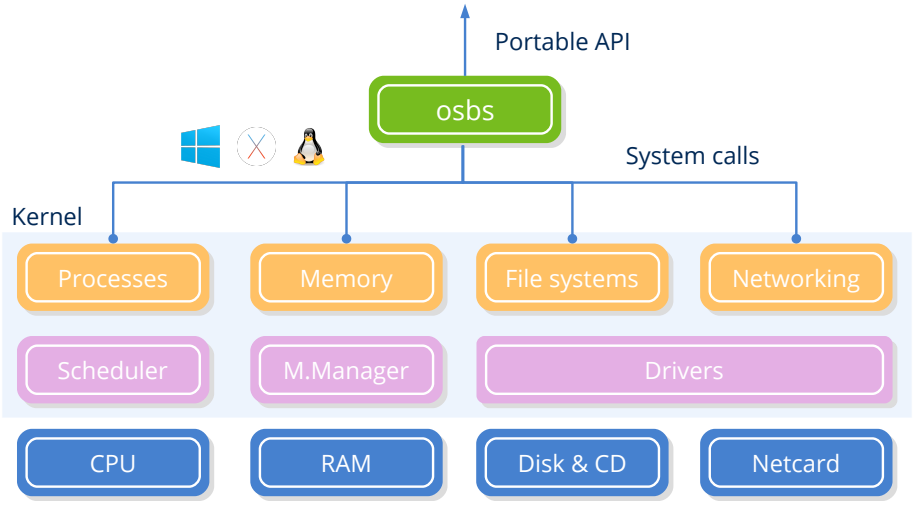


Figura 14.1: Las llamadas al sistema son la puerta de entrada al núcleo del sistema operativo.

Darwin, el *kernel* de macOS, y Linux son sistemas tipo-Unix, por tanto, comparten el mismo conjunto de llamadas al sistema (con pequeñas diferencias). Pero Windows presenta una arquitectura y juego de funciones radicalmente diferente. La librería **osbs** de NAppGUI no es más que una pequeña envoltura que internamente maneja estas diferencias y provee una vía común para acceder a los mismos recursos en diferentes plataformas (Figura 14.2). Solo depende de *"Sewer"* (Página 151) y sus funcionalidades se han dividido en diferentes módulos:

- *"Procesos"* (Página 169), *"Hebras"* (Página 172), *"Exclusión mutua"* (Página 177).

- *"Carga de librerías"* (Página 178).
- *"Archivos y directorios"* (Página 179).
- *"Sockets"* (Página 181).
- *"Tiempo"* (Página 185).

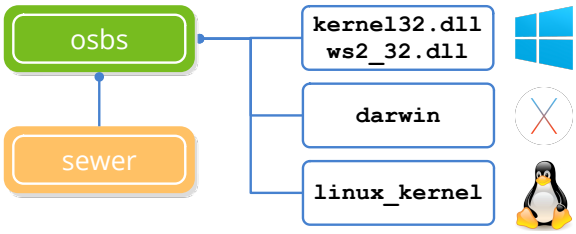


Figura 14.2: Dependencias de *osbs*. Ver *"NAppGUI API"* (Página 147).

14.2. Procesos

Desde la perspectiva del programador, el multi-procesamiento es la posibilidad de iniciar e interactuar con otros procesos (hijos) desde proceso principal (padre). El sistema operativo puede ejecutar el proceso hijo en otro núcleo de la CPU *(true multitasking)* o en el mismo que el padre *(context switch)*. Esta es una decisión del sistema en la que el programador no puede influir que dependerá del tipo de procesador y de su carga de trabajo. El efecto final será que ambos procesos (padre e hijo) se ejecutan en paralelo.

- Utiliza `bproc_exec` para lanzar un nuevo proceso desde la propia aplicación.
- Utiliza `bproc_read` para leer desde la salida estándar del proceso.
- Utiliza `bproc_write` para escribir en la entrada estándar del proceso.

14.2.1. Lanzando procesos

`bproc_exec` lanzará un proceso desde nuestro propio programa en C de forma similar a como lo hace el Terminal (Figura 14.3). En este caso, la *"E/S Estándar"* (Página 163) `stdin`, `stdout` y `stderr` se redireccionará al objeto `Proc` mediante tuberías anónimas. Desde aquí, podemos utilizar `bproc_write` para escribir en el canal `stdin` del hijo y `bproc_read` para leer desde su `stdout`. Las reglas de lectura/escritura son las que rigen los *pipes* del sistema operativo y que podemos resumir en:

- Si el padre llama a `bproc_read` y el hijo no ha escrito nada (búfer vacío), el padre se bloqueará (esperará) hasta que haya información en el canal de salida del hijo.
- Si el hijo termina y el padre está esperando para leer, `bproc_read` devolverá `FALSE` y el padre continuará su ejecución.

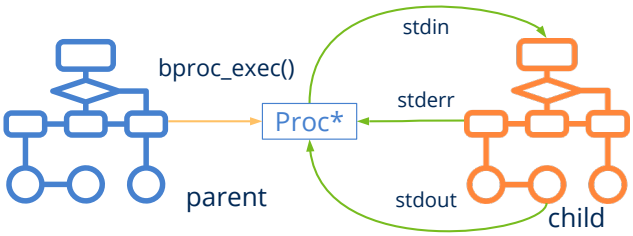


Figura 14.3: Lanzando un proceso desde nuestro propio código C.

- Si el padre llama a `bproc_write` y el búfer de escritura está lleno, el padre se bloqueará (esperará) hasta que el hijo lea desde su `stdin` y libere espacio en el canal.
- Si el hijo termina y el padre está bloqueado por escritura, `bproc_write` devolverá `FALSE` y el padre continuará su ejecución.
- Algunos comandos o procesos (p.e `sort`) no se iniciarán hasta que se lea todo el contenido de `stdin`. En estos casos, el proceso padre deben usar `bproc_write_close` para indicar al hijo que la escritura en su `stdin` ha concluido.
- Cuando el padre llama a `bproc_close`, todos los canales de E/S se cerrarán y ambos procesos continuarán su ejecución de forma independiente. Para terminar la ejecución del proceso hijo *(kill)* utiliza `bproc_cancel`.
- `bproc_wait` detendrá al proceso padre hasta que el hijo finalice. Para evitar la sobrecarga del buffer de salida `stdout` del hijo, cerrar el canal mediante `bproc_read_close` .
- `bproc_finish` comprobará, de forma no bloqueante, si el hijo ha terminado su ejecución.

14.2.2. Ejemplos multi-procesamiento

Veamos algunos ejemplos prácticos de IPC *Inter-Process Communication* utilizando los canales E/S estándar en procesos vinculados padre-hijo. En (Listado 14.1) volcaremos en un fichero la salida `stdout` del proceso hijo. En (Listado 14.2) redirigiremos ambos canales, escribiremos en `stdin` y leeremos de `stdout` apoyándonos en archivos de disco. Por último, implementaremos un protocolo asíncrono donde el padre y el hijo intercambian peticiones y respuestas. En (Listado 14.4) mostramos el código del proceso hijo, en (Listado 14.3) el del proceso padre y en (Listado 14.5) el resultado de la comunicación, escrito por el proceso padre.

Listado 14.1: Leyendo del `stdout` de un proceso y guardándolo en un archivo.

```
byte_t buffer[512];
uint32_t rsize;
File *file = bfile_create("out.txt", NULL);
```

```
Proc *proc = bproc_exec("dir C:\Windows\System32", NULL);
while(bproc_read(proc, buffer, 512, &rsize, NULL) == TRUE)
    bfile_write(file, buffer, rsize, NULL, NULL);
bproc_close(&proc);
bfile_close(&file);
```

Los comandos del shell no son portables en general. Los utilizamos solo como ejemplo.

Listado 14.2: Redirigiendo el `stdin` y `stdout` de un proceso.

```
byte_t buffer[512];
uint32_t rsize;
File *fsrc = bfile_open("members.txt", ekFILE_READ, NULL);
File *fdes = bfile_create("sorted_members.txt", NULL);
Proc *proc = bproc_exec("sort", NULL);

// Writes to stdin
while (bfile_read(fsrc, buffer, 512, &rsize, NULL) == TRUE)
    bproc_write(proc, buffer, rsize, NULL, NULL);

// Closes child stdin
bproc_write_close(proc);

// Reads child stdout
while(bproc_read(proc, buffer, 512, &rsize, NULL) == TRUE)
    bfile_write(fdes, buffer, rsize, NULL, NULL);

bfile_close(&fsrc);
bfile_close(&fdes);
bproc_close(&proc);
```

Listado 14.3: Protocolo asíncrono (proceso padre).

```
Proc *proc;
uint32_t commands[] = { 326, 32, 778, 123, 889, 712, 1, 55, 75, 12 };
uint32_t exit_command = 0;
uint32_t i;

proc = bproc_exec("child", NULL);

for (i = 0; i < 10; ++i)
{
    uint32_t response;
    uint32_t time;
    // Send command to child
    bproc_write(proc, (byte_t*)&commands[i], sizeof(uint32_t), NULL);

    // Waits for child response
    bproc_read(proc, (byte_t*)&response, sizeof(uint32_t), NULL);
    bproc_read(proc, (byte_t*)&time, sizeof(uint32_t), NULL);
```

```
    bstd_printf("Child command %d in %d milliseconds.\n", response, time);
}

bproc_write(proc, (byte_t*)&exit_command, sizeof(uint32_t), NULL);
bproc_close(&proc);
```

Listado 14.4: Protocolo asíncrono (proceso hijo).

```
for (;;)
{
    uint32_t command;
    // Reads from standard input a command from parent.
    if (bstd_read((byte_t*)&command, sizeof(command), NULL) == TRUE)
    {
        if (command != 0)
        {
            // Waits random time (simulates processing).
            uint32_t timer = bmath_randi(1000, 2000);
            bthread_sleep(timer);

            // Writes to standard output the response to parent.
            bstd_write((const byte_t*)&command, sizeof(command), NULL);
            bstd_write((const byte_t*)&timer, sizeof(timer), NULL);
        }
        else
        {
            // Command 0 = Exit
            break;
        }
    }
}
```

Listado 14.5: Resultado de la ejecución del proceso padre.

```
Child command 326 in 1761 milliseconds.
Child command 32 in 1806 milliseconds.
Child command 778 in 1989 milliseconds.
Child command 123 in 1909 milliseconds.
Child command 889 in 1043 milliseconds.
Child command 712 in 1153 milliseconds.
Child command 1 in 1780 milliseconds.
Child command 55 in 1325 milliseconds.
Child command 75 in 1157 milliseconds.
Child command 12 in 1426 milliseconds.
```

14.3. Hebras

Los **hilos** o **hebras** son diferentes caminos de ejecución dentro del mismo proceso (Figura 14.4). También son conocidos como **procesos ligeros**, ya que son más ágiles de

crear y gestionar que los procesos propiamente dichos. Comparten código y espacio de memoria con el programa principal, por lo que es muy fácil intercambiar información entre ellos a través de variables de memoria. Una hebra comienza su ejecución en un método conocido como *thread_main* y, en el momento que se lanza, se ejecuta en paralelo con el hilo principal. Al igual que los procesos, son objetos controlados por el núcleo del sistema que dictaminará, en última instancia, si la hebras se ejecutarán en otro CPU-core *(true multitasking)* o lo compartirán *(context switch)*.

- Utiliza `bthread_create` para crear un nuevo hilo de ejecución.
- Utiliza `bthread_wait` para obligar al hilo principal a que espere que se ejecute el hilo.

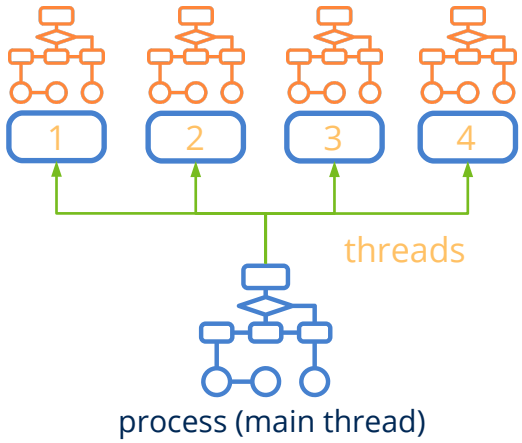


Figura 14.4: Un proceso con múltiples hilos de ejecución.

14.3.1. Lanzando hebras

Cada llamada a `bthread_create` creará un nuevo hilo en paralelo comenzando en la función pasada como parámetro *(thread_main)*. La forma "natural" de finalizarlo es retornando de *thread_main*, aunque es posible abortarlo desde el hilo principal.

Código básico para lanzar un hilo de ejecución paralelo.

```
static uint32_t i_thread(ThData *data)
{
    // Do something
    ...
    // Thread execution ends
    return 0;
}

Thread *thread = bthread_create(i_thread, data, ThData);
// Main thread will continue here
// Second thread will run 'i_thread'
```

14.3.2. Variables compartidas

Cada nuevo hilo tiene su propio *"Segmento StackSegmento Stack"* (Página 164) por tanto, todas las variables automáticas, llamadas a función y reservas dinámicas serán privadas a dicho hilo. Pero también puede recibir datos globales del proceso a través del parámetro `data` de *thread_main*. Debemos tener cuidado al acceder a datos globales a través de múltiples hilos concurrentes, ya que modificaciones realizadas por otros hilos pueden alterar la ejecución lógica del código produciendo errores muy difíciles de depurar. El programa (Listado 14.6) es correcto para programas de un solo hilo, pero si la variable `vector` es accedida por dos hebras simultáneas, puede derivar en un error *Segmentatin Fault* si la hebra-1 libera la memoria mientras la hebra-2 está ejecutando el bucle.

Listado 14.6: Acceso peligroso a variables compartidas.

```
if (shared->vector != NULL)
{
    shared->total = 0;
    for(i = 0; i < shared->n; i++)
        shared->total += shared->vector[i];
    bmem_free(shared->vector);
    shared->vector = NULL;
}
```

Para evitar este problema, deberemos proteger los accesos a variables compartidas a través de un `Mutex` (Listado 14.7). Este mecanismo de *"Exclusión mutua"* (Página 177) garantiza que solo un hilo puede acceder al recurso en un instante de tiempo. Un hilo será detenido si pretende ejecutar el código situado entre `bmutex_lock` y `bmutex_unlock` si otro hilo se encuentra dentro de esta *sección crítica*.

Listado 14.7: Acceso seguro a variables compartidas.

```
bmutex_lock(shared->mutex);
if (shared->vector != NULL)
{
    shared->total = 0;
    for(i = 0; i < shared->n; i++)
        shared->total += shared->vector[i];
    bmem_free(shared->vector);
    shared->vector = NULL;
}
bmutex_unlock(shared->mutex);
```

14.3.3. Ejemplo multi-hilo

Lo complicado de la programación multi-hilo es descomponer una solución en partes que puedan correr en paralelo y organizar las estructuras de datos para que esto se pueda

llevar a cabo de la forma más equilibrada posible. En (Listado 14.8) el programa correrá cuatro veces más rápido (x4) ya que se ha hecho una división perfecta del problema (Figura 14.5). Esto no es más que un ejemplo teórico y este resultado será muy difícil de conseguir en situaciones reales. También deberemos reducir al mínimo la cantidad de variables compartidas y el tiempo de las secciones críticas, de lo contrario los posibles inter-bloqueos reducirán la ganancia.

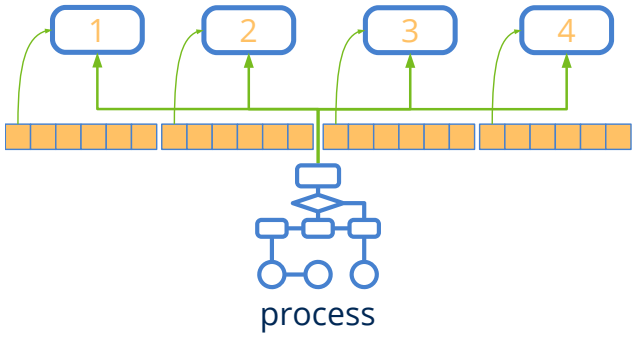


Figura 14.5: Colaboración de cuatro hebras en el cálculo de un vector.

Listado 14.8: Procesamiento multi-hilo de un vector muy grande.

```
typedef struct _app_t App;
typedef struct _thdata_t ThData;

struct _app_t
{
    uint32_t total;
    uint32_t n;
    uint32_t *elems;
    Mutex *mutex;
};

struct _thdata_t
{
    uint32_t thread_id;
    uint32_t start;
    uint32_t end;
    uint64_t time;
    App *app;
};

static uint32_t i_thead(ThData *data)
{
    uint32_t i, total = 0;
    uint64_t t1 = btime_now();
    for (i = data->start; i < data->end; ++i)
    {
        // Simulates processing
        uint32_t time = bmath_randi(0, 100);
        bthread_sleep(time);
```

```
        total += data->app->elems[i];
    }

    // Mutual exclusion access to shared variable 'total'
    bmutex_lock(data->app->mutex);
    data->app->total += total;
    bmutex_unlock(data->app->mutex);
    data->time = (btime_now() - t1) / 1000;
    return data->thread_id;
}

// Threads creating function
uint32_t i, m;
uint64_t t;
App app;
ThData thdata[4];
Thread *thread[4];

// App data vector
i_init_data(&app);
app.mutex = bmutex_create();
m = app.n / 4;

// Thread data
for (i = 0; i < 4; ++i)
{
    thdata[i].thread_id = i;
    thdata[i].app = &app;
    thdata[i].start = i * m;
    thdata[i].end = (i + 1) * m;
}

// Launching threads
t = btime_now();
for (i = 0; i < 4; ++i)
    thread[i] = bthread_create(i_thead, &thdata[i], ThData);

// Wait for threads end
for (i = 0; i < 4; ++i)
{
    uint32_t thid = bthread_wait(thread[i]);
    bstd_printf("Thread %d finished in %d ms.\n", thid, thdata[thid].time);
    bthread_close(&thread[i]);
}

// Process total time
t = (btime_now() - t) / 1000;
bstd_printf("Proccessing result = %d in %d ms.\n", app.total, t);

bmutex_close(&app.mutex);
```

Listado 14.9: Resultado.

```
Thread 0 finished in 13339 ms.
Thread 1 finished in 12506 ms.
Thread 2 finished in 12521 ms.
Thread 3 finished in 12999 ms.
Proccessing result = 499500 in 13344 ms.
```

14.4. Exclusión mutua

En procesos con múltiples hilos, la exclusión mutua garantiza que solo uno de ellos pueda ejecutar una **sección crítica** en un instante concreto de tiempo. La sección crítica es un bloque de código que normalmente protege a un recurso compartido que no soporta el acceso concurrente.

- Utiliza `bmutex_create` para crear un cerrojo.
- Utiliza `bmutex_lock` para bloquear una sección crítica.
- Utiliza `bmutex_unlock` para desbloquear una sección crítica.

14.4.1. Cerrojos

Los cerrojos, candados o `Mutex` son objetos de sincronización gestionados por el sistema operativo que marcan el inicio y final de una sección crítica (Figura 14.6). Cuando una hebra va a acceder a un determinado recurso compartido, deberá llamar al método `bmutex_lock` para garantizar el acceso exclusivo. Si otro hilo está utilizando el recurso (ha llamado previamente a `bmutex_lock`), el hilo actual se detendrá hasta que el recurso se libere mediante `bmutex_unlock`. Del bloqueo y desbloqueo de hilos se encarga el propio sistema operativo. El programador, tan solo debe preocuparse por identificar y proteger las secciones críticas. *"Ejemplo multi-hiloEjemplo multi-hilo"* (Página 174).

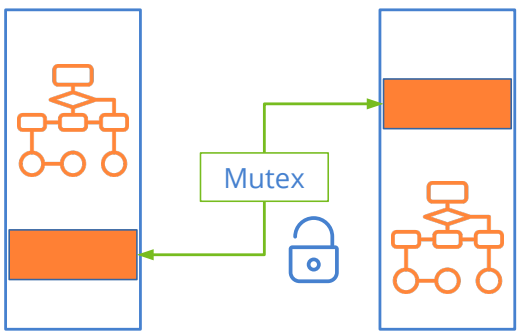


Figura 14.6: Un mutex protegiendo las secciones críticas de dos hebras, que no podrán ejecutarse concurrentemente. El resto del código puede correr en paralelo.

14.5. Carga de librerías

La habitual, en proyectos de relativo tamaño, es dividir el código del programa en librerías con el fin de poder reutilizarlas en diferentes proyectos. El enlace de estas librerías dentro del ejecutable final puede realizarse de tres formas:

- **Tiempo de compilación:** El código de la librería se copia dentro del ejecutable, formando parte de inseparable del mismo (librerías estáticas) (Figura 14.7) (a).
- **Tiempo de carga:** El código de la librería se distribuye por separado (librerías dinámicas) y se carga junto con el programa principal, al mismo tiempo (Figura 14.7) (b).
- **Tiempo de ejecución:** Librerías dinámicas que el programa carga en el momento que las necesita (Figura 14.7) (c).

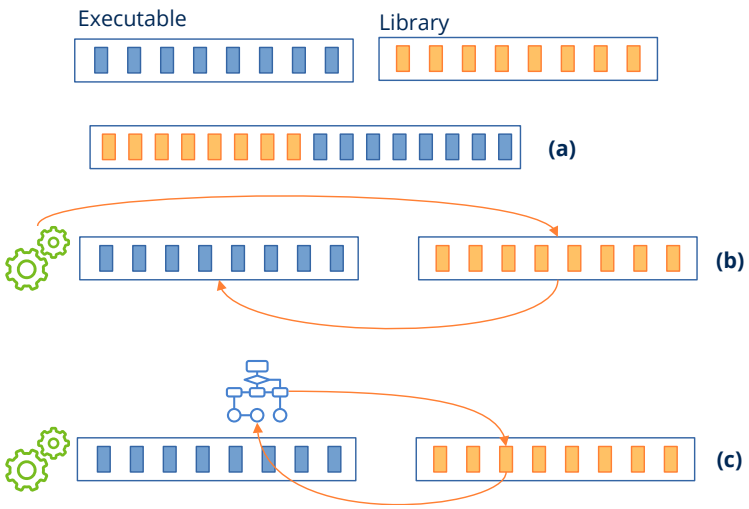


Figura 14.7: Enlace de librerías y carga dinámica.

El proceso de enlace es relativamente complicado y se gestiona automáticamente mediante el compilador y el cargador del sistema operativo. El programador solo debe intervenir en el tercer caso, ya que es necesario incluir código para cargar las librerías y acceder a los métodos o variables oportunos en cada momento.

- Utiliza `dlib_open` para cargar una librería en tiempo de ejecución.
- Utiliza `dlib_proc` para obtener un puntero a una función de la librería.
- Utiliza `dlib_var` para obtener un puntero a una variable de la librería.

14.5.1. Rutas de búsqueda de librerías

Una librería dinámica se encuentra en un archivo diferente al de los ejecutables que pueden hacer uso de ella. Cada sistema operativo implementa diferentes estrategias de búsqueda que debemos conocer para instalar y/o configurar los programas de la forma correcta.

14.5.2. Orden de búsqueda en Windows

- El directorio `path` de `dlib_open`.
- El mismo directorio que el ejecutable.
- El directorio actual `bfile_dir_work`.
- El directorio `%SystemRoot%\System32`.
- El directorio `%SystemRoot%`.
- Los directorios especificados en la variable de entorno `PATH`.

14.5.3. Orden de búsqueda en Linux/macOS

- Los directorios especificados en la variable de entorno `LD_LIBRARY_PATH` (Linux) o `DYLD_LIBRARY_PATH` (macOS).
- Los directorios especificados en el ejecutable `rpath`.
- Los directorios del sistema `/lib, /usr/lib, etc`.

14.6. Archivos y directorios

14.6.1. Sistema de archivos

El sistema de archivos *(filesystem)* es la estructura jerárquica compuesta por directorios y ficheros que permite organizar los datos persistentes de la computadora (Figura 14.8). Es algo con lo que los usuarios de ordenadores estamos muy familiarizados, sobre todo tras la irrupción de los sistemas gráficos que introdujeron la analogía de escritorio, carpeta y documento. Comienza en un directorio denominado raíz (`/` en Unix ó `C:\` en Windows) y, a partir de aquí, cuelgan todos los sub-directorios y ficheros formando un árbol que crece en profundidad. A nivel de programación, el sistema de archivos se gestiona a través de llamadas al sistema que permiten crear directorios, navegar por su contenido, abrir ficheros, eliminarlos, obtener atributos, etc.

- Utiliza `bfile_create` para crear un nuevo archivo.
- Utiliza `bfile_dir_create` para crear un directorio.

- Utiliza `bfile_dir_open` para abrir un directorio con el fin de explorar su contenido.
- Utiliza `bfile_dir_get` para obtener información sobre una entrada del directorio.

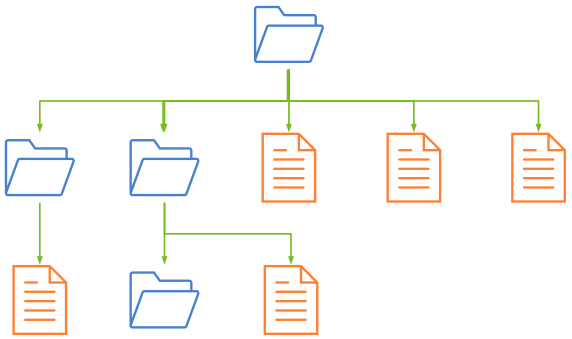


Figura 14.8: Estructura típica de un sistema de archivos.

14.6.2. Archivos y flujos de datos

Un proceso puede leer o escribir datos en un fichero después de abrir un canal de E/S (*"Streams"* (Página 197)) el cual provee un flujo de datos binarios desde o hacia el propio proceso (Figura 14.9). Existe un puntero que se va desplazando de forma secuencial cada vez que se leen o escriben datos. Inicialmente se encuentra en el byte 0, pero podemos modificarlo para acceder a posiciones aleatorias del archivo sin necesidad de leer el contenido (Figura 14.10). Esto puede resultar muy útil a la hora de trabajar con ficheros grandes cuyos datos están indexados de alguna manera.

- Utiliza `bfile_open` para abrir un archivo existente.
- Utiliza `bfile_read` para leer datos desde un archivo.
- Utiliza `bfile_write` para escribir datos a un archivo.
- Utiliza `bfile_seek` para modificar el puntero de archivo.

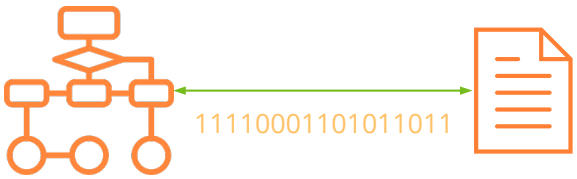


Figura 14.9: Tras abrir un fichero, el proceso dispone de un canal E/S para leer o escribir datos.

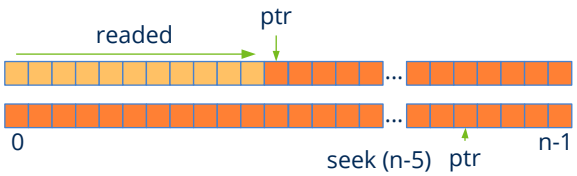


Figura 14.10: Lectura secuencial o acceso aleatorio.

14.6.3. Filename y pathname

Estos dos conceptos son recurrentes y ampliamente utilizados por las funciones del API que manipulan archivos. Cuando navegamos por el contenido de un directorio `bfile_dir_get`, obtenemos una secuencia de *filenames* que es el nombre "plano" del elemento (archivo o subdirectorio) sin incluir su ruta dentro del sistema del archivos (sin caracteres `'/'` o `'\'`). Por otro lado el *pathname* es una secuencia de uno o varios *filenames* separados por `'/','\'`, que indica el camino a seguir para localizar un determinado elemento. Este camino puede ser **absoluto** cuando comienza por el directorio raíz (`C:\Users\john\docs\images\party.png`) o **relativo** (`docs\images\party.png`) cuando indica la ruta parcial a partir del *working directory* del proceso.

- Utiliza `bfile_dir_work` para obtener el directorio de trabajo actual.
- Utiliza `bfile_dir_set_work` para establecer el directorio de trabajo.

14.6.4. Home y AppData

Estos son dos directorios típicos utilizados por las aplicaciones para almacenar los archivos relativos a un determinado usuario. Por un lado, *home* indica el directorio personal del usuario actualmente registrado en el sistema, típicamente `c:\Users\john` (Windows), `/home/john` (Linux) ó `/Users/john` (macOS). Por otro lado *appdata* es un directorio reservado para guardar datos temporales o de configuración de las aplicaciones. Localizaciones típicas pueden ser `C:\Users\john\AppData\Roaming` (Windows), `/home/john/.config` (Linux) ó `/User/john/Library` (macOS). Lo habitual será crear una sub-carpeta con el nombre de la aplicación `/User/john/Library/TheApp`.

- Utiliza `bfile_dir_home` para obtener el directorio *home* del usuario.
- Utiliza `bfile_dir_data` para obtener el directorio de datos de aplicaciones.
- Utiliza `bfile_dir_exec` para obtener el directorio del ejecutable.

14.7. Sockets

Podemos definir un **socket** como un canal de comunicación entre dos procesos que están ejecutándose en diferentes máquinas. Utilizan como base la familia de protocolos TCP/IP que rigen la comunicación por Internet desde los primeros prototipos de la gran red allá por el año 1969. Por su parte, el protocolo IP *(Internet Protocol)* se encarga del envío de pequeños paquetes de datos entre dos computadores remotos a través de la red. Como hay paquetes que pueden perderse o tomar diferentes caminos al atravesar los nodos de Internet, TCP *(Transmission Control Protocol)* se encargará de ordenarlos secuencialmente y volver a pedir aquellos que se hayan perdido. Otro aspecto importante que añade TCP es el concepto de **puerto** lo que permite que una misma máquina disponga de múltiples

conexiones abiertas al mismo tiempo. La conjunción de TCP/IP provee al proceso de un canal fiable de comunicación bidireccional *(full-duplex)* con el proceso remoto y es la base del modelo cliente/servidor (Figura 14.11).

- Utiliza `bsocket_connect` en el proceso cliente para crear un canal de comunicación con un servidor remoto.
- Utiliza `bsocket_server` en el proceso servidor para quedar a la escucha de peticiones de clientes.
- Utiliza `bsocket_accept` para aceptar la petición de un cliente y empezar la comunicación.
- Utiliza `bsocket_read` para leer datos de un socket.
- Utiliza `bsocket_write` para escribir datos en un socket.

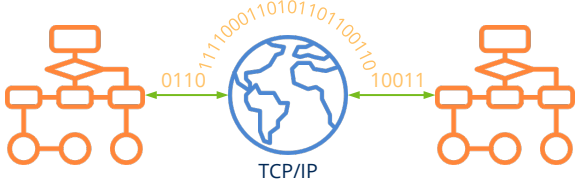


Figura 14.11: Los sockets TCP/IP permiten conectar dos procesos a través de Internet.

Los sockets son la primitiva de comunicación de más bajo nivel accesible por las aplicaciones. Son extremadamente rápidos pero, por lo general, sus funciones son bloqueantes, es decir, detendrán al proceso hasta que el otro interlocutor responda.

- `bsocket_connect` detendrá al proceso cliente hasta que el servidor responda o se cumpla el `timeout`.
- `bsocket_accept` detendrá al proceso servidor hasta que llegue una petición de un cliente o se cumpla el `timeout`.
- `bsocket_read` detendrá al proceso hasta que el otro interlocutor escriba datos en el canal o se cumpla el `timeout`.
- `bsocket_write` detendrá al proceso hasta que el otro interlocutor lea datos del canal y libere el búfer intermedio o se cumpla el `timeout`.

Al margen de estas indicaciones, trabajar con *sockets* es muy parecido a hacerlo con archivos en disco. La implementación de TCP/IP es complicada y forma parte del sistema operativo, por lo que el establecimiento de la conexión se ha simplificado a través de las llamadas al sistema vistas anteriormente. Ya que un socket tan solo permite enviar y recibir bytes, ambos interlocutores necesitan definir un **protocolo** que indique el orden, secuencia y tipo de datos a compartir de tal forma que la comunicación sea satisfactoria y libre de interbloqueos. Algunos de los protocolos más utilizados en Internet son: HTTP, SMTP, FTP, SSH, etc.

14.7.1. Ejemplo Cliente/Servidor

Como ejemplo vamos a ver como dos procesos intercambian información mediante sockets. El protocolo es extremadamente sencillo. Tras la conexión el cliente (Listado 14.11) enviará una serie de valores numéricos al servidor (Listado 14.10) y este le responderá reenviando el mismo valor. Cuando el cliente envíe el valor `UINT32_MAX` la comunicación terminará.

Listado 14.10: Sencillo servidor basado en sockets.

```
uint32_t client_id = 0;
Socket *server_sock = bsocket_server(3444, 32, NULL);

if (server_sock == NULL)
    return;

for(;;)
{
    Socket *income_sock = NULL;
    uint32_t ip0, ip1;
    uint16_t p0, p1;

    bstd_printf("Waiting for a new client\n");

    income_sock = bsocket_accept(server_sock, 0, NULL);
    if (income_sock == NULL)
        continue;

    bstd_printf("Client %d arrives\n", client_id);
    bsocket_local_ip(income_sock, &ip0, &p0);
    bsocket_remote_ip(income_sock, &ip1, &p1);
    bstd_printf("Local IP: %s:%d\n", bsocket_ip_str(ip0), p0);
    bstd_printf("Remote IP: %s:%d\n", bsocket_ip_str(ip1), p1);

    for (;;)
    {
        byte_t data[4];
        uint32_t rsize;
        if (bsocket_read(income_sock, data, sizeof(data), &rsize, NULL) == TRUE
            ↪ )
        {
            uint32_t i;
            bsocket_ntoh4((byte_t*)&i, data);
            if (i != UINT32_MAX)
            {
                bstd_printf("Readed %d from client\n", i);
                bsocket_hton4(data, (byte_t*)&i);
                if (bsocket_write(income_sock, data, sizeof(data), NULL, NULL)
                    ↪ == TRUE)
                {
                    bstd_printf("Sending %d to client\n", i);
```

```
                }
                else
                {
                    bstd_printf("Error writting to client\n");
                    break;
                }
            }
            else
            {
                bstd_printf("Client %d say bye!\n", client_id);
                break;
            }
        }
        else
        {
            bstd_printf("Error reading from client\n");
            break;
        }
    }

    bstd_printf("\n\n");
    bsocket_close(&income_sock);
    client_id += 1;
}

bsocket_close(&server_sock);
```

Listado 14.11: Proceso cliente.

```
Socket *sock = NULL;
serror_t error;
uint32_t i = 0;
byte_t data[4];

sock = bsocket_connect(bsocket_str_ip("192.168.1.21"), 3444, 5000, &error);

if (sock == NULL)
{
    bstd_printf("Connection error\n");
    return;
}

bsocket_read_timeout(sock, 2000);
bsocket_write_timeout(sock, 5000);

while (i < kPING_COUNTER)
{
    bsocket_hton4(data, (const byte_t*)&i);
    if (bsocket_write(sock, data, sizeof(data), NULL, NULL) == TRUE)
    {
        bstd_printf("Sending %d to server\n", i);
```

```
    }
    else
    {
        bstd_printf("Error writting in socket\n");
        break;
    }

    if (bsocket_read(sock, data, sizeof(data), NULL, NULL) == TRUE)
    {
        uint32_t j;
        bsocket_ntoh4((byte_t*)&j, data);
        bstd_printf("Readed %d from server\n", j);
        if (j != i)
        {
            bstd_printf("Error data corruption\n");
            break;
        }

        i += 1;
    }
    else
    {
        bstd_printf("Error reading in socket\n");
        break;
    }
}

if (i == kPING_COUNTER)
{
    i = UINT32_MAX;
    bsocket_hton4(data, (const byte_t*)&i);
    if (bsocket_write(sock, data, sizeof(data), NULL, NULL) == TRUE)
    {
        bstd_printf("Sending FINISH to server\n");
    }
    else
    {
        bstd_printf("Error writting in socket\n");
    }
}

bsocket_close(&sock);
```

14.8. Tiempo

El sistema operativo mide el paso del tiempo utilizando un reloj interno, implementado típicamente mediante un contador de los *ticks* que han pasado desde un instante inicial denominado *epoch*. En sistemas tipo Unix este contador representa el número de segundos transcurridos desde el 1 de Enero de 1970 UTC. Sin embargo, en Windows representa el

número de intervalos de 100 nanosegundos desde el 1 de Enero de 1601 coincidiendo con el inicio del calendario Gregoriano. En NAppGUI se han unificado estos valores para trabajar con el *Unix Epoch* en todas las plataformas.

- Utiliza `btime_now` para obtener el número de micro-segundos trascurridos desde el 1 de Enero de 1970 UTC.
- Utiliza `btime_date` para obtener la fecha del sistema.
- Utiliza `btime_to_micro` y `btime_to_date` para convertir fechas en Unix Time y viceversa.

Figura 14.12: Instante 0 del Unix Epoch.

La diferencia entre dos instantes nos dará el tiempo transcurrido durante la ejecución de una tarea.

```
uint64_t ed, st = btime_now();

// Do something...
...

ed = btime_now();
bstd_printf("Total elapsed micro-seconds: %lu\n", ed - st);
```

14.9. Log

Un *log* o diario es un registro de anomalías que ocurren en tiempo de ejecución y que ayudan a la posterior depuración del programa o a determinar la causa de un error (Figura 14.13). Este informe está dirigido en mayor medida a los programadores o administradores del software y no al usuario final, por lo que es recomendable incluir información técnica específica sobre la causa del problema. Los mensajes dirigidos al usuario final deben estar escritos en un tono más amigable, lejos de tecnicismos y enviados a la salida estándar (`stdout stderr`) o al sistema de ventanas, si estamos ante una aplicación de escritorio.

- Utiliza `log_printf` para escribir un mensaje en el diario de ejecución.

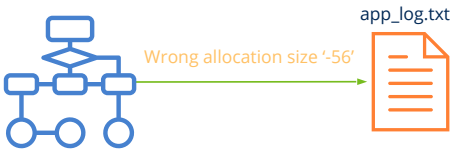


Figura 14.13: Mensajes relativos a anomalías internas del programa, pueden enviarse a un registro *log*.

Capítulo 15

Librería Core

Un core fuerte mejorará tu técnica, fuerza, resistencia y complementará todo lo que hagas.

Susan Trainor

15.1. Core

De igual forma que un edificio necesita una cimentación fuerte, cualquier aplicación o librería debe sustentarse sobre pilares robustos y eficientes. Es inútil invertir horas y horas en una bonita interfaz si el motor interno está quebrado. Para este cometido ha sido desarrollada la librería *core* (Figura 15.1). Proveé estructuras, utilidades y algoritmos de uso común en programación, que nos facilitarán el desarrollo de programas garantizando la máxima eficiencia y portabilidad. *Core* supone el tercer nivel dentro del SDK de NAppGUI y aún no tiene conocimiento sobre las posibilidades gráficas del sistema operativo, por lo que puede utilizarse en la implementación de cualquier tipo de proyecto.

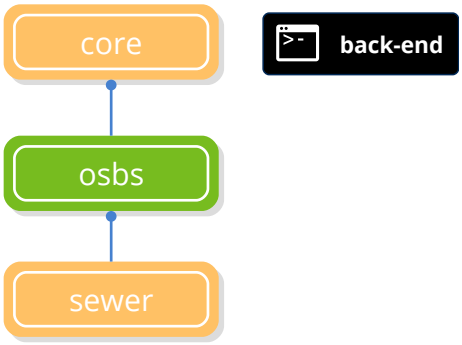


Figura 15.1: Dependencias de *core*. Ver *"NAppGUI API"* (Página 147).

Los servicios que proporciona *core* se han dividido en una serie de módulos.

- *"Heap - Gestor de memoria"* (Página 192).
- *"Buffers"* (Página 196).
- *"Strings"* (Página 196).
- *"Streams"* (Página 197).
- *"Arrays"* (Página 212).
- *"Árboles binarios de búsqueda"* (Página 222).
- *"Expresiones regulares"* (Página 227).
- *"Data binding"* (Página 230).
- *"Eventos"* (Página 235).
- *"Operaciones con archivos"* (Página 236).
- *"Relojes"* (Página 239).

15.2. Heap - Gestor de memoria

Heap es gestor y auditor de memoria dinámica muy eficiente incluido en la librería *core* y disponible para todos los proyectos basados en NAppGUI (librerías y aplicaciones). Es habitual que las aplicaciones soliciten gran cantidad de pequeños bloques de memoria para albergar diferentes objetos (cadenas de caracteres, controles de interfaz, instancias de estructuras, búfer E/S, etc). La estrategia detrás de gestor no es otra que pedir al sistema operativo páginas de memoria de cierto tamaño (64kb o más) mediante `bmem_malloc` y utilizarlas para resolver varias solicitudes de manera muy eficiente.

- Utiliza `heap_new` para crear dinámicamente un objeto.
- Utiliza `heap_malloc` para reservar un bloque de memoria.
- Utiliza `heap_delete` para destruir un objeto.
- Utiliza `heap_free` para liberar un bloque de memoria.

```
Product *product = heap_new(Product);
byte_t *memblock = heap_malloc(1024, "MyOwnBlock");

// Do something
...

heap_delete(&product, Product);
heap_free(&memblock, "MyOwnBlock");
```

El uso de **Heap** en lugar de llamadas al sistema nos proporcionará ciertos beneficios:

- Rendimiento: Una llamada a `heap_malloc` se resuelve tan solo incrementando el valor de un contador. `heap_free` únicamente actualiza la cabecera de la página afectada.
- Localidad: Dos llamadas consecutivas a `heap_malloc()` son ubicadas en posiciones físicas de memoria contiguas. Esto reduce el número de fallos de caché ya que, según el principio de localidad, hay una gran probabilidad de que dos objetos que se crean juntos se utilicen juntos.
- Memory leaks: *heap* apunta las reservas y liberaciones por tipo de objeto. Llegado el caso, avisará al programador por medio de *"Asserts"* (Página 155) o *"Log"* (Página 186) de que existen objetos no liberados. La gran ventaja de este auditor sobre otras herramientas es que siempre se está ejecutando como parte del programa. Esto explota la coherencia temporal, ya que si tras un cambio se detectan *leaks* donde antes no había, es muy probable que podamos acotar y detectar el error, ya que será algo en lo que acabamos de trabajar.
- Estadísticas: Podemos obtener perfiles del uso de memoria (tiempo/bytes). Esto puede ayudarnos a detectar cuellos de botella (especialmente en el arranque) u optimizar

el tamaño de página.

15.2.1. Memoria múlti-hilo

Por defecto, *heap* está configurado para funcionar de forma óptima en aplicaciones mono-hilo. Si queremos que varias hebras del mismo proceso reserven o liberen memoria dinámica de forma concurrente y segura deberemos utilizar:

- `heap_start_mt` para iniciar el soporte multi-hilo.
- `heap_end_mt` para terminar el soporte multi-hilo.

En el momento que se llama a `heap_start_mt`, se activan los mecanismos de sincronización dentro del Heap para garantizar la exclusión mutua al gestor de memoria hasta que se reciba una llamada a `heap_end_mt` que lo devolverá al modo de funcionamiento mono-hilo. Llamadas sucesivas a `heap_start_mt` se irán acumulando, por lo que permanecerá en modo multi-hilo hasta que se cierren todos los bloques abiertos (Listado 15.1). Es responsabilidad del programador utilizar este par de funciones en aquellos puntos del programa que lo requieran.

Toda sección que comienza con `heap_start_mt` debe cerrarse con `heap_end_mt`.

No hay ningún problema en activar el soporte multi-hilo en secciones mono-hilo, salvo una ligera penalización en el rendimiento.

Listado 15.1: Secciones multi-hilo.

```
// Single-threaded block
...
...

heap_start_mt();
// Multi-threaded block
...
heap_start_mt();
...
heap_end_mt();
// Continue multi-threaded block
...
heap_end_mt();

// Single-threaded block
...
```

15.2.2. Como funciona Heap

Al iniciar el programa, *heap* crea una página de memoria por defecto. Los primeros bytes se reservan como cabecera, una pequeña estructura que controla el estado de la página. Cada petición se asigna secuencialmente dentro de la misma página, incrementando el valor de un puntero (Figura 15.2). Cuando la página se queda sin espacio, se crea una nueva mediante `bmem_malloc`, que es enlazada con la anterior y etiquetada como la nueva **página por defecto** (Figura 15.3). Cada llamada a `heap_free` actualiza la cabecera con el número de bloques/bytes liberados (Figura 15.4). Estos bloques **no son reutilizados**, de lo contrario se complicaría la lógica de *heap* volviéndolo más lento. La dirección de la cabecera es guardada al final de cada bloque, por lo que no hay que iterar para localizarlo. Cuando todos los bloques de la página han sido liberados, la página entera es destruida por `bmem_free` y los punteros entre páginas vecinas restaurados (Figura 15.5).

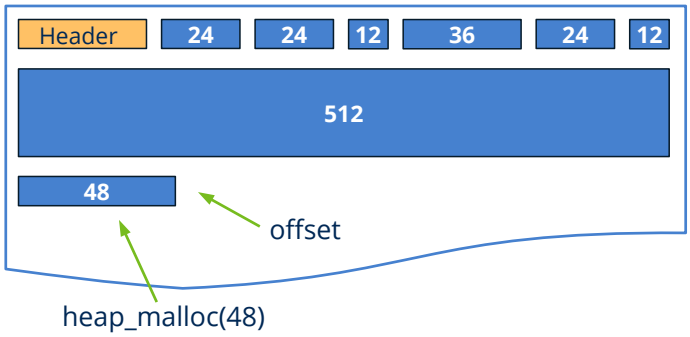


Figura 15.2: Reserva de un nuevo bloque de memoria con `heap_malloc()`.

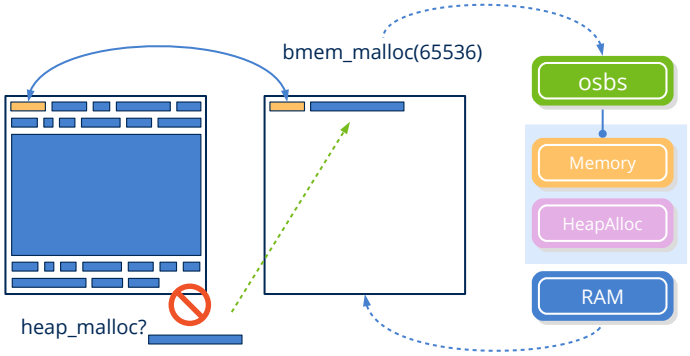


Figura 15.3: Petición al sistema operativo de una nueva página vacía.

Heap también contabiliza el número de reversas/liberaciones por tipo de objeto mediante el parámetro `name` de `heap_malloc`. Al finalizar la ejecución del programa, si la aplicación carece de *memory leaks*, escribirá en *"Log"* (Página 186) un mensaje como este:

```
[12:58:08] [OK] Heap Memory Staticstics
[12:58:08] ============================
[12:58:08] Total a/dellocations: 1126, 1126
[12:58:08] Total bytes a/dellocated: 74611, 74611
[12:58:08] Max bytes allocated: 54939
```

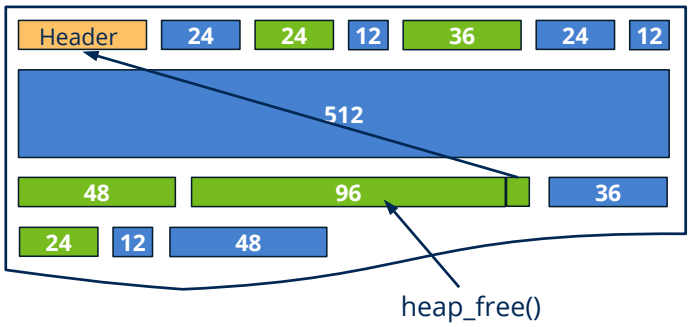


Figura 15.4: Liberando un bloque de memoria (solo actualiza la cabecera).

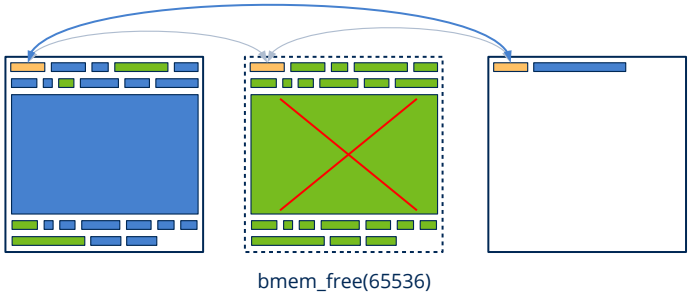


Figura 15.5: Destruyendo la página entera.

```
[12:58:08] Effective reallocations: (0/34)
[12:58:08] Real allocations: 2 pages of 65536 bytes
[12:58:08] ============================
```

Pero si tras la ejecución, la aplicación tiene memoria por liberar, el mensaje será distinto:

```
[13:00:35] [FAIL] Heap Object Leaks!!!
[13:00:35] ==========================
[13:00:35] 'App' a/deallocations: 1, 0 (1 leaks)
[13:00:35] 'String' a/deallocations: 414, 410 (4 leaks)
[13:00:35] ==========================
[13:00:35] [FAIL] Heap Global Memory Leaks!!!
[13:00:35] ==================================
[13:00:35] Total a/dellocations: 1161, 1156 (5 leaks)
[13:00:35] Total bytes a/dellocated: 75704, 75596 (108 bytes)
[13:00:35] Max bytes allocated: 54939
[13:00:35] ==================================
```

Que advierte que tenemos un objeto `App` y cuatro `String` sin liberar. Si en la ejecución anterior no había *leaks*, es muy probable que podamos acotar el error sin demasiada dificultad.

El auditor de heap no pretende sustituir a herramientas más avanzadas de testeo de memoria, tan solo es un primer filtro que nos avisa constantemente durante la fase de desarrollo y test. A pesar de que la sobrecarga que produce en tiempo de ejecución es mínima, el auditor se deshabilita complemente en la configuración `Release`.

15.3. Buffers

Los objetos *Buffer* son, sencillamente, bloques de memoria reservados dinámicamente y almacenados en el *"Segmento HeapSegmento Heap"* (Página 165). Son útiles para compartir datos genéricos entre diferentes funciones o hebras. Para este último caso, deben estar protegidos por un `Mutex` si varios hilos pueden acceder a él de forma concurrente (no son *thread-safe*). Son de tamaño fijo. Una vez creados no se pueden re-dimensionar, aunque pueden ser re-escritos cuantas veces sea necesario.

- Utiliza `buffer_create` para crear un bloque de memoria dinámica.
- Utiliza `buffer_destroy` para liberar un bloque de memoria dinámica.
- Utiliza `buffer_data` para obtener un puntero al bloque de memoria.

15.4. Strings

Los objetos **String** contienen cadenas de caracteres *"UTF-8UTF-8"* (Página 160) reservadas dinámicamente. A pesar que es posible insertar cadenas de texto estáticas en el código fuente o acceder a ellas a través de los paquetes de recursos (`respack_text`), suele ser necesario componer textos en tiempo de ejecución o almacenar dinámicamente cadenas recibidas por algún canal de entrada (teclado, archivos, red, etc). El módulo `strings.h` de NAppGUI ofrece multitud de funciones para trabajar con cadenas de texto UTF8, tanto estáticas como dinámicas.

- Utiliza `str_c` para crear una copia dinámica de una cadena C estática.
- Utiliza `str_printf` para componer una cadena dinámica utilizando el mismo formato que el `printf` de C.
- Utiliza `tc` para obtener un puntero `const char_t*` al contenido de un `String`.

```
String *str1 = str_c("This a static char array.");
String *str2 = str_printf("Code: %s, Price %8.2f.", tc(product->code),
    ↪ product->price);
const char_t *cstr1 = tc(str1);
const char_t *cstr2 = tc(str2);
cstr1 = "This a static char array."
cstr2 = "Code: 456-34GH-JKL, Price   439.67."
```

No confundir los objetos `String` con las cadenas de texto en C `const char_t *str` ó `char_t str[128]`. Los primeros contienen un puntero al área de memoria dinámica y un entero con el número de bytes reservados.

En el caso que sea necesario componer textos más extensos a partir de bucles, la forma más eficiente es crear un `Stream` y, posteriormente, obtener el `String` asociado.

```
String *str = NULL;
Stream *stm = stm_memory(2048);
uint32_t n = arrpt_size(products, Product);
stm_printf(stm, "List of %d products\n", n);
arrpt_foreach(product, products, Product);
    stm_printf(stm, "Code: %s, Price %8.2f.\n", tc(product->code), product->
        ↪ price);
arrpt_end();
str = stm_str(stm);
stm_close(&stm);

// Do something with 'str'
...

str_destroy(&str);
```

15.5. Streams

Un *stream* es un flujo de datos que corre desde un origen a un destino. Piensa en una llamada telefónica. Tenemos un origen (la persona que habla), un destino (la persona que escucha) y un canal (la propia línea). En programación, el stream es el equivalente a la línea telefónica, es la tubería que une la aplicación con un origen o destino de datos (Figura 15.6) y por la que circula información binaria, secuencias de bits. Al igual que ocurre con cualquier otro canal de comunicación, la información es volátil, disponible por un tiempo muy limitado. Una vez que llega al receptor, desaparece.

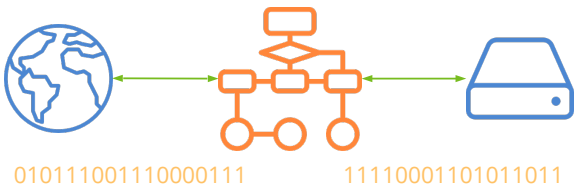


Figura 15.6: Los streams conectan al proceso con la máquina y con el mundo.

En esencia, son tres las operaciones elementales a realizar cuando trabajamos con streams: Crear el canal, leer datos y escribir datos.

- Utiliza `stm_memory` para crear un stream de lectura/escritura en memoria.
- Utiliza `stm_read_r32` para leer un `float` desde el stream.

- Utiliza `stm_write_r32` para escribir un `float` en el stream.
- Utiliza `stm_close` para cerrar el canal y liberar los recursos (destructor).

15.5.1. Tipos de stream

En realidad, es más correcto hablar de tipos de interlocutores (origen y destino) que de tipos de stream. Desde la perspectiva del programador, un stream es un tipo abstracto que presenta la misma funcionalidad independientemente de los extremos que conecta. Por tanto, al hablar de *tipo de stream* nos estamos refiriendo al tipo de constructor.

15.5.2. File stream

En *File streams* (Figura 15.7), el origen es la memoria del proceso y el destino un archivo en disco. Lo contrario también puede ocurrir: que el origen sea el archivo y el destino la memoria, dependerá de como creemos el canal. No será posible realizar operaciones de escritura en un archivo abierto para lectura o viceversa (Listado 15.2). Internamente, el stream se encargará del acceso al sistema de ficheros, como vimos en *"Archivos y directorios"* (Página 179).

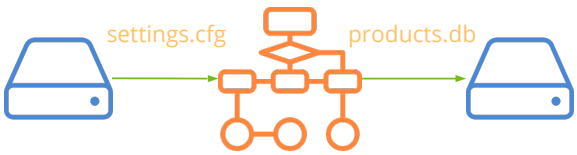


Figura 15.7: Los *File streams* permiten la comunicación con el sistema de archivos.

- Utiliza `stm_from_file` para abrir un archivo y leer de él.
- Utiliza `stm_to_file` para crear un archivo y escribir en él.
- Utiliza `stm_append_file` para añadir contenido a un archivo existente.

Listado 15.2: Ejemplo de escritura en un archivo.

```
Stream *stm = stm_to_file("C:\Users\user\john\out.txt", NULL);
if (stm != NULL)
{
    stm_writef(stm, "One ");
    stm_writef(stm, "Two ");
    stm_writef(stm, "Three");
    stm_writef(stm, ".");
    stm_close(&stm);
    // 'out.txt' is closed = "One Two Three."
}
```

15.5.3. Socket stream

Un *socket* es un canal de comunicación entre dos procesos a través de Internet (Figura 15.8). A diferencia de los *file streams*, los sockets permiten la comunicación bidireccional *(full-duplex)*, es decir, ambos extremos pueden enviar y recibir información. La secuencia de intercambio de mensajes entre interlocutores está determinada por el protocolo (Listado 15.3), siendo HTTP, FTP, SMTP o LDAP algunos de los más utilizados para las transmisiones por Internet. Ver *"Sockets"* (Página 181).

Figura 15.8: Un *socket stream* abre un canal de comunicación por Internet.

- Utiliza `stm_socket` para crear un canal de comunicación con un proceso remoto.

Listado 15.3: Descarga de una página web, mediante el protocolo HTTP.

```
uint32_t ip = bsocket_url_ip("www.myserver.com", NULL);
Socket *socket = bsocket_connect(ip, 80, 0, NULL);
if (socket != NULL)
{
    Stream *stm = stm_socket(socket);
    stm_writef(stm, "GET /mypage.html HTTP/1.1\r\n");
    stm_writef(stm, "Host: www.myserver.com\r\n");
    stm_writef(stm, "\r\n");
    stm_lines(line, stm)
        bstd_printf(line);
        bstd_printf("\n");
    stm_next(line, stm);

    // Socket will be closed too
    stm_close(&stm);
}
```

15.5.4. Block stream

Los *block streams* se utilizan para leer datos con formato a partir de un bloque de memoria genérico. (Figura 15.9). Esta zona de memoria se considera de solo lectura y no será modificada, por lo que no se permitirán operaciones de escritura en este tipo de streams. Cuando se llegue al final del bloque, se activará el estado `ekSTEND`.

- Utiliza `stm_from_block` para leer datos desde un bloque de memoria.

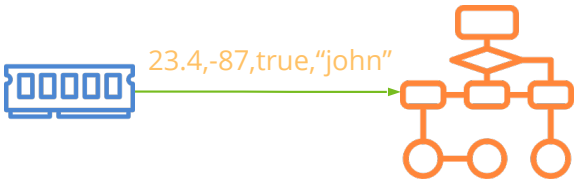


Figura 15.9: Con *block streams* leeremos datos con formato desde áreas de memoria.

15.5.5. Memory stream

Los *memory streams* son canales de lectura/escritura que permiten implementar el modelo productor/consumidor (Figura 15.10). En primer lugar, la información llega al stream mediante operaciones de escritura y se almacena en un búfer interno de memoria. Posteriormente, dicha información puede ser leída por otra función, hebra o proceso. Tras cada lectura la información leída desaparecerá del canal. El concepto es similar al de las tuberías *(IPC-Pipes)*, con la salvedad de que no existe un límite de tamaño para el búfer, sino que crecerá bajo demanda. Las operaciones de lectura y escritura se pueden ir simultaneando en función del protocolo establecido.

- Utiliza `stm_memory` para crear un stream en memoria.
- Utiliza `stm_buffer` para acceder al búfer interno.

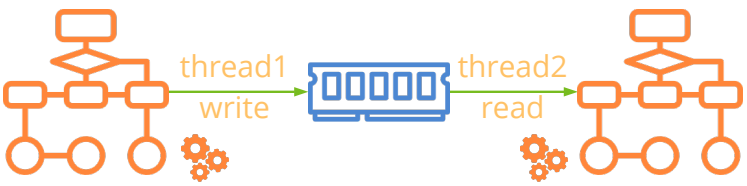


Figura 15.10: Modelo productor/consumidor implementado con *memory streams*.

A pesar que este tipo de stream soporta operaciones de lectura y escritura no se considera full-duplex. La lectura se hace sobre datos escritos previamente, pero no puede "contestar" al interlocutor. No es una "conversación".

15.5.6. Standard stream

La *"E/S Estándar"* (Página 163) puede ser gestionada mediante *streams* utilizando tres objetos predefinidos (Figura 15.11). Estos objetos se crean al iniciar el programa y se liberarán automáticamente al terminar.

- Utiliza `kSTDIN` para leer de la entrada estándar.
- Utiliza `kSTDOUT` para escribir en la salida estándar.
- Utiliza `kSTDERR` para escribir en la salida de errores.

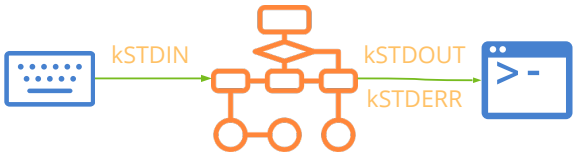


Figura 15.11: Acceso a la E/S estándar a través de streams.

```
real64_t value;
const char_t *line;
value = stm_read_r64(kSTDIN);
line = stm_read_line(kSTDIN);
stm_printf(kSTDOUT, "Value = %.4f", value);
```

15.5.7. Null stream

En ocasiones puede ser útil disponer de un "sumidero" que ignore todas las operaciones de escritura (Figura 15.12). Piensa en tareas de depuración donde queramos activar o desactivar la salida de información pero que borrar o comentar el código sea engorroso. La idea es similar al `/dev/null` de Unix.

- Utiliza `kDEVNULL` para escribir en un sumidero que ignorará todos los datos recibidos.

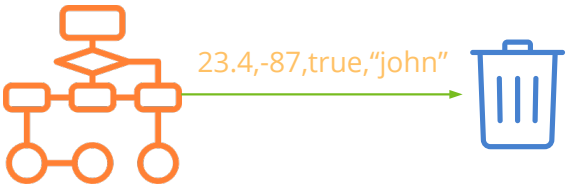


Figura 15.12: Con *null streams* se ignorará todo lo que se escriba.

```
#if defined __ASSERTS__
Stream *stm = kSTDOUT;
#else
Stream *stm = kDEVNULL;
#endif

...
i_large_dump_func(obj1, stm);
...
// More debug functions
stm_printf(stm, "More debug data...\n");
...
i_other_dump_func(obj2, stm);
```

No es posible leer desde ***kDEVNULL****.*

15.5.8. Stream binario

Por un stream siempre viajan datos binarios genéricos como flujo de bytes. Como estos datos sean interpretados depende de los interlocutores y de su protocolo de comunicación. Pero al hacer énfasis en "datos binarios" nos referimos a que los valores numéricos se escriben en el canal tal y como aparecen en los registros de la CPU utilizando el código binario, complemento a dos o IEEE754 (Figura 15.13). En tipos multi-byte debemos tener en cuenta el *"Orden de bytesOrden de bytes"* (Página 211). En `stream.h` se definen varias funciones para leer y escribir tipos binarios.

- Utiliza `stm_read_u32` para leer un entero sin signo de 32 bits.
- Utiliza `stm_write_r64` para escribir un real de 64bits (`double`).
- Utiliza `stm_write_bool` para escribir un booleano.

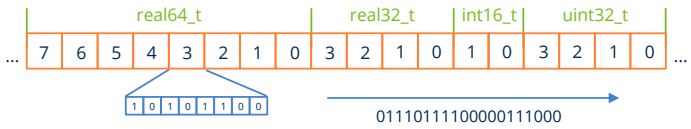


Figura 15.13: Números en formato binario.

15.5.9. Stream de texto

Los streams de texto son un caso particular donde se asume que la información binaria representa códigos de caracteres Unicode *(codepoints)* (Figura 15.14) (Listado 15.4). Esto significa que el contenido del stream será legible directamente por un humano, pero requerirá un post-proceso (*parsing*) en destino para interpretar dichos textos y traducirlos a binario. No hay que hacer nada especial al crear un stream para indicar que es de tipo texto, tan solo hay que utilizar las funciones adecuadas.

- Utiliza `stm_printf` para escribir texto en un stream.
- Utiliza `stm_read_char` para leer un carácter desde un stream.
- Utiliza `stm_read_line` para leer una línea de texto desde un stream.
- Utiliza `stm_col` para obtener el número de columna del último carácter leído.
- Utiliza `stm_row` para obtener el número de fila del último carácter leído.

Figura 15.14: En streams de texto la información se puede leer directamente.

Listado 15.4: Lectura de un archivo de texto mediante streams.

```
Stream *stm = stm_from_file("/home/fran/Desktop/text.txt", NULL);
const char_t *line = stm_read_line(stm);
```

```
while(line != NULL)
{
    // Do something with 'line'
    textview_writef(text, line);
    textview_writef(text, "\n");

    // Read next line
    line = stm_read_line(stm);
}

stm_close(&stm);
```

`stm_read_line` y demás funciones de lectura nos devolverán el texto siempre en `UTF8` . Pero si los datos dentro del stream estuvieran en otro formato, deberemos utilizar `stm_set_read_utf`, con el fin de realizar la conversión correctamente. Ver *"Codificaciones UTFCodificaciones UTF"* (Página 159).

Por otro lado, `stm_printf` también recibe el texto en UTF8, pero es posible que el receptor lo necesite en otro formato. Utilizaremos `stm_set_write_utf` para establecer la codificación de salida. Nosotros escribiremos en UTF8, pero al canal se enviará en UTF16 o UTF32.

> *Los streams no tienen porqué ser de texto o binarios "puros". Pueden combinar ambos tipos de representaciones.*

15.5.10. Tokens

Al leer desde streams de texto, se hace necesaria una interpretación (*parsing*) del contenido con el fin de trasladar los datos a variables de memoria (en binario). El primer paso es fragmentar el texto en símbolos (o palabras) denominados *tokens*. Internamente, los streams incorporan un sencillo **analizador léxico** que reconoce los tokens propios del lenguaje C, muy comunes en infinidad de gramáticas y formatos de archivo (Figura 15.15). Está implementado como una máquina de estados finitos y nos facilitará enormemente el procesamiento de estos flujos de texto. En (Listado 15.5) vemos el código necesario para leer uno a uno todos los tokens de un archivo `.c`. El resultado de procesar el archivo (Listado 15.6) lo tenemos en (Listado 15.7).

- Utiliza `stm_read_token` para leer un token.
- Utiliza `stm_token_lexeme` para obtener la cadena asociada al último token leído.
- Utiliza `stm_read_r64_tok` para leer un `real64_t` a partir de texto.
- Utiliza `stm_token_col` para obtener la columna del último token.
- Utiliza `stm_token_row` para obtener la fila del último token.

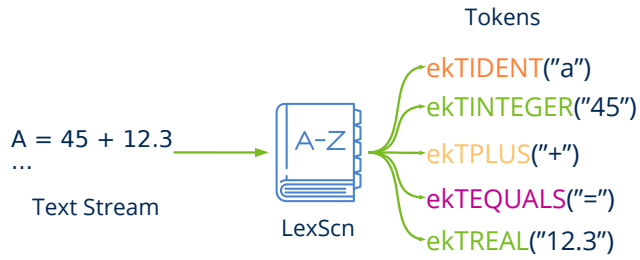


Figura 15.15: Descomposición de un texto en *tokens*.

Listado 15.5: Lectura de los *tokens* de un archivo en C.

```
Stream *stm = stm_from_file("source.c", NULL);
token_t token;

while ((token = stm_read_token(lex)) != ekTEOF)
{
    switch (token) {
    case ekTIDENT:
        // It's an IDENTIFIER
        ...

    case ekTREAL:
        // It's a REAL NUMBER
        ...
    }
}
```

Listado 15.6: Archivo *source.c*.

```
void func(int a)
{
    int i;
    char *str = "Hello";

    i = 5 + 2.5;
}
```

Listado 15.7: Análisis léxico de *source.c*.

```
Token           Lexeme
-----           ------
ekTIDENT        "void"
ekTIDENT        "func"
ekTOPENPAR      "("
ekTIDENT        "int"
ekTIDENT        "a"
ekTCLOSPAR      ")"
ekTOPENCURL     "{"
ekTIDENT        "int"
ekTIDENT        "i"
```

```
ekTSCOLON       ";"
ekTIDENT        "char"
ekTASTERK       "*"
ekTIDENT        "str"
ekTEQUALS       "="
ekTSTRING       ""Hello""
ekTSCOLON       ";"
ekTIDENT        "i"
ekTEQUALS       "="
ekTINTEGER      "5"
ekTPLUS         "+"
ekTREAL         "2.5"
ekTSCOLON       ";"
```

15.5.11. Identificadores

Un identificador es una "palabra" alfanumérica que debe comenzar por una letra o `'_'` y contiene números, letras o `'_'`. Se utiliza para nombrar variables, funciones, palabras reservadas, etc. No permiten espacios ni símbolos. (Listado 15.8) (Figura 15.16).

Listado 15.8: Identificadores correctos e incorrectos.

```
OK: while cos _reSult a56B _06_t aG h9 _12AcVb
NO: 045 ?er "_5G _tg(
```

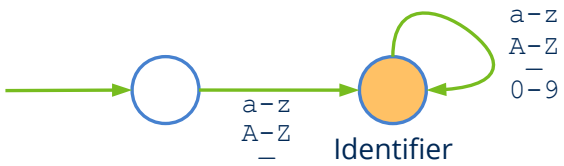


Figura 15.16: Autómata finito que reconoce un identificador.

Se pueden reservar ciertos identificadores para que actúen como **palabras clave** del lenguaje. Por ejemplo `for`, `while` o `if` son palabras clave de C y no podrán ser utilizadas para el nombrado de variables o funciones. Al ser de propósito general, nuestro escáner no reconoce ningún tipo de palabra reservada, sino hay que etiquetarla expresamente tras leer el token (Listado 15.9).

Listado 15.9: Reconociendo la palabra clave **while**.

```
while ((token = stm_read_token(stm)) != ekTEOF)
{
    if (token == ekTIDENT)
    {
        const char_t *lex = stm_token_lexeme(stm, NULL);

        if (str_equ_c(lex, "while") == TRUE)
            token = ekTRESERVED;
```

```
    }
}
```

15.5.12. Cadenas

Una cadena de texto es una serie de caracteres Unicode puestos entre comillas (`""`) (Figura 15.17). El analizador reconoce las secuencias de escape de C para representar códigos no imprimibles o caracteres no disponibles en el teclado (Listado 15.10).

- Utiliza `stm_token_escapes` para hacer efectivas las secuencias de escape al leer cadenas.

Listado 15.10: Secuencias de escape aceptadas en `ekTSTRING`.

```
\a       07  Alert (Beep, Bell) (added in C89)
\b       08  Backspace
\f       0C  Formfeed Page Break
\n       0A  Newline (Line Feed)
\r       0D  Carriage Return
\t       09  Horizontal Tab
\v       0B  Vertical Tab
\\       5C  Backslash
\'       27  Single quotation mark
\"       22  Double quotation mark
\?       3F  Question mark (used to avoid trigraphs)
\nnn         Octal number
\xhh         Hexadecimal number
\Uhhhhhhhh   Unicode code point
\uhhhh       Unicode code point
```

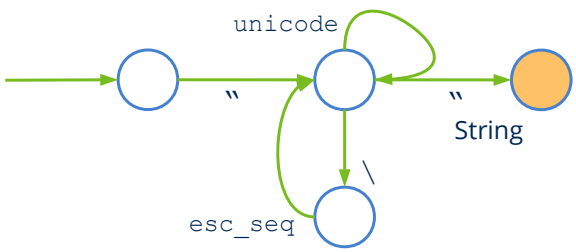


Figura 15.17: Autómata finito que reconoce una cadena de texto.

15.5.13. Números

En el caso de *tokens* numéricos la cosa se complica un poco debido a las diferentes bases numéricas y a la representación exponencial de números reales (Figura 15.18). La resumimos brevemente, aunque es común a muchos lenguajes de programación (C incluido).

- Si el número empieza por `0` será considerado octal (base 8), por lo tanto, las cifras siguientes están limitadas a `0-7`, pe: `043, 001, 0777`.

- Si el número empieza por `0x` se considerará hexadecimal (base 16) con cifras `0-9 a-f A-F`, pe: `0x4F, 0XAA5, 0x01EAC`.
- En el momento que aparezca un punto decimal `'.'` se considerará número real. El punto inicial es válido, pe: `.56`.
- Un número entero o real permite notación exponencial con el carácter `'e'('E')`, pe: `12.4e2, .56e3, 1e4`.

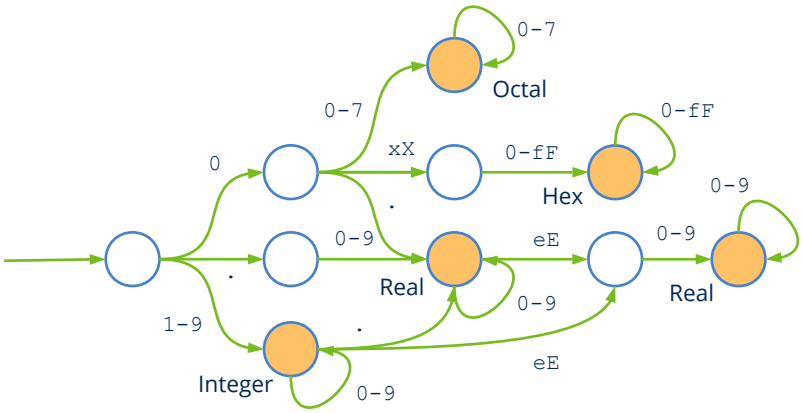


Figura 15.18: Autómata finito que reconoce números.

15.5.14. Símbolos

Los símbolos son *tokens* de un solo carácter que representan casi la totalidad de signos de puntuación US-ASCII y que suelen utilizarse como operadores, separadores o limitadores dentro de las gramáticas (Listado 15.11) (Figura 15.19).

Listado 15.11: Símbolos reconocidos como *tokens* individuales.

```
< > , . ; : ( ) [ ] { } + - * = $ % # & ' " ^ ! ? | / \ @
```

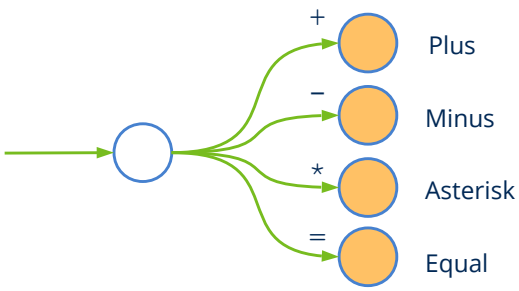


Figura 15.19: Autómata finito que reconoce algunos símbolos.

15.5.15. Comentarios

Por defecto los comentarios de C `/*Comment */` o de C++ `//Comment` son ignorados por `stm_read_token`.

- Utiliza `stm_token_comments` para que devuelva `ekTSLCOM` o `ekTMLCOM` si encuentra alguno.
- Utiliza `stm_token_spaces` para que devuelva `ekTSPACE` cuando encuentre espacios en blanco.

15.5.16. Ventajas de los streams

A pesar que es posible leer o escribir directamente en los canales E/S (*"Memoria"* (Página 164), *"Archivos y directorios"* (Página 179), *"Sockets"* (Página 181), *"E/S Estándar"* (Página 163)), hacerlo a través de objetos `Stream` tiene ciertas ventajas. Por esto recomendamos utilizarlos en lugar de las API de bajo nivel por los siguientes motivos:

15.5.17. Unificar la serialización

Los streams ofrecen una interfaz uniforme, independientemente del origen y destino de los datos (Figura 15.20). Para la serialización de objetos tan solo debemos escribir un lector y un escritor, sin preocuparnos si objeto será guardado en disco, transmitido por Internet o almacenado temporalmente en memoria (Listado 15.12).

Listado 15.12: (De)serialización de un objeto mediante streams.

```
typedef struct _product_t
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
} Product;

void product_write(Stream *stm, Product *product)
{
    stm_write_enum(stm, product->type, type_t);
    str_write(stm, product->code);
    str_write(stm, product->description);
    image_write(stm, product->image64);
    stm_write_r32(stm, product->price);
}

void product_read(Stream *stm, Product *product)
{
    product->type = stm_read_enum(stm, type_t);
```

```
    product->code = str_read(stm);
    product->description = str_read(stm);
    product->image64 = image_read(stm);
    product->price = stm_read_r32(stm);
}
```

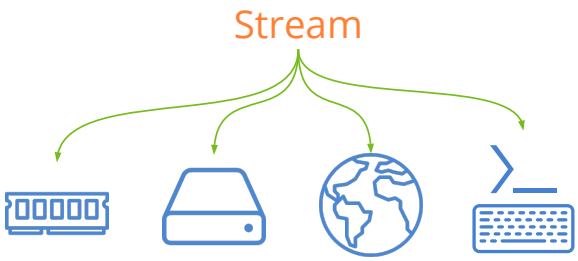


Figura 15.20: Por medio de streams manejamos todos los canales E/S con la misma interfaz.

15.5.18. Mayor elegancia

Los canales E/S solo trabajan con bloques de bytes. Los streams implementan funciones de alto nivel para textos y tipos binarios. Esto hará nuestro código mucho más legible (Listado 15.13).

Listado 15.13: Escritura de un objeto en disco directamente o a través de un stream.

```
void product_write(File *file, Product *product)
{
    uint32_t size = str_len(product->description);
    const char_t *data = tc(product->description);
    bfile_write(file, (byte_t*)&product->id, sizeof(uint32_t), NULL, NULL);
    bfile_write(file, (byte_t*)&product->price, sizeof(real64_t), NULL, NULL);
    bfile_write(file, (byte_t*)&size, sizeof(uint32_t), NULL, NULL);
    bfile_write(file, (byte_t*)data, size, NULL, NULL);
}

void product_write(Stream *stream, Product *product)
{
    stm_write_u32(stream, product->id);
    stm_write_r64(stream, product->price);
    str_write(stream, product->description);
}
```

15.5.19. Mayor productividad

Relacionada con la anterior, los streams pueden "parsear" cadenas de texto directamente. Se pueden obtener caracteres, líneas o tokens sin tener que escanear la entrada carácter a carácter (Listado 15.14).

Listado 15.14: Leer una línea de texto directamente o a través de un stream.

```
String *getline(File *file)
{
    /* Potentially unsafe. */
    /* Risk of buffer overflow. */
    char_t buffer[MAXBUFF];
    uint32_t i = 0;
    char_t c;

    bfile_read(file, (byte_t*)&c, 1, NULL, NULL);
    while (c != '\n')
    {
        buffer[i] = c;
        i += 1;
        bfile_read(file, (byte_t*)&c, 1, NULL, NULL);
    }

    buffer[i] = '\0';
    return str_c(buffer);
}

String *getline(Stream *stream)
{
    /* Totally safe. */
    /* 'line' is managed by dynamic cache. */
    const char_t *line = stm_read_line(stream);
    return str_c(line);
}
```

15.5.20. Mayor rendimiento

File streams y *standard streams* implementan una caché interna. Esto permite acceder menos veces al canal con un mayor volumen de datos, lo que implica mayor velocidad de proceso (Figura 15.21).

- Utiliza `stm_flush` para vaciar la caché y volcar los datos en el canal.

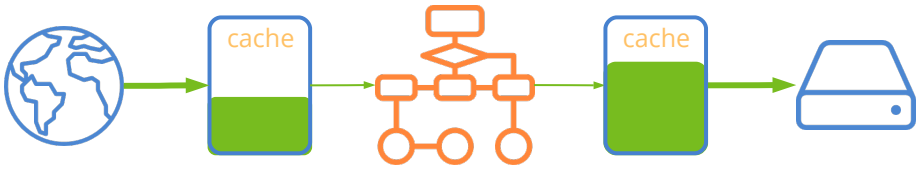


Figura 15.21: Los streams implementan memoria caché, lo que incrementa el rendimiento.

15.5.21. Orden de bytes

Al leer o escribir datos binarios de un canal E/S hay que prestar especial atención al orden de los bytes en tipos de datos de 16, 32 o 64 bits lo que se conoce como *endianness*. En máquinas *litte endian*, como es el caso de la familia de procesadores Intel x86/x64, el byte de menor orden se ubicará en la dirección de memoria más baja. En el caso de los *big endian* (Motorola 68000, PowerPC) ocurre al contrario, irá en en la más alta. Por ejemplo, si escribimos un entero de 32-bits en un archivo o *socket* desde una máquina *little endian* y lo leemos desde una *big endian* los datos estarán corrompidos al alterar el orden interno de bits (Figura 15.22). Los objetos `Stream` ajustan automáticamente el *endianness* en cada operación de lectura/escritura. Por defecto se establece `ekLITEND`, salvo en *sockets* que será `ekBIGEND` por ser el convenio aceptado para las comunicaciones en red. No obstante, puede cambiarse si es necesario.

- Utiliza `stm_set_write_endian` para establecer el *endianness* del canal de salida. Los datos pasarán de *CPU endian* a *Stream endian* antes de ser escritos.
- Utiliza `stm_set_read_endian` para establecer el *endianness* del canal de entrada. Los datos pasarán de *Stream endian* a *CPU endian* en el momento de ser leídos.

Figura 15.22: Debemos tener en cuenta el *endianness* a la hora de compartir datos entre máquinas de diferente arquitectura.

El endianness no influye en las cadenas de texto "UTF-8UTF-8" (Página 160), pero sí en las "UTF-16UTF-16" (Página 159) y "UTF-32UTF-32" (Página 159).

15.5.22. Estado del stream

Un stream puede verse afectado por dos tipos de problemas. Por un lado la **corrupción de datos** que queda patente cuando leemos datos binarios desde el stream. Un claro ejemplo sería leer un booleano mediante `stm_read_bool` y obtener un valor de `129` cuando, evidentemente, este valor debería ser `0` (`TRUE`) ó `1` (`FALSE`). Normalmente, un stream se corrompe debido a la falta de coordinación entre el escritor y el lector y, por lo general, se debe a un error de programación. Esta situación debe resolverse depurando y corrigiendo la serialización de objetos o revisando el protocolo de datos. Por otro lado, pueden existir errores "físicos" en el canal (archivo eliminado, perdida de conexión a Internet, permisos, etc). En ambos casos el stream quedará invalidado y serán ignoradas las operaciones subsecuentes de lectura o escritura que efectuemos sobre el mismo. También podemos preguntar

la cantidad total de bytes leídos y/o escritos en el canal, en el caso que necesitemos saber si existe información disponible para lectura.

- Utiliza `stm_state` para conocer el estado actual del canal.
- Utiliza `stm_file_err` para obtener información extendida del error en streams de disco.
- Utiliza `stm_sock_err` para obtener información extendida del error en *sockets*.
- Utiliza `stm_corrupt` para marcar un stream como `ekSTCORRUPT`. En ocasiones es el propia aplicación la que detecta que el dato no es correcto (pe. fuera de rango).
- Utiliza `stm_bytes_written` para obtener el número total de bytes escritos en el stream.
- Utiliza `stm_bytes_readed` para obtener el número total de bytes leídos del stream.

```
uint32_t nw = stm_bytes_written(stm);
uint32_t nr = stm_bytes_readed(stm);
if (nw - nr > 0)
{
    if (stm_state(stm) == ekSTOK)
    {
        uint32_t v1 = stm_read_u32(stm);
        real32_t v2 = stm_read_r32(stm);
        ...
    }
    else
    {
        // Error in stream
    }
}
else
{
    // No data in stream
}
```

15.6. Arrays

Poder trabajar con colecciones de datos es algo fundamental a la hora de diseñar nuestro modelo. Además de los tipos básicos y los `struct`, `union` o `class` el lenguaje C nos ofrece la construcción *array*, que permite almacenar varios elementos bajo un mismo nombre de variable (Listado 15.15):

Listado 15.15: Arrays en C.

```
typedef struct _product_t Product;
struct _product_t
```

```
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
};

DeclSt(Product);
DeclPt(Product);

uint32_t integers[100];
real32_t reals[100];
Product products[100];
```

O, de forma dinámica (Listado 15.16):

Listado 15.16: Arrays dinámicos.

```
uint32_t n = get_n();
uint32_t *integers = heap_new_n(n, uint32_t);
real32_t *reals = heap_new_n(n, real32_t);
Product *products = heap_new_n(n, Product);
```

Los **arrays de C** guardan los elementos en posiciones contiguas de memoria y, aunque son muy rápidos de consultar, carecen de funcionalidad para insertar, borrar, buscar u ordenar. En muchas ocasiones, los datos no están disponibles cuando se crea el contenedor, sino que se van entrando o saliendo de forma dinámica durante la ejecución del programa, por lo que no podemos prever de antemano un número máximo con el que realizar la reserva de memoria. El tipo `Array` implementado en NAppGUI es, en esencia, un array de C dinámico más una serie de métodos para manipularlo. Por dinámico entendemos que la estructura ajusta su tamaño a la cantidad real de elementos, conservando la premisa principal de que **todos permanezcan juntos en memoria**.

Cuando se crea un `Array`, se reserva memoria para unos pocos registros (Figura 15.23). Posteriormente, podremos añadir nuevos elementos al final (lo típico) o insertarlos en cualquier posición aleatoria en el caso de que ya tengamos datos en el contenedor. En este último caso, el resto de elementos serán desplazados hacia la derecha. En el momento que se supere la cantidad de registros reservados, se duplicará el bloque dinámico interno para dar cabida a las nuevas posiciones. De igual forma es posible eliminar cualquier elemento de la colección, desplazando el resto hacia la izquierda para mantener la coherencia espacial de la estructura. Si el número de items decreciese a la mitad, el bloque de memoria se reducirá. De esta forma, durante el tiempo de vida del contenedor, la memoria se irá ajustando multiplicando o dividiendo por 2 el número de elementos reservados.

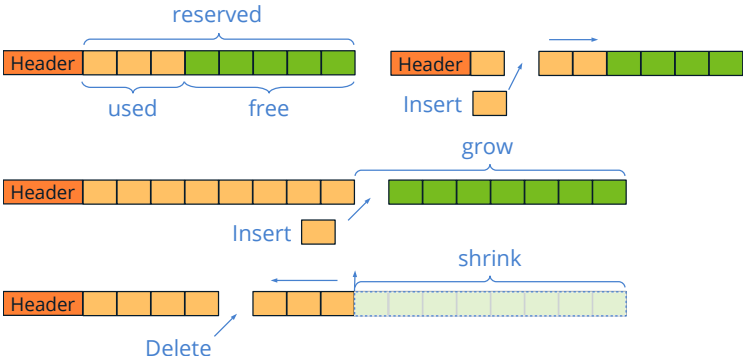


Figura 15.23: Los `Array` adaptarán su memoria interna al número de elementos.

15.6.1. Registros o punteros

Un objeto tipo `Product`, nuestra estructura de ejemplo, ocupa 20 bytes en sistemas de 32bits (Figura 15.24). Los campos `code`, `description` e `image64` son punteros que apuntan a otras zonas de memoria, donde residen los campos tipo `String` e `Image` reservados dinámicamente.

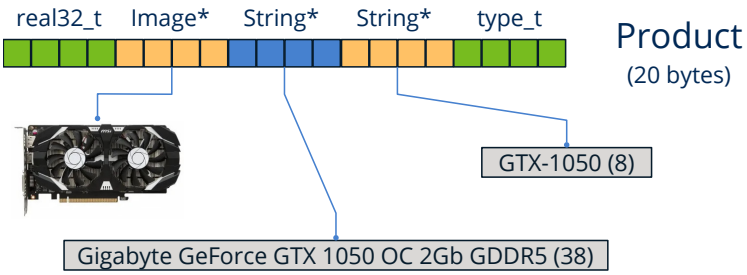


Figura 15.24: Objeto Product.

En función de lo que se almacene dentro del contenedor, podemos utilizar dos clases de array (Listado 15.17). Los array de registros guardarán la totalidad del objeto (los 20 bytes) y los array de punteros solo una referencia al mismo (4 bytes), estando el objeto real ubicado en otra dirección de memoria (Figura 15.25). A pesar que la gestión interna de la estructura es la misma, el acceso a los elementos difiere ligeramente.

- Utiliza `arrst_create` para crear un array de registros.
- Utiliza `arrpt_create` para crear un array de punteros.

Listado 15.17: Creación de un array.

```
ArrSt(Product) *arrst = arrst_create(Product);
ArrPt(Product) *arrpt = arrpt_create(Product);
```

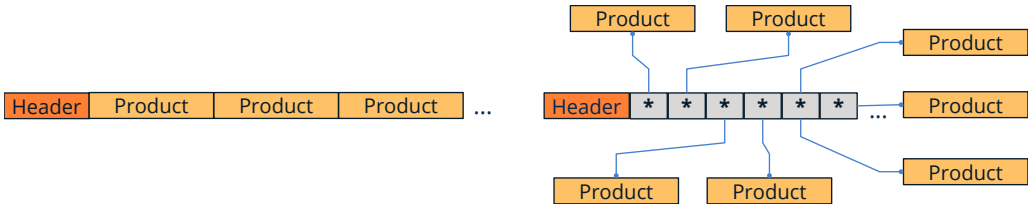


Figura 15.25: Arrays de registros y de punteros.

Utilizar `ArrSt` puede mejorar ligeramente el rendimiento, gracias a la coherencia espacial, que reduce los fallos de caché, y al ahorro de llamadas al gestor de memoria *"Comparativa Arrays vs SetsComparativa Arrays vs Sets"* (Página 226). Pero esto no siempre será posible, y no podremos utilizarlos en estos casos:

- Objetos opacos: Si la definición del tipo no es pública, el contenedor no puede calcular el espacio necesario para cada elemento, por lo que solo podremos trabajar con punteros a los mismos.
- Objetos compartidos: Si otras estructuras del modelo mantienen punteros a los elementos del contenedor, tendremos problemas del tipo *Segmentation Fault* debido al cambio de las direcciones de memoria al reubicar el bloque interno del contenedor (Figura 15.26).

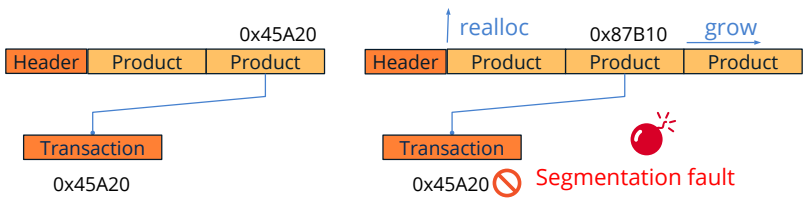


Figura 15.26: Los arrays de registros son peligrosos con referencias externas.

15.6.2. Comprobación de tipos

Habrás observado en (Listado 15.15) que aparecen dos sentencias justamente después de la definición del `struct Product`: `DeclSt` y `DeclPt`. Son dos macros que habilitan la comprobación de tipos en tiempo de compilación, definiendo una interfaz personalizada en los contenedores para este nuevo tipo (Listado 15.6.2). Salvando las distancias, imitan las `template<>` de C++. `DeclSt` habilita los contenedores de registros y `DeclPt` los de punteros.

```
Product *p1 = arrst_new(Product);
Product *p2 = arrst_get(arrst, 5, Product);
const Product *p3 = arrst_get_const(arrst, 5, Product);
```

Aunque no es aconsejable, puedes prescindir del uso de estas macros y utilizar las interfaces "en crudo" de los contenedores, definidas en `array.h` y `rbtree.h`. En este caso tu código será mucho menos legible y no tendrás el soporte del compilador.

Las cabeceras `array.h` y `rbtree.h` no están documentadas.

15.6.3. Constructores

Cuando se reserva memoria para un objeto, bien sea como **variables automáticas** en el *"Segmento StackSegmento Stack"* (Página 164)

```
Product product;
```

en el *"Segmento HeapSegmento Heap"* (Página 165), mediante memoria dinámica

```
Product *product = heap_new(Product);
```

o en un contenedor

```
Product *product = arrst_new(array, Product);
```

el contenido inicial del mismo es basura, entendido como bytes indeterminados. Inicializar un objeto es asignar valores válidos y coherentes a cada campo del mismo (Listado 15.18).

Listado 15.18: Inicializando un objeto `Product`.

```
static void i_init(Product *product)
{
    product->type = ekCPU;
    product->code = str_c("");
    product->description = str_c("");
    product->image64 = image_copy(gui_image(NOIMAGE_PNG));
    product->price = 0.f;
}
```

Por su parte, un constructor es un inicializador que previamente reserva memoria de forma dinámica para almacenar el objeto (Listado 15.19).

Listado 15.19: Constructor de un objeto `Product`.

```
static Product *i_create(void)
{
    Product *product = heap_new(Product);
    i_init(product);
    return product;
}
```

Cuando utilizamos arrays de registros, solo necesitaremos inicializar el objeto, ya que el espacio para almacenarlo ha sido reservado por el propio contenedor (Listado 15.20). Sin embargo, en arrays de punteros, la memoria para el objeto debe ser reservada explícitamente, ya que el contenedor solo guardará una referencia.

Listado 15.20: Insertar objetos correctamente inicializados.

```
// Add an item using an automatic variable (a copy is required)
Product product;
i_init(&product);
arrst_append(array, product, Product);

// Add an item directly (avoiding copying)
Product *product = arrst_new(array, Product);
i_init(product);

// Add a pointer to a newly created object on the heap
Product *product = i_create();
arrpt_append(array, product, Product);
```

Utilizar `arrst_new`, `arrst_insert_n` o `arrst_prepend_n` siempre que sea posible para insertar en arrays de registros, ya que evitan una copia del objeto.

15.6.4. Recorrido de un array

Para iterar sobre todos los elementos del array, podemos elegir entre dos tipos de sintaxis para implementar el bucle.

```
uint32_t i, n = arrst_size(arrst, Product);
for (i = 0; i < n; ++i)
{
    const Product *product = arrst_get(arrst, i, Product);

    // Do something
    ...
}

arrst_foreach(product, arrst, Product)
    // Do something
    ...
arrst_end();

// In reverse order
arrst_foreach_rev(product, arrst, Product)
    // Do something
    ...
arrst_end();
```

15.6.5. Copia de objetos

De forma similar a los constructores, existen dos métodos para copiar objetos (Listado 15.21). En el primero de ellos generamos memoria dinámica para los campos del objeto, pero no para el propio objeto bien por ser una variable automática o estar almacenado en un array de registros. En el segundo caso, reservamos memoria dinámica tanto para el objeto como para sus elementos.

Listado 15.21: Copiando un objeto `Product`.

```
static void i_copy_data(Product *dest, const Product *src)
{
    dest->type = src->type;
    dest->code = str_copy(src->code);
    dest->description = str_copy(src->description);
    dest->image64 = image_copy(src->image64);
    dest->price = src->price;
}

static Product *i_copy(const Product *product)
{
    Product *new_product = heap_new(Product);
    i_copy_data(new_product, product);
    return new_product;
}

ArrSt(Product) *arrst = arrst_copy(arrst_src, i_copy_data, Product);
ArrPt(Product) *arrpt = arrpt_copy(arrpt_src, i_copy, Product);
```

15.6.6. Serialización

Un caso especial de constructor son los **readers** (de-serializadores). Cuando creamos un array a partir del contenido de *"Streams"* (Página 197) (Listado 15.22), necesitamos un método capaz de crear o inicializar un elemento desde el propio stream. Dependiendo del tipo de contenedor será necesario reservar memoria para cada elemento o no.

Listado 15.22: Leyendo un array desde un stream.

```
static void i_read_data(Stream *stm, Product *product)
{
    product->type = stm_read_enum(stm, type_t);
    product->code = str_read(stm);
    product->description = str_read(stm);
    product->image64 = image_read(stm);
    product->price = stm_read_r32(stm);
}

static Product *i_read(Stream *stream)
{
```

```
    Product *product = heap_new(Product);
    i_read_data(stream, product);
    return product;
}

ArrSt(Product) *arrst = arrst_read(i_read_data, Product);
ArrPt(Product) *arrpt = arrpt_read(i_read, Product);
```

De igual forma podemos escribir (serializar) el contenido de un array en un stream de escritura (Listado 15.23). En este caso, una sola función de escritura es suficiente para ambos tipos de contenedores, ya que cada uno sabe como acceder a sus elementos.

Listado 15.23: Escribiendo un array en un stream.

```
static void i_write(Stream *stm, const Product *product)
{
    stm_write_enum(stm, product->type, type_t);
    str_write(stm, product->code);
    str_write(stm, product->description);
    image_write(stm, product->image64);
    stm_write_r32(stm, product->price);
}

arrst_write(stm, arrst, i_write, Product);
arrpt_write(stm, arrpt, i_write, Product);
```

15.6.7. Destructores

En programación muchas veces nos crean confusión los verbos: *'delete', 'destroy', 'free', 'erase', 'remove', 'clear'* ya que en esencia significan lo mismo pero con pequeños matices. En NAppGUI utilizaremos un verbo u otro en función de acciones concretas:

- **Free:** Solo libera la memoria dinámica asignada a un objeto (Listado 15.24). Necesita un doble puntero, ya que el objeto será invalidado (=NULL) tras liberarlo, evitando referencias a zonas de memoria liberadas.

Listado 15.24: Liberando la memoria de un objeto.

```
Product *product = heap_new(Product);
...
heap_free(&product, Product);
// product = NULL
```

- **Remove:** Destruye los campos de un objeto, pero no libera la memoria del propio objeto. Es el opuesto al *inicializador* (Listado 15.25).

Listado 15.25: Liberando la memoria de los campos del objeto.

```
static void i_remove(Product *product)
```

```
{
    str_destroy(&product->code);
    str_destroy(&product->description);
    image_destroy(&product->image64);
}

arrst_destroy(&arrst, i_remove, Product);
```

- **Destroy:** La combinación de las dos anteriores. Destruye los campos del objeto y libera su memoria (Listado 15.26). Es el opuesto al constructor. Obviamente, requiere un doble puntero para invalidar la referencia.

Listado 15.26: Libera la memoria del objeto y todo su contenido.

```
static void i_destroy(Product **product)
{
    i_remove(*product);
    heap_free(product, Product);
}

arrpt_destroy(&arrpt, i_destroy, Product);
```

- **Delete:** Borra un elemento de un array u otro tipo de contenedor (Listado 15.27). Puede tener asociado un destructor o *'remover'*, aunque no es obligatorio.

Listado 15.27: Elimina un elemento de un contenedor.

```
// Just delete.
arrst_delete(arrst, 4, NULL, Product);

// Delete and remove (arrst).
arrst_delete(arrst, 4, i_remove, Product);

// Delete and destroy (arrpt).
arrpt_delete(arrpt, 4, i_destroy, Product);
```

- **Clear:** Borra todos los elementos de un contenedor, pero no lo destruye, solo lo deja a cero (Listado 15.28). Al igual que `arrst_delete`, opcionalmente puede liberar memoria de los elementos.

Listado 15.28: Elimina todos los elementos de un contenedor.

```
// Just delete all.
arrst_clear(arrst, NULL, Product);

// Delete and remove all (arrst).
arrst_clear(arrst, i_remove, Product);

// Delete and destroy all (arrpt).
arrpt_clear(arrpt, i_destroy, Product);
```

15.6.8. Ordenar y buscar

La forma habitual de utilizar arrays será ir añadiendo elementos al final mediante `arrst_new` o `arrpt_append` para después iterar sobre el conjunto. Este orden "natural" será suficiente en la mayoría de casos, pero es posible que necesitemos organizar los elementos siguiendo otro criterio para:

- Presentar la información ordenada por uno o varios campos de la estructura.
- Optimizar búsquedas. Para localizar un determinado elemento, no hay más remedio que recorrer todo el array, con coste lineal `O(n)`. Pero podemos resolver la búsqueda en tiempo logarítmico `O(logn)` si el array está ordenado, incrementando drásticamente el rendimiento especialmente en conjuntos grandes (Figura 15.27).

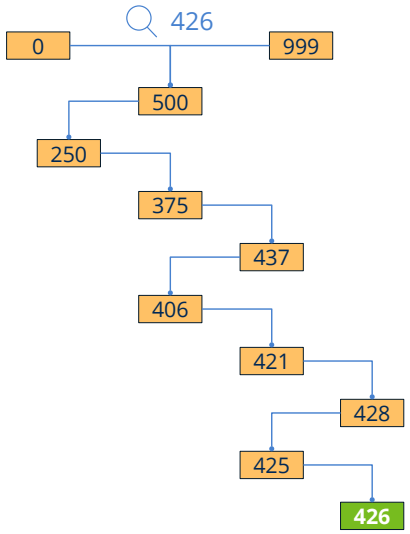


Figura 15.27: En un máximo de 10 pasos encontraremos un elemento entre mil (20 pasos para un millón).

- Utiliza la función `arrst_sort`, para ordenar un array. Tendremos que pasar una función de comparación, que es la que determinará la relación de orden (Listado 15.29).

Listado 15.29: Ordenar arrays por código de producto.

```
static int i_compare(const Product *p1, const Product *p2)
{
    return str_scmp(p1->code, p2->code);
}

arrst_sort(arrst, i_compare, Product);
arrpt_sort(arrpt, i_compare, Product);
```

Para buscar un elemento dentro de un array, también deberemos proporcionar una función que compare el objeto con una clave. Esta clave contiene el criterio de búsqueda

y, normalmente, es más reducida que el elemento en sí. Muchas veces no es más que un simple número o una cadena de texto (Listado 15.30).

- `arrst_search` Método lento. Buscará elementos de forma lineal, uno a uno `O(n)`.
- `arrst_bsearch` Método rápido. Buscará elementos de forma logarítmica, `O(logn)`. El array debe estar ordenado bajo el mismo criterio que la búsqueda.

Listado 15.30: Búsqueda de un elemento por su código.

```
static int i_compare_key(const Product *p1, const char_t *key)
{
    return str_cmp(p1->code, key);
}

uint32_t pos;
Product *pr1, *pr2;
// Slow O(n)
pr1 = arrst_search(arrst, i_compare_key, "G3900", &pos, Product, char_t);

// Fast O(logn)
pr2 = arrst_bsearch(arrst, i_compare_key, "G3900", &pos, Product, char_t);
```

15.6.9. Arrays de tipos básicos

Los tipos básicos son un caso particular de estructura de un solo campo, por lo utilizaremos `ArrSt`. En el caso concreto de `enum` deberemos crear un alias mediante `typedef`, ya que `ArrSt(type)` no admite el *keyword* `enum`, de igual forma que tampoco admite el *keyword* `struct`. En C++ este alias no es necesario. Al destruir el array pasaremos `NULL` al parámetro destructor, ya que los tipos básicos no generan memoria dinámica.

```
typedef enum _type_t type_t;
ArrSt(uint32_t) *integers = arrst_create(uint32_t);
ArrSt(type_t) *types = arrst_create(type_t);
arrst_destroy(&integers, NULL, uint32_t);
arrst_destroy(&types, NULL, type_t);
```

15.7. Arrays (punteros)

15.8. Árboles binarios de búsqueda

Al igual que los *array* los **binary search trees (BST)**, también conocidos como conjuntos (`set`) o mapas (`map`), son contenedores que nos permiten trabajar con una colección de objetos. La principal diferencia con respecto a los primeros es que los elementos no se almacenan de forma lineal en posiciones contiguas de memoria, sino que utilizan una

estructura en forma de árbol donde cada nodo tiene dos descendientes (Listado 15.31) (Figura 15.28).

Listado 15.31: Creación de arrays y sets.

```
typedef struct _product_t Product;
struct _product_t
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
};

static int i_compare(const Product *p1, const Product *p2)
{
    return str_scmp(p1->code, p2->code);
}

ArrSt(Product) *arrst = arrst_create(Product);
ArrPt(Product) *arrpt = arrpt_create(Product);
SetSt(Product) *setst = setst_create(i_compare, Product);
SetPt(Product) *setpt = setpt_create(i_compare, Product);
```

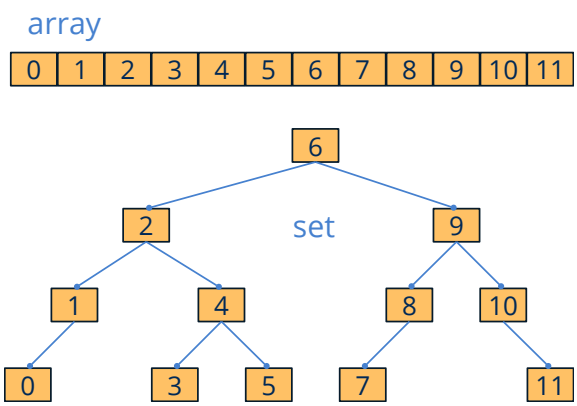


Figura 15.28: Representación de arrays y sets.

Los *BST* son estructuras optimizadas para casos donde sean muy frecuentes las inserciones, borrados y búsquedas. Están permanentemente ordenados, de ahí que sea posible insertar, eliminar o localizar cualquier elemento en tiempo logarítmico `O(logn)`, sin necesidad de utilizar funciones de ordenación como `arrst_sort` (Figura 15.29). Para que el mantenimiento se pueda llevar a cabo de forma eficiente, el árbol que soporta la estructura debe cumplir una serie de características:

- **Binario**: Cada nodo solo puede tener 0, 1 ó 2 hijos.
- **Ordenado**: Todos los descendientes a la izquierda de un nodo son de menor valor y

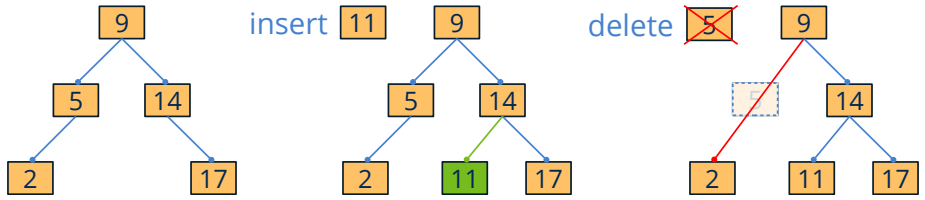


Figura 15.29: En árboles de búsqueda la inserción o borrado no rompe el orden del conjunto.

los situados a la derecha de mayor valor. El criterio de orden y búsqueda se establece en el constructor mediante una función de comparación (`i_compare` en el ejemplo anterior) y no se puede cambiar durante el tiempo de vida del contenedor. Los nuevos elementos se insertarán en su posición correcta conforme a este orden. No soporta elementos duplicados ni en posiciones arbitrarias.

- **Balanceado**: Un árbol puede cumplir las dos propiedades anteriores, pero haber degenerado a una lista donde las búsquedas ya no pueden resolverse en tiempo logarítmico (Figura 15.30). Internamente, los contenedores `Set` de NAppGUI están implementados con los llamados *árboles rojo-negro*, donde se garantiza una altura máxima de `2log(n+1)`. Esto se consigue reestructurando el árbol después de cada inserción o borrado, por lo que añadir un nuevo elemento (o eliminarlo) se resuelve en un máximo de `O(logn)`. Esto es muchísimo más rápido que en arrays, donde hay que desplazar todos los elementos para insertar un registro en una posición concreta, con un coste asociado de `O(n)`.

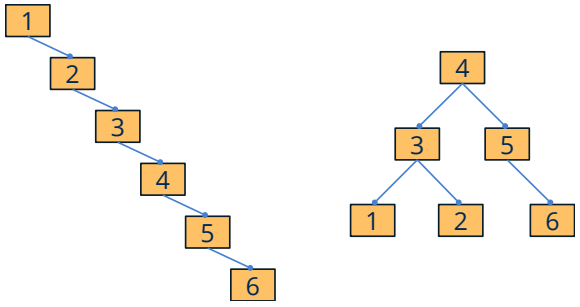


Figura 15.30: Árbol de búsqueda degenerado y balanceado.

Como ya vimos en *"Registros o punterosRegistros o punteros"* (Página 214), tenemos dos modalidades a la hora de crear conjuntos (Figura 15.31). La versión basada en registros es más eficiente que la basada en punteros, aunque menos flexible.

- Utiliza `setst_create` para crear un conjunto de registros.
- Utiliza `setpt_create` para crear un conjunto de punteros.

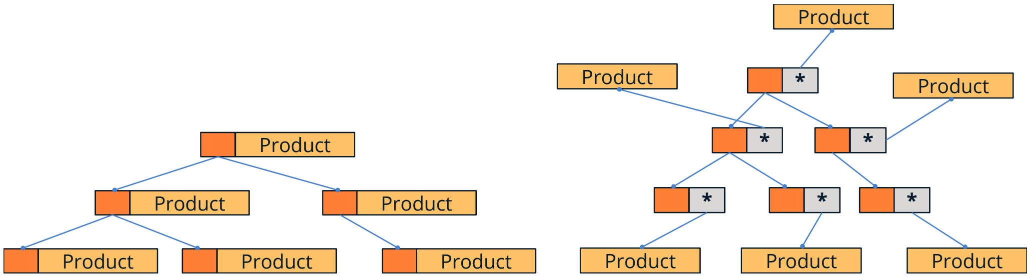


Figura 15.31: Conjuntos de registros y de punteros.

15.8.1. Iteradores

No podemos acceder a los elementos de un set mediante un índice aleatorio, como ocurría con los arrays. Los nodos están dispersos en diferentes zonas de memoria, lo que impide calcular la posición de un elemento concreto a partir de una dirección base. Un iterador no es más que un puntero dentro del set que actúa como marcador del elemento seleccionado actualmente (Figura 15.32). A partir de una posición concreta, podemos desplazarnos al elemento anterior o posterior, pero nunca dar saltos arbitrarios. Podemos controlar la posición del iterador con diferentes funciones (Listado 15.32):

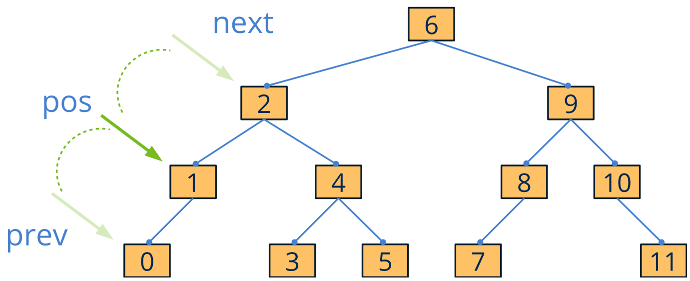


Figura 15.32: Los iteradores nos permiten movernos por la estructura.

- Utiliza `setst_get` para buscar un elemento. El iterador quedará fijado en él.
- Utiliza `setst_next` para desplazar el iterador al siguiente elemento.
- Utiliza `setst_prev` para desplazar el iterador al elemento anterior.
- Utiliza `setst_first` para desplazar el iterador al primer elemento del set.
- Utiliza `setst_last` para desplazar el iterador al último elemento del set.

Listado 15.32: Iterando sobre los elementos de un *set*.

```
const Product *product = setst_first(setst, Product);
while (product != NULL)
{
    // Do something
    ...
```

```
    product = setst_next(setst, Product);
}

setst_foreach(product, setst, Product)
    // Do something
    ...
setst_fornext(product, setst, Product)

// In reverse order
setst_forback(product, setst, Product)
    // Do something
    ...
setst_forprev(product, setst, Product)
```

15.8.2. Comparativa Arrays vs Sets

Hemos realizado un test para ver el comportamiento de estos dos tipos de estructuras en situaciones reales, al margen de la mera teoría (Tabla 15.1). Se ha utilizado la estructura `Product` descrita en (Listado 15.31). Compararemos seis tipos de contenedores `ArrSt(Product)` y `ArrPt(Product)` (desordenados), `ArrSt(Product)` y `ArrPt(Product)` (ordenados), `SetSt(Product)` y `SetPt(Product)`.

- Los elementos se ordenarán por el campo `code` utilizando el método `i_compare` descrito en (Listado 15.31).
- Los elementos han sido creados previamente y residen en memoria. Los tiempos solo reflejan la gestión realizada por los contenedores.
- El campo `code` toma valores desde `"0"` hasta `"n-1"`, donde `n=100.000` es el número de elementos. Los elementos han sido previamente desordenados utilizando la función `bmem_shuffle_n`.
- Las pruebas se han realizado en una **Raspberry Pi 3 Model B** con NAppGUI compilado en versión Release (*"ConfiguracionesConfiguraciones"* (Página 97)). Hemos elegido esta plataforma por su clara inferioridad técnica con respecto a otras. De esta forma la diferencia asintótica resulta más evidente.

Operation	ArrSt	ArrPt	ArrSt-Sort	ArrPt-Sort	SetSt	SetPt
Add(100k)	0.006	0.004	27.600	2.896	0.159	0.274
Loop(100k)	0.000	0.000	0.000	0.000	0.022	0.025
Search(100k)	84.139	588.080	0.101	0.218	0.121	0.232
Sort(100k)	0.085	0.205	-	-	-	-

Operation	ArrSt	ArrPt	ArrSt-Sort	ArrPt-Sort	SetSt	SetPt
Delete(100k)	0.004	0.003	31.198	3.064	0.171	0.253

Tabla 15.1: Resultados de la comparativa (en segundos).

A la vista de estos datos, podemos llegar a las siguientes conclusiones:

- Las búsquedas lineales `O(n)` son tremendamente lentas.
- Mantener un array ordenado tras cada inserción o borrado es costoso. Es más eficiente añadir todos los elementos y luego ordenar, aunque esto no siempre será posible. Si los elementos entran o salen de forma arbitraria pero el conjunto debe siempre estar ordenado, es mejor utilizar Sets.
- Las estructuras basadas en registros son más eficientes en consultas, pero menos al insertar o borrar. No obstante, este test no incluye el tiempo de crear o liberar memoria dinámica, algo inherente a los contenedores de punteros.
- Iterar en arrays sale prácticamente gratis, pero iterar en sets tiene un pequeño coste debido a la lógica de salto entre nodos.
- No podemos decir que un contenedor sea mejor que otro en general. Dependerá de cada caso concreto.
- Para grupos pequeños (menos de 1000 elementos) las diferencias son prácticamente imperceptibles.
- Para grupos extremadamente pequeños (hasta 100 elementos) utilizar siempre arrays. La mejora asintótica de los Sets se ve empañada por la implementación mucho más eficiente de los Arrays.

15.9. Árboles binarios de búsqueda (punteros)

15.10. Expresiones regulares

Las expresiones regulares definen un patrón de texto que puede utilizarse para buscar o comparar cadenas (Listado 15.33).

- Utiliza `regex_create` para crear una expresión regular.
- Utiliza `regex_match` para comprobar si una cadena cumple con el patrón.

Listado 15.33: Uso de expresiones regulares.

```
RegEx *regex = regex_create(".*.txt");

const char_t *str[] = {
    "file01.txt",
```

```
    "image01.png",
    "sun01.jpg",
    "films.txt",
    "document.pdf"};

uint32_t i, n = sizeof(str) / sizeof(char_t*);

for (i = 0; i < n; ++i)
{
    if (regex_match(regex, str[i]) == TRUE)
        bstd_printf("YES: %s\n", str[i]);
    else
        bstd_printf("NO:  %s\n", str[i]);
}

regex_destroy(&regex);
```

Resultado de (Listado 15.33).

```
YES: file01.txt
NO:  image01.png
NO:  sun01.jpg
YES: films.txt
NO:  document.pdf
```

15.10.1. Definir patrones

Podemos construir una expresión regular a partir de una cadena de texto, siguiendo estas sencillas reglas:

- Un patrón cadena se corresponde únicamente con esa misma cadena.

```
"hello" --> {"hello"}
```

- Un punto `'.'` equivale a "cualquier carácter".

```
"h.llo" --> {"hello", "htllo", "hällo", "h5llo", ...}
```

- Un guión `'A-Z'` establece un rango de caracteres, utilizando el código Unicode de ambos extremos.

```
"A-Zello" --> {"Aello", "Bello", "Cello", ..., "Zello"}

'A-Z': (65-90) (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
'0-9': (48-57) (0123456789)
'á-ú': (225-250) (áâãäåæçèéêëìíîïðñòóôõö÷øùú)
```

Al igual que los objetos `String`, los patrones se expresan en "UTF-8UTF-8" (Página 160), por tanto, puede utilizarse todo el conjunto Unicode para crear expresiones regulares.

- Los corchetes `'[áéíóú]'` permiten alternar entre varios caracteres.

```
"h[áéíóú]llo" --> {"hállo", "héllo", "híllo", "hóllo", "húllo"}
```

- El asterisco `'*'` permite que el último carácter aparezca cero o más veces.

```
"he*llo" --> {"hllo", "hello", "heello", "heeello", "heeeello", ...}
"h.*llo" --> {"hllo", "hello", "hallo", "hillo", "hasello", ...}
"hA-Z*llo" --> {"hllo", "hAllo", "hABllo", "hVFFRREASllo" }
           --> {"hAQWEDllo", hAAABBRSllo", ...}
"FILE_0-9*.PNG" --> {"FILE_.PNG", "FILE_0.PNG", "FILE_01.PNG" }
                --> {"FILE_456.PNG", "FILE_112230.PNG",...}
```

- Los paréntesis `'(he*llo)'` permiten agrupar una expresión regular, de tal forma que se comporte como un único carácter.

```
"[(hello)(bye)]" --> {"hello", "bye" }
"[(red)(blue)(1*)]" --> {"red", "blue", "", "1", "11", "111", ... }
"(hello)*" --> {"", "hello", "hellohello", "hellohellohello", ... }
"(he*llo)ZZ" --> {"hlloZZ", "helloZZ", "heelloZZ", "heeelloZZ", ... }
```

- Para que `'.', '-', '[]', '*', '()'` sean interpretados como carácteres, utilizar el *backslash* `'\'`.

```
"\(he\*\-llo\)" --> {"(he*-llo)"}
```

Recuerda que para expresiones insertadas como constantes en código C, el carácter backslash se representa con doble barra `"\\(he\\*\\-llo\\)"`.

15.10.2. Lenguajes regulares y autómatas

Los lenguajes regulares son aquellos que se definen de forma recursiva utilizando tres operaciones básicas sobre el conjunto de carácteres (o símbolos) disponibles. Se pueden describir mediante las expresiones regulares comentadas anteriormente.

- Cada carácter **'a'** es un lenguaje regular **'A'**.
- La unión de dos lenguajes regulares, es un lenguaje regular **A B**.
- La concatenación de dos lenguajes regulares, es un lenguaje regular **A · B**.
- La clausura de un lenguaje regular, es un lenguaje regular **A***. Aquí es donde entra en juego la recursión.

En este contexto los símbolos son todos los carácteres Unicode. Pero pueden definirse lenguajes basados en otros alfabetos, incluido el binario {0, 1}.

Para reconocer si una cadena pertenece o no a un determinado lenguaje regular, es necesario construir un **Autómata Finito** basándonos en las reglas reflejadas en (Figura 15.33).

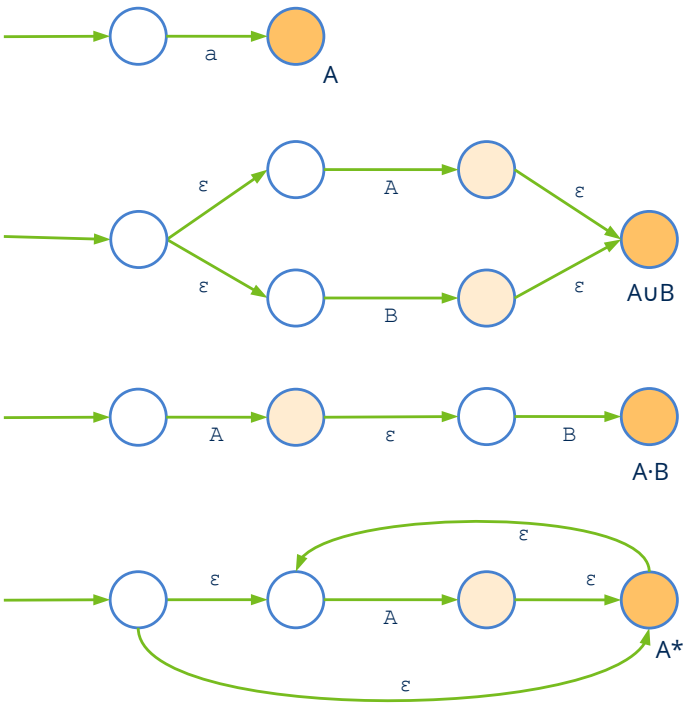


Figura 15.33: Construcción de autómatas finitos para filtrar expresiones regulares.

15.11. Data binding

Entendemos por *Data Binding* (vinculación de datos o enlace de datos) la posibilidad de sincronizar automáticamente las estructuras del programa con diferentes fuentes de entrada/salida. Partimos del sencillo modelo (Listado 15.34) que presentamos en *"Arrays"* (Página 212) compuesto por un `struct` y un `enum`.

Listado 15.34: Sencillo modelo de datos basado en `struct`.

```
typedef struct _product_t Product;

typedef enum _type_t
{
    ekCPU,
    ekGPU,
    ekHDD,
```

```
    ekSCD
} type_t;

struct _product_t
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
};
```

Lo primero que tenemos que hacer, es registrar este modelo en **dbind**, una especie de "base de datos" general dentro de nuestra aplicación (Listado 15.35). Únicamente es necesario realizar este proceso una vez al arrancar el programa. De esta forma se crearán unas tablas internas con la descripción de cada estructura del modelo de datos (Figura 15.34), quedando el programa preparado para automatizar ciertas tareas al trabajar con objetos de dichas clases.

- Utiliza `dbind` para registrar los campos de una estructura.
- Utiliza `dbind_enum` para registrar los diferentes valores de tipos enumerados.

Listado 15.35: Registrando el modelo de datos de (Listado 15.34).

```
dbind_enum(type_t, ekCPU, "");
dbind_enum(type_t, ekGPU, "");
dbind_enum(type_t, ekHDD, "");
dbind_enum(type_t, ekSCD, "");
dbind(Product, type_t, type);
dbind(Product, String*, code);
dbind(Product, String*, description);
dbind(Product, Image*, image64);
dbind(Product, real32_t, price);
```

15.11.1. Sincronización con interfaces gráficas

Uno de los usos más extendidos del enlace de datos, es la posibilidad de sincronizar la interfaz gráfica con los objetos que forman el modelo de datos. Este paradigma es conocido como MVVM *(Model-View-ViewModel)* (Figura 15.35) y profundizaremos más en él *"GUI Data binding"* (Página 355).

15.11.2. Lectura y escritura de JSON

El análisis sintáctico de scripts JSON también se puede automatizar gracias a *dbind* (Figura 15.36). En *"JSON"* (Página 383) dispondrás de información detallada de como hacerlo.

```
enum type_t
{
  ekCPU,
  ekGPU,
  ekHDD,
  ekSCD
};

struct Product
{
  type_t type;
  String *code;
  String *description;
  Image *image64;
  real32_t price;
};
```

type_t

name	val
ekCPU	0
ekGPU	1
ekHDD	2
ekSCD	3

Product

offset	name	type
0	type	type_t
4	code	String*
8	description	String*
12	image64	Image*
16	price	real32_t

Figura 15.34: Tablas internas creadas por *dbind* al registrar el modelo de datos.

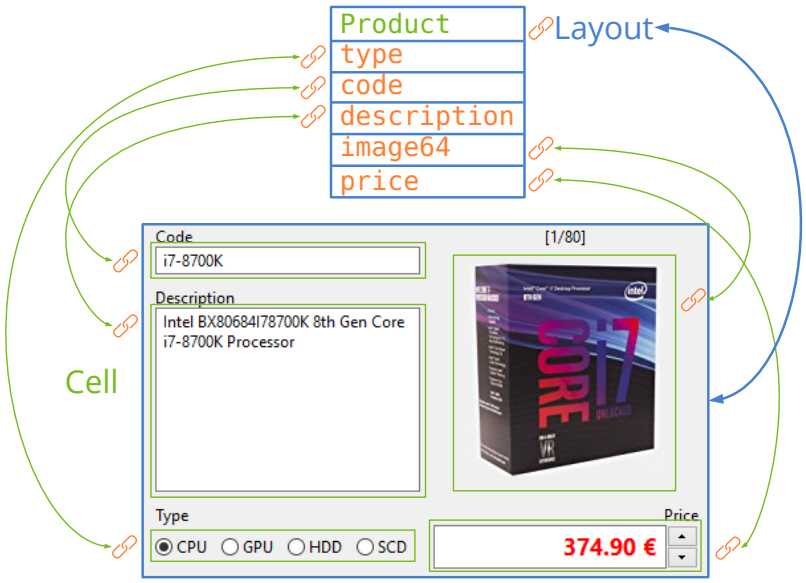


Figura 15.35: Sincronización automática de datos e interfaz.


```
{
    "code":0,
    "size":80,
    "data":[
    {"id":0,
    "code":"i7-8700K",
    "description":"Intel BX80684I78700K 8th Gen Core i7-8700K Processor",
    type":0,
```

```
    "price":374.889999999999986357579473406076431274414062 5,
    "image":"cpu_00.jpg",
    "image64":"\/9j\/4AAQSkZJRgABAQ....
    },
    ...
}
```

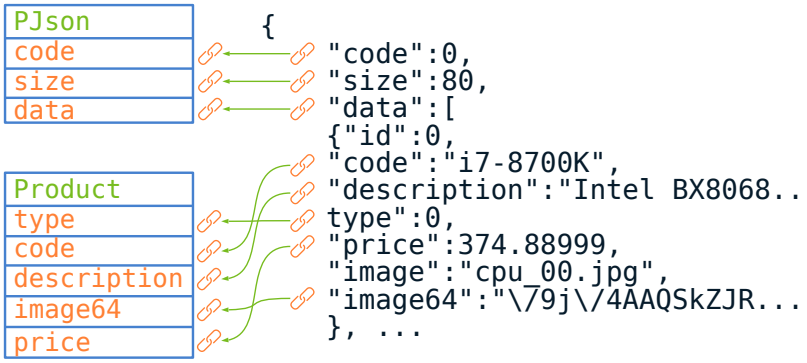


Figura 15.36: Data Binding en el análisis de scripts JSON.

15.11.3. Serialización con DBind

Como ya vimos en *"SerializaciónSerialización"* (Página 218) y *"Unificar la serializaciónUnificar la serialización"* (Página 208) necesitamos definir funciones de lectura y escritura de objetos para enviarlos o recibirlos a través de streams. Afortunadamente, *dbind* conoce la composición detallada de cada objeto registrado, por lo que puede acceder a la E/S sin que sea necesario programar explícitamente dichas funciones (Listado 15.36) (Figura 15.37).

Listado 15.36: Serialización de objetos con *dbind*.

```
ArrPt(Product) *products = dbind_read(stream, ArrPt(Product));
...
dbind_write(stream, products, ArrPt(Product));
```

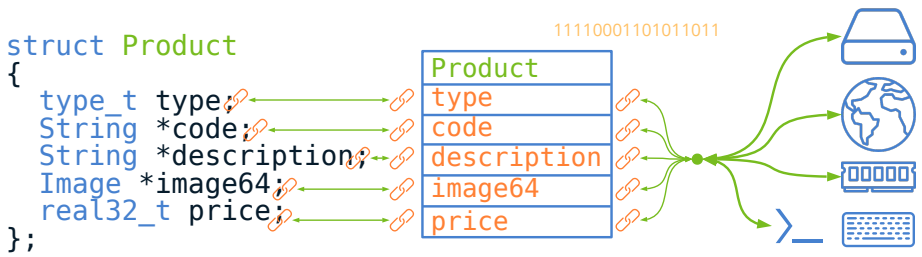


Figura 15.37: Lectura/Escritura de objetos mediante *dbind*.

15.11.4. Constructor por defecto

Gracias a *dbind* también podemos crear objetos inicializados con valores por defecto sin necesidad de crear constructores específicos (Listado 15.37). También se pueden destruir garantizando la correcta liberación recursiva de la memoria de todos sus campos.

- Utiliza `dbind_create` para crear un objeto *"ConstructoresConstructores"* (Página 216).
- Utiliza `dbind_init` para inicializar un objeto.
- Utiliza `dbind_destroy` para destruir un objetos *"DestructoresDestructores"* (Página 219).

Listado 15.37: Construcción y destrucción sin métodos adicionales.

```
ArrSt(Product) *array = dbind_create(ArrSt(Product));
Product *pr1 = dbind_create(Product);
Product *pr2 = arrst_new(array, Product);
dbind_init(pr2, Product);

// Use objects
...

dbind_destroy(&pr1, Product);
dbind_destroy(&array, ArrSt(Product));
```

Los valores por defecto al inicializar los campos del objeto son `0` para números, `FALSE` para booleanos, `""` para Strings y contenedores vacíos en el caso de arrays o sets. Si el objeto contiene sub-objetos anidados, estos también serán creados/inicializados de forma recursiva. Estos valores por defecto se pueden cambiar si fuera necesario (Listado 15.38).

- Utiliza `dbind_default` para establecer el valor por defecto.

Listado 15.38: Cambiando los valores por defecto.

```
dbind_default(Product, type_t, type, ekHDD);
dbind_default(Product, String*, code, "Empty-code");
dbind_default(Product, real32_t, price, 5.f);
dbind_default(Product, Image*, image64, gui_image(NOIMAGE_PNG));
```

15.11.5. Rangos numéricos

Es posible configurar los campos numéricos `uint32_t`, `int8_t`, `real64_t` para acotar los valores aceptados a un determinado rango (Listado 15.39). dbind se encargará de validar los datos cada vez que lea valores de cualquier origen de datos (GUI, JSON, Streams, etc).

- Utiliza `dbind_range` para establecer un máximo y mínimo en valores numéricos.

- Utiliza `dbind_precision` para establecer la precisión numérica. Por ejemplo `0.01` en valores monetarios.
- Utiliza `dbind_increment` para establecer el valor de incrementos discretos.
- Utiliza `dbind_suffix` para establecer un sufijo que será añadido al convertir los números a texto.

Listado 15.39: Rango y precisión del valor `price`.

```
dbind_default(Product, real32_t, price, 10f);
dbind_range(Product, real32_t, price, .50f, 10000f);
dbind_precision(Product, real32_t, price, .01f);
dbind_increment(Product, real32_t, price, 5.f);
dbind_suffix(Product, real32_t, price, "€");
```

15.12. Eventos

Un evento es una acción que ocurre durante la ejecución del programa, normalmente de forma asíncrona o impredecible y sobre la que un determinado objeto debe ser notificado. En aplicaciones con interfaz gráfica multitud de eventos están ocurriendo constantemente cuando el usuario interactúa con los diferentes controles. No obstante, también pueden ocurrir en aplicaciones de consola, por ejemplo, al terminar de escribir un archivo en disco o al descargar una página de Internet. En un sistema de eventos intervienen dos actores: El emisor *(sender)*, que tiene constancia cuando ocurre la acción y el receptor *(receiver)* al que se notifica que dicha acción ha ocurrido. Para conectar ambos extremos debemos realizar estos sencillos pasos (Listado 15.40) (Figura 15.38):

- Crear un `listener` indicando el objeto receptor y la función *callback* a la que debe llamar el emisor.
- Se asigna dicho *listener* al emisor mediante el método apropiado. Por ejemplo, el tipo `Button` provee el método `button_OnClick` para notificar de una pulsación.
- Cuando se produce el evento, el emisor llama a la función *callback*, indicando el objeto receptor (parámetro de `listener`) e información detallada sobre el evento recogida en el objeto `Event`.

Listado 15.40: Función *callback* y evento de pulsación de un botón.

```
static void OnClick(AppCtrl *ctrl, Event *event)
{
    // TODO: Response to click
}

...

void CreateButton(AppCtrl *ctrl)
```

```
{
    Button *button = button_push();
    button_text(button, "Ok");
    button_OnClick(button, listener(ctrl, OnClick, AppCtrl));
}
```

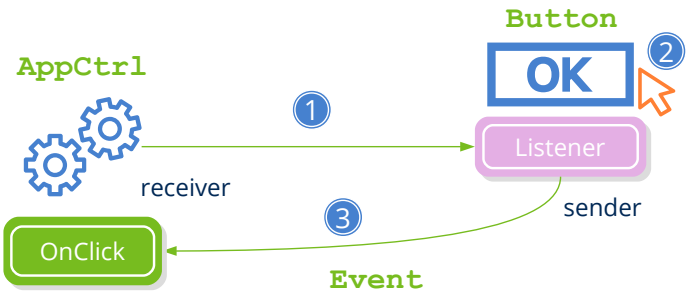


Figura 15.38: Evento `OnClick`.

Los eventos se utilizan de forma masiva en aplicaciones con interfaz gráfica, pero también pueden ser útiles en aplicaciones por línea de comandos. Ver, por ejemplo `hfile_dir_loop` en "Operaciones con archivos" (Página 236).

15.13. Búfer de teclado

El sistema operativo genera eventos relacionados con la pulsación o liberación de las teclas `view_OnDown` `view_OnUp`. En ciertas ocasiones necesitamos saber el estado de una tecla sin tener por ello que estar pendientes de los eventos que generan. `KeyBuf` nos ofrece un mecanismo muy sencillo de consulta utilizando tan solo el valor de la tecla `vkey_t`.

- Utiliza `keybuf_create` para crear el búfer.
- Utiliza `view_keybuf` para asignar el búfer a cualquier vista genérica, que será la encargada de capturar eventos y actualizarlo. El estado podrá ser consultado desde cualquier función del programa en cualquier momento.

15.14. Operaciones con archivos

Si bien en *"Archivos y directorios"* (Página 179) ya vimos como acceder al sistema de ficheros a bajo nivel, en ocasiones se hacen necesarias ciertas operaciones de alto nivel sobre los datos en disco. El mero hecho de borrar completamente un directorio lleva asociadas muchas operaciones individuales de bajo nivel. La librería *Core*, por medio de `<hfile.h>` provee de ciertas utilidades que nos pueden simplificar la vida en determinados momentos.

- Utiliza `hfile_dir_create` para crear un directorio, creando también sus predecesores si no existieran.

- Utiliza `hfile_dir_destroy` para eliminar un directorio y todo su contenido de forma recursiva.
- Utiliza `hfile_dir_sync` para sincronizar el contenido de dos directorios. Algo parecido al `rsync` de Unix.
- Utiliza `hfile_dir_loop` para recorrer un directorio en profundidad (Listado 15.41).
- Utiliza `hfile_buffer` para cargar en memoria el contenido de un archivo.

Listado 15.41: Uso de **hfile_dir_loop** para recorrer un directorio de tres niveles.

```
typedef struct _query_t Query;

static void i_OnEntry(Query *query, Event *e)
{
    const EvFileDir *p = event_params(e, EvFileDir);

    // First level (year)
    if (p->depth == 0)
    {
        // The entry is a directory
        if (event_type(e) == ekEENTRY)
        {
            bool_t *enter = event_result(e, bool_t);
            int16_t year = str_to_i16(p->filename, 10, NULL);

            // The loop enter in this subdir (depth 1)
            if (i_process_year(query, year) == TRUE)
                *enter = TRUE;
            else
                *enter = FALSE;
        }
    }
    // Second level (month)
    else if (p->depth == 1)
    {
        // The entry is a directory
        if (event_type(e) == ekEENTRY)
        {
            bool_t *enter = event_result(e, bool_t);
            uint8_t month = str_to_u8(p->filename, 10, NULL);

            // The loop enter in this subdir (depth 2)
            if (i_process_month(query, month) == TRUE)
                *enter = TRUE;
            else
                *enter = FALSE;
        }
    }
    // Third level (files)
    else if (p->depth == 2)
```

```
    {
        // The entry is a file
        if (event_type(e) == ekEFILE)
            i_process_file(query, p->pathname);
    }
}

/*---------------------------------------------------------------------------*/

Query query = i_init_query(&query);

hfile_dir_loop("main_path", listener(&query, i_OnEntry, Query), TRUE, FALSE,
    ↪ NULL);
```

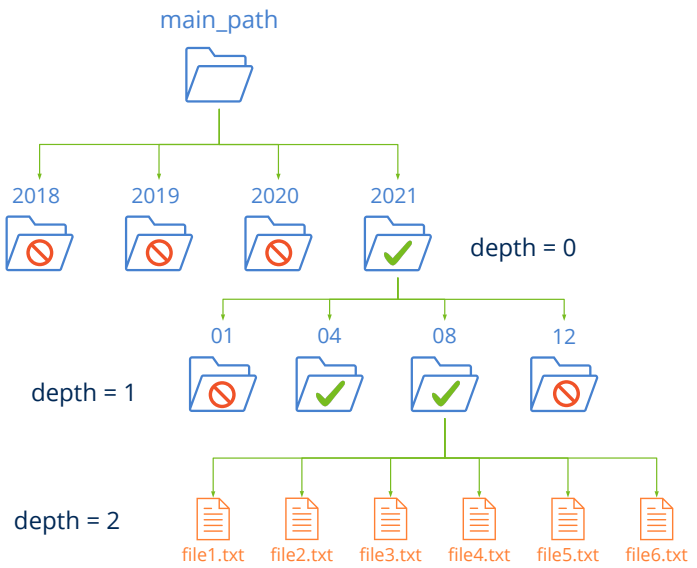


Figura 15.39: Representación del directorio de (Listado 15.41).

15.15. Paquetes de recursos

15.16. Fechas

Se incluyen dentro de *core* una serie de funciones para trabajar con fechas.

- Utiliza `date_system` para obtener la fecha del sistema.
- Utiliza `date_add_seconds` para incrementar una fecha determinada.
- Utiliza `date_cmp` para comparar dos fechas.

15.17. Relojes

Sencillos objetos que nos permiten, de forma cómoda, medir el lapso de tiempo acaecido entre dos instantes. También son útiles para lanzar eventos a intervalos regulares de tiempo (Listado 15.42).

Listado 15.42: Animación a 25fps.

```
Clock *clock = clock_create(.04);
for (;;)
{
    ...
    if (clock_frame(clock) == TRUE)
        listener_event(transition, ekGUI_EVENT_ANIMATION, NULL, params, NULL,
            ↪ void, EvTransition, void);
    ...
}
clock_destroy(&clock);
```

Capítulo 16

Librería Geom2D

16.1. Geom2D

Estamos ante una librería de cálculo geométrico en dos dimensiones. Geom2D permite trabajar con primitivas en el plano real: Puntos, vectores, transformaciones, curvas y

superficies. Ofrece tan solo funcionalidad matemática, es decir, no define ningún tipo de representación ni operación de dibujado. Depende únicamente de la librería *"Core"* (Página 191) (Figura 16.1), por lo que puede utilizarse tanto en aplicaciones de escritorio como en utilidades por línea de comandos. Todos los tipos y funciones se definen en simple `(float)` y `double` precisión, además de poder hacer uso de las *"Plantillas matemáticas-Plantillas matemáticas"* (Página 53) en C++.

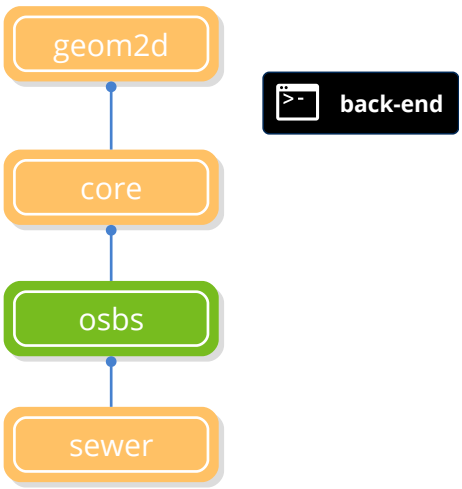


Figura 16.1: Dependencias de *geom2d*. Ver *"NAppGUI API"* (Página 147).

Todos los elementos geométricos se basan en coordenadas **(x, y)** en el plano. Geom2D no asume como serán interpretadas dichas coordenadas. Eso dependerá del sistema de referencia definido por la aplicación. Los más utilizados son el cartesiano y el de pantalla (Figura 16.2), aunque podrían utilizarse otros sistemas llegado el caso (Figura 16.3).

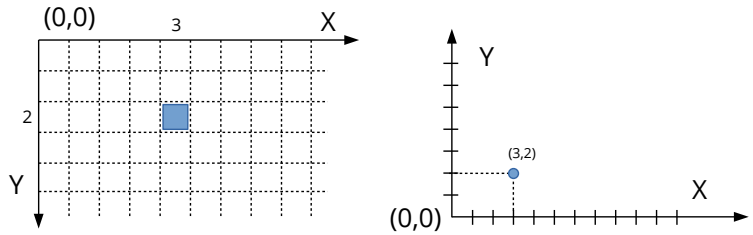


Figura 16.2: Interpretación de la coordenada (3,2) en monitores (izquierda) y en el plano cartesiano (derecha).

16.2. Vectores 2D

El vector (`V2Df`, `V2Dd`) es el elemento geométrico más elemental. Representa un punto, una dirección o un desplazamiento mediante sus dos componentes **x** e **y** (Figura 16.4).

El Álbegra Vectorial define una serie de operaciones básicas: Suma, negación, multipli-

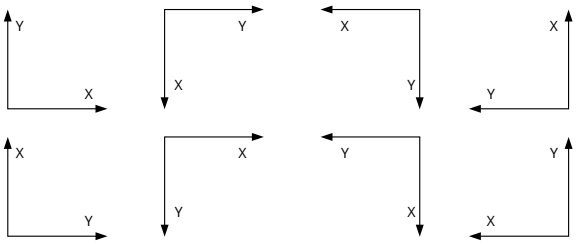


Figura 16.3: Diferentes sistemas de coordenadas en 2D.

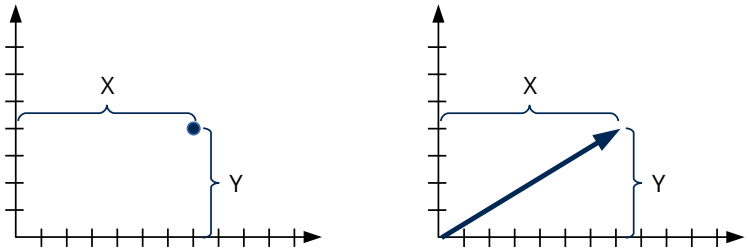


Figura 16.4: Un vector 2D representa una posición o un desplazamiento en el plano.

cación por un escalar, módulo y normalización (Fórmula 16.1). La representación visual de dichas operaciones la tenemos en (Figura 16.5).

$$
\begin{aligned}
\vec{v} &= \vec{a} + \vec{b} \\
&= (a.x + b.x, a.y + b.y) \\
\vec{v} &= p2 - p1 \\
&= (p2.x - p1.x, p2.y - p1.y) \\
-\vec{a} &= (-a.x, -a.y) \\
\vec{v} &= s \cdot \vec{a} \\
&= (s \cdot a.x, s \cdot a.y) \\
|\vec{a}| &= \sqrt{a.x^2 + a.y^2} \\
\hat{a} &= \left(\frac{a.x}{|\vec{a}|}, \frac{a.y}{|\vec{a}|} \right)
\end{aligned}
$$

Fórmula 16.1: Álbegra vectorial elemental.

- Utiliza `v2d_addf` para sumar dos vectores.
- Utiliza `v2d_subf` para restar dos vectores.
- Utiliza `v2d_mulf` para multiplicar por un escalar.

- Utiliza `v2d_lengthf` para calcular el módulo de un vector.
- Utiliza `v2d_normf` para normalizar un vector.

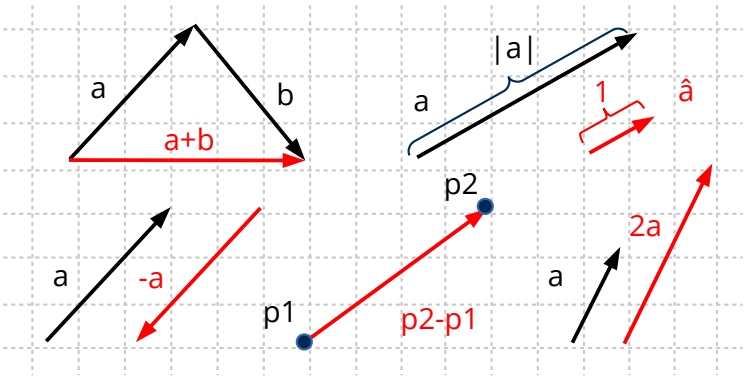


Figura 16.5: Interpretación geométrica de las operaciones básicas con vectores.

16.2.1. Ángulos CW y CCW

El ángulo de rotación de un vector se expresará siempre en **radianes** y el **sentido positivo** corresponde al giro desde el **eje X hacia el eje Y**. Normalmente se asocia como positiva la dirección antihoraria *counter clockwise (CCW)* y negativa la dirección horaria *clockwise (CW)*. Esto es cierto en coordenadas cartesianas pero no en otro tipo de sistemas de referencia, como imágenes o monitores (Figura 16.6). Debemos tenerlo presente para evitar confusiones, algo que sucede con relativa frecuencia. El mismo criterio se aplica al calcular el vector perpendicular, diferenciando entre positivo y negativo.

- Utiliza `v2d_anglef` para obtener el ángulo entre dos vectores.
- Utiliza `v2d_rotatef` para aplicar un giro a un vector.
- Utiliza `v2d_perp_posf` para calcular el vector perpendicular positivo.

*Para evitar confusiones recuerda que el sentido positivo es el que gira desde el eje X al eje Y. Será **dirección antihoraria** en coordenadas cartesianas y **dirección horaria** en coordenadas de pantalla.*

16.2.2. Proyección de vectores

Otra operación utilizada con bastante frecuencia en geometría es la proyección de puntos sobre un vector. Intuitivamente, podemos verlo como el punto sobre el vector más cercano al punto original y que siempre estará sobre la línea perpendicular. La calcularemos con

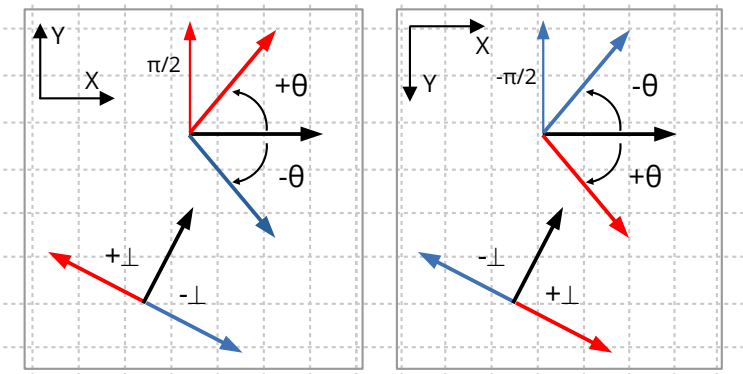


Figura 16.6: Rotación de un vector en los sistemas cartesiano y pantalla.

el producto escalar (Fórmula 16.2) y su valor (escalar) será la distancia desde el origen a la proyección en la dirección del vector (Figura 16.7).

- Utiliza `v2d_dotf` para calcular el producto escalar de dos vectores.

$$
\begin{aligned}
proj_{\vec{v}}(p) &= \frac{v.x \cdot p.x + v.y \cdot p.y}{|\vec{v}|} \\
proj_{\vec{4,3}}(1,2) &= \frac{4 \cdot 1 + 3 \cdot 2}{5} = 2 \\
proj_{\vec{4,3}}(2,-2) &= 0{,}4 \\
proj_{\vec{4,3}}(5,1) &= 4{,}6 \\
proj_{\vec{4,3}}(-3,1) &= -1{,}8
\end{aligned}
$$

Fórmula 16.2: Proyección de varios puntos en un vector.

Si lo que nos interesa es la posición relativa entre diferentes proyecciones, podemos evitar la división por el módulo del vector, lo que es más eficiente computacionalmente al no calcular raíces cuadradas.

16.3. Tamaños 2D

La estructura `S2Df`, `S2Dd` guarda información acerca de una medida o tamaño en dos dimensiones mediante sus campos `width` y `height` (Figura 16.8).

- Utiliza `s2df` para componer una medida a través de sus campos elementales.

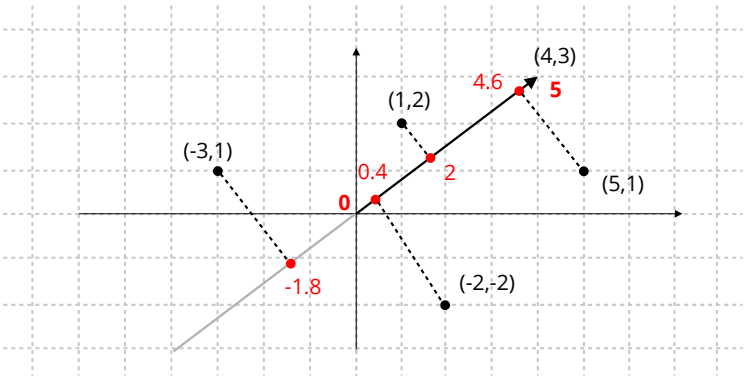


Figura 16.7: Interpretación geométrica de las proyecciones.

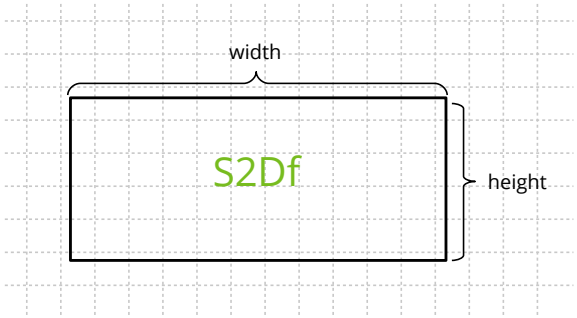


Figura 16.8: Size2D.

16.4. Rectángulos 2D

Un rectángulo (o *frame*) (`R2Df`, `R2Dd`) (Figura 16.9) sirve para ubicar elementos en interfaces de usuario u otros sistemas 2D mediante un punto de origen `V2Df` y un tamaño `S2Df`. También pueden utilizarse en operaciones de recorte (*clipping*), a la hora de optimizar tareas de dibujo.

- Utiliza `r2d_collidef` para determinar si dos rectángulos colisionan.
- Utiliza `r2d_clipf` para determinar si un rectángulo es visible dentro de un área.
- Utiliza `r2d_joinf` para unir dos rectángulos.

16.5. Transformaciones 2D

Las transformaciones afines son operaciones matemáticas que permiten realizar cambios de coordenadas entre diferentes sistemas de referencia. Por ejemplo, en (Figura 16.10) **(a)** construimos un polígono expresando las coordenadas de sus vértices en un sistema cartesiano: [(4,1), (2,5), (-3,5), (-4,2), (0,-3)]. Ahora imaginemos que queremos dibujar varias instancias del mismo en un plano, cada una de ellas con diferente posición, orientación

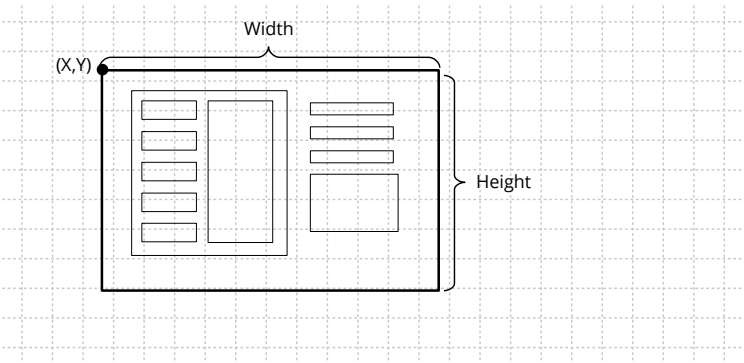


Figura 16.9: Posicionamiento de elementos GUI mediante rectángulos.

y tamaño (Figura 16.10) **(b)**. Necesitaríamos calcular las coordenadas de los puntos del polígono original en las nuevas ubicaciones, con el fin de trazar correctamente las líneas que delimitan cada copia.

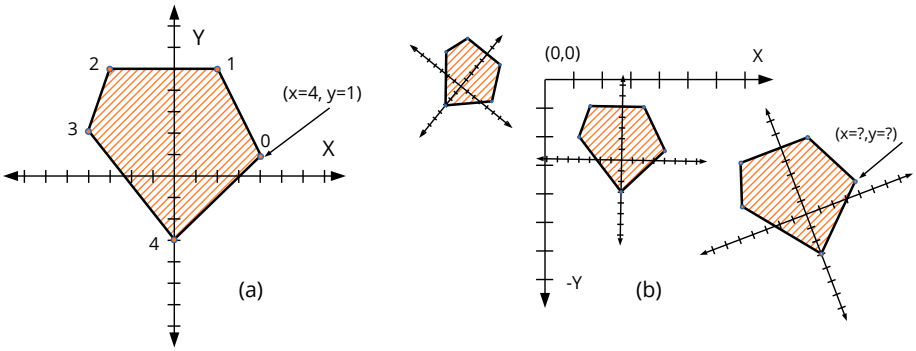


Figura 16.10: Modelo geométrico (a) Expresado en un sistema cartesiano. (b) Después de aplicar transformaciones.

El Álgebra Vectorial nos brinda una poderosa herramienta con la que se puede expresar la relación entre dos sistemas utilizando seis número reales (Figura 16.11). Los primeros cuatro valores corresponden a una matriz 2x2 con las coordenadas de los vectores X=[1,0] e Y=[0,1] del nuevo sistema de referencia. Esta matriz integra una posible rotación y escalado de los ejes. Los últimos dos valores indican un desplazamiento del origen. En (Fórmula 16.3) tenemos el desarrollo matemático para transformar el punto [4,1] a una nueva base girada 25° con respecto al origen y desplazada 11 unidades en el eje X y -5 en el eje Y. Aplicando la misma operación a cada punto, transformaríamos el objeto completo.

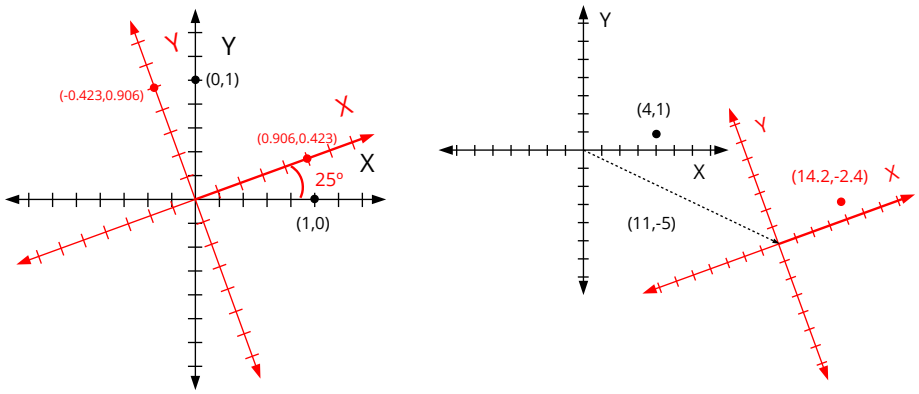


Figura 16.11: Cambio de base. Relación de un punto en dos sistemas de referencia diferentes.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} i.x & j.x \\ i.y & j.y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p.x \\ p.y \end{bmatrix} \\
&= \begin{bmatrix} 0{,}906 & -0{,}423 \\ 0{,}423 & 0{,}906 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \end{bmatrix} + \begin{bmatrix} 11 \\ -5 \end{bmatrix} \\
&= \begin{bmatrix} 3{,}2 \\ 2{,}6 \end{bmatrix} + \begin{bmatrix} 11 \\ -5 \end{bmatrix} \\
&= \begin{bmatrix} 14{,}2 \\ -2{,}4 \end{bmatrix}
\end{aligned}
$$

Fórmula 16.3: Transformación del punto [4,1].

16.5.1. Clasificación de transformaciones

En principio, cualquier combinación de valores [i.x, i.y, j.x, j.y, p.x, p.y] proporcionaría una transformación válida, aunque si no los elegimos con cierto criterio obtendremos aberraciones poco útiles en la práctica. Las transformaciones más utilizadas en aplicaciones gráficas y de ingeniería son (Figura 16.12) (Figura 16.13) (Fórmula 16.4):

- Traslación **(a)**: Desplaza el origen del objeto a otro punto.
- Rotación **(b)**: Gira el objeto sobre el origen de su sistema local.
- Escalado **(c)**: Cambia el tamaño. Si $sx < 1$, reduce. $sx > 1$, aumenta. $sx = 1$, no varía. En escalados no uniformes, sx y sy tienen valores diferentes, lo que producirá una deformación en la relación de aspecto.
- Identidad **(d)**: Es la transformación nula. Al aplicarla, los vectores permanecen inal-

terados.

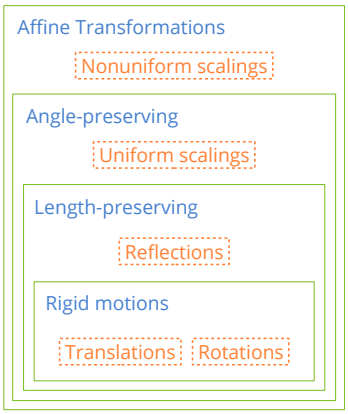


Figura 16.12: Clasificación de transformaciones afines.

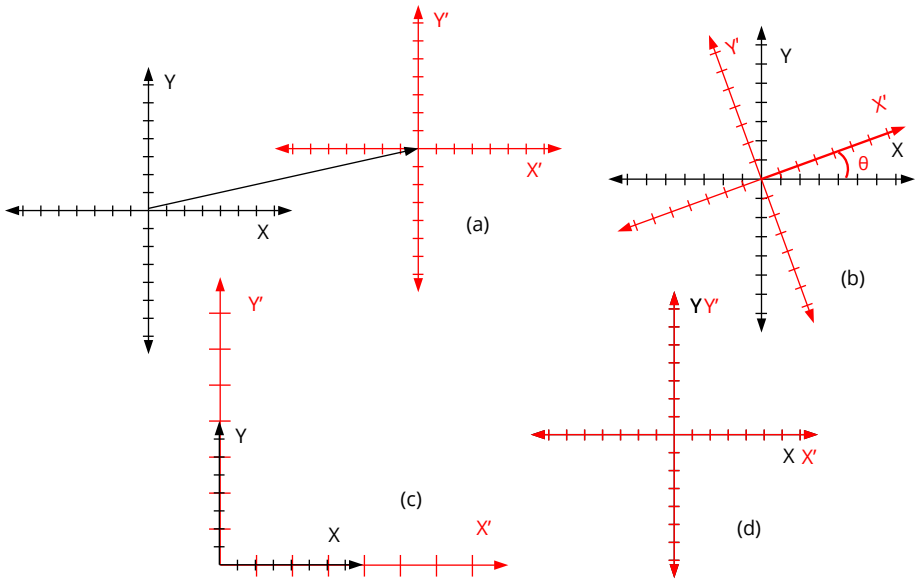


Figura 16.13: Representación geométrica de transformaciones elementales. (a) Traslación, (b) Rotación, (c) Escalado, (d) Identidad.

16.5.2. Composición de transformaciones

Es posible componer o acumular transformaciones mediante la multiplicación de matrices (Fórmula 16.5). Lo habitual en modelos 2d será obtener la ubicación final de un objeto a partir de las transformaciones elementales traslación, rotación y escalado. La acumulación también es útil para posicionar elementos en estructuras jerárquicas, donde la ubicación de cada objeto depende directamente de la de su nodo superior (padre).

- Utiliza `t2d_movef` para acumular un desplazamiento a una transformación existente.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p.x \\ p.y \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

Fórmula 16.4: Traslación, Rotación, Escalado e Identidad.

- Utiliza `t2d_rotatef` para acumular una rotación.
- Utiliza `t2d_scalef` para acumular un escalado.
- Utiliza `t2d_multf` para acumular una transformación a otra ya existente.
- Utiliza `t2d_vmultf` para aplicar una transformación a un vector.
- Utiliza `t2d_vmultnf` para aplicar una transformación a varios vectores.
- Utiliza `kT2D_IDENTf` para hacer referencia a la transformación identidad.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} i_d.x & j_d.x \\ i_d.y & i_d.y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p_d.x \\ p_d.y \end{bmatrix} \\
i_d.x &= i_1.x \cdot i_2.x + j_1.x \cdot i_2.y \\
i_d.y &= i_1.y \cdot i_2.x + j_1.y \cdot i_2.y \\
j_d.x &= i_1.x \cdot j_2.x + j_1.x \cdot j_2.y \\
j_d.y &= i_1.y \cdot j_2.x + j_1.y \cdot j_2.y \\
p_d.x &= i_1.x \cdot p_2.x + j_1.x \cdot p_2.y + p_1.x \\
p_d.y &= i_1.y \cdot p_2.x + j_1.y \cdot p_2.y + p_1.y
\end{aligned}
$$

Fórmula 16.5: Composición de dos transformaciones arbitrarias.

La multiplicación matricial no es conmutativa, si no que el orden en el que se apliquen las operaciones afectará al resultado final. Por ejemplo, en (Figura 16.14) **(a)**, se ha trasladado el origen y después aplicado una rotación. En (Figura 16.14) **(b)** se ha hecho al contrario,

primero rotar y luego trasladar.

Listado 16.1: Acumulación de transformaciones.

```
// (a) First move, then rotate
T2Df t2d;
t2d_movef(&t2d, kT2D_IDENTf, 11, 0);
t2d_rotatef(&t2d, &t2d, kBMATH_PIf / 4);

// (b) First rotate, then move
T2Df t2d;
t2d_rotatef(&t2d, kT2D_IDENTf, kBMATH_PIf / 4);
t2d_movef(&t2d, &t2d, 11, 0);
```

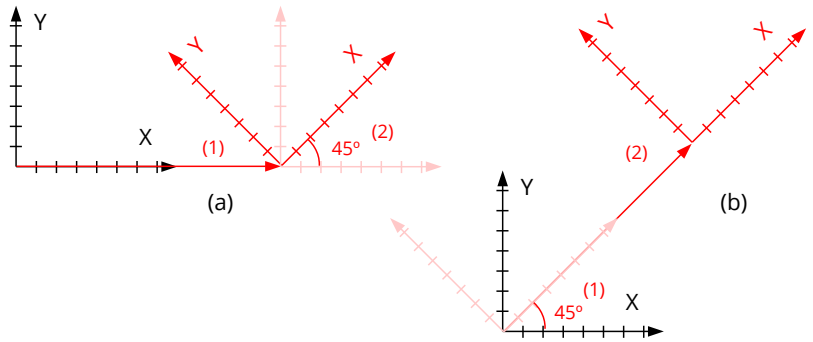


Figura 16.14: Efecto del orden de aplicación de transformaciones.

16.5.3. Descomposición e inversa

Cualquier cadena de traslaciones, rotaciones y escalados define un sistema de referencia afín que puede ser expresado en función de un único desplazamiento, rotación y escalado (Figura 16.15). Podemos "deshacer" dicha transformación y volver al origen mediante la transformación inversa (Listado 16.2).

- Utiliza `t2d_decomposef` para obtener las componentes de una transformación.
- Utiliza `t2d_inversef` para obtener la transformación inversa.

Listado 16.2: Componentes de un sistema de referencia e inversa.

```
T2Df t2d, inv, inv2;
V2Df pos, sc;
real32_t a;

// Transform sequence
t2d_rotatef(&t2d, kT2D_IDENTf, kBMATH_PIf / 4);
t2d_movef(&t2d, &t2d, 11, 0);
t2d_movef(&t2d, &t2d, 10, - 10);
t2d_rotatef(&t2d, &t2d, - kBMATH_PIf / 2);
```

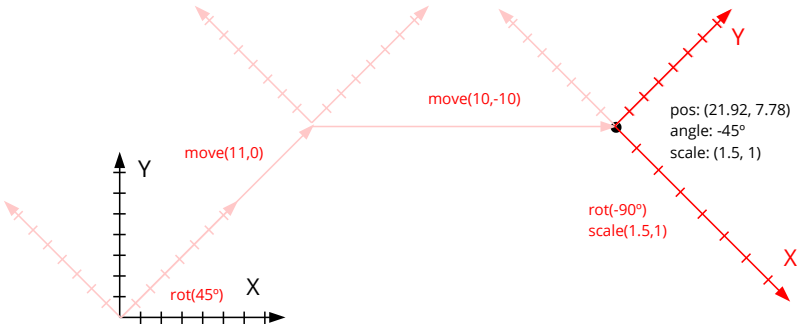


Figura 16.15: Cadena de transformaciones y sistema final.

```
t2d_scalef(&t2d, &t2d, 1.5f, 1);

// Transform components
t2d_decomposef(&t2d, &pos, &a, &sc);

// Transform inverse
t2d_inversef(&inv, &t2d);

// Inverse from components
t2d_scalef(&inv2, kT2D_IDENTf, 1/sc.x, 1/sc.y);
t2d_rotatef(&inv2, &inv2, -a);
t2d_movef(&inv2, &inv2, -pos.x, -pos.y);

// inv == inv2 ('inv' more numerical accurate)
```

16.6. Segmentos 2D

Los segmentos son fragmentos de recta comprendidos entre dos puntos **p0** y **p1** (Figura 16.16). Son las primitivas geométricas más simples, después de los vectores. Definimos el parámetro **t** como la posición normalizada dentro del segmento. Valores entre `0` y `1` corresponderán a puntos internos del segmento, con los límites `t=0` (`p0`) y `t=1` (`p1`). Fuera de este rango tendremos los puntos externos al segmento, pero dentro de la recta que lo contiene. Por ejemplo `t=2` sería el punto pasado `p1` situado a una distancia igual a la longitud del segmento.

- Utiliza `seg2d_lengthf` para obtener la longitud del segmento.
- Utiliza `seg2d_close_paramf` para obtener el valor del parámetro más cercano a un punto determinado.
- Utiliza `seg2d_evalf` para obtener el punto 2D a partir del parámetro.
- Utiliza `seg2d_sqdistf` para obtener la distancia (al cuadrado) entre dos segmentos.

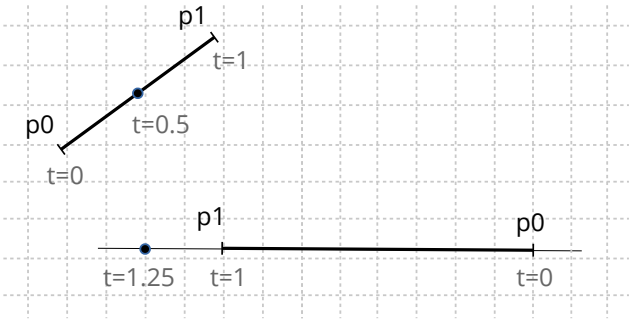


Figura 16.16: Segmentos en el plano.

16.7. Círculos 2D

Los círculos nos permiten agrupar un conjunto de puntos dentro de un mismo volumen contenedor. La detección de colisiones se realizará de manera óptima ya que es el test geométrico que menos operaciones requiere. Dado un conjunto de puntos, podemos calcular el círculo contenedor de varias formas (Figura 16.17) según la precisión y velocidad necesarias.

- Utiliza `cir2d_from_boxf` para obtener el círculo a partir de una caja 2D.
- Utiliza `cir2d_minimumf` para el obtener el círculo de radio mínimo a partir de un conjunto de puntos.
- Utiliza `cir2d_from_pointsf` para obtener el círculo a partir de la media del conjunto. Opción más equilibrada en cuanto a precisión/rendimiento.

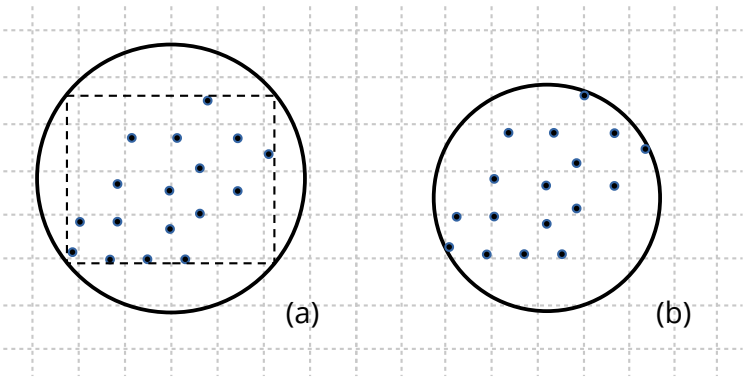


Figura 16.17: Círculo contenedor: A partir de BBox **(a)**. De radio mínimo **(b)**.

16.8. Cajas 2D

Los cajas 2D o *(Axis-Aligned Bounding boxes)* delimitan la zona del plano que ocupan diferentes elementos geométricos (Figura 16.18). Son útiles en la detección de colisiones u operaciones de recortado *(clipping)*, ya que impiden que se dibujen figuras no visibles, mejorando el rendimiento global. También son muy rápidas de computar a partir de un conjunto de puntos.

- Utiliza `box2d_from_pointsf` para crear una caja 2D a partir de un conjunto de puntos.
- Utiliza `box2d_addnf` para cambiar las dimensiones en función de nuevos puntos.
- Utiliza `box2d_segmentsf` para obtener los cuatro segmentos que delimitan la caja.

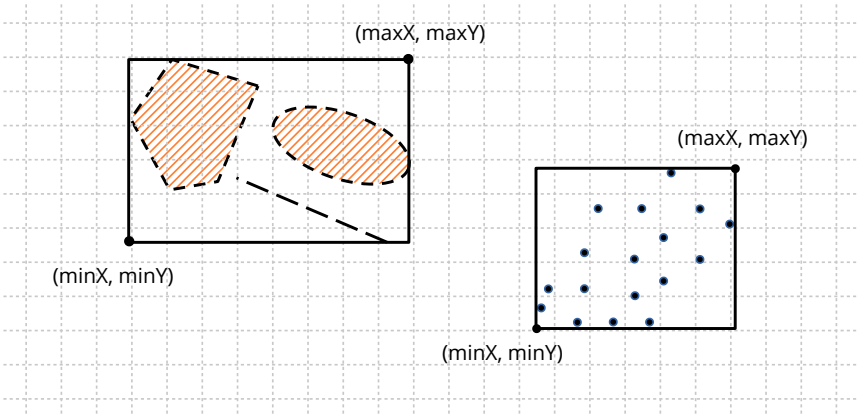


Figura 16.18: Cajas 2D como contenedor de otros objetos.

16.9. Cajas Orientadas 2D

Las cajas orientadas *(Oriented Bounding Box)* son cajas 2D que pueden rotan sobre su centro (Figura 16.19), por lo que ya no estarán alineadas con los ejes. Aquí la detección de colisiones se complica un poco con respecto a las cajas 2D alineadas, a cambio de proporcionar un mejor ajuste ante objetos alargados que puedan girar en el plano.

- Utiliza `obb2d_from_pointsf` para crear una caja orientada a partir de un conjunto de puntos.
- Utiliza `obb2d_from_linef` para crear una caja orientada a partir de un segmento.
- Utiliza `obb2d_transformf` para aplicar una transformación 2D a la caja.
- Utiliza `obb2d_boxf` para obtener la caja alineada que contiene a la caja orientada.

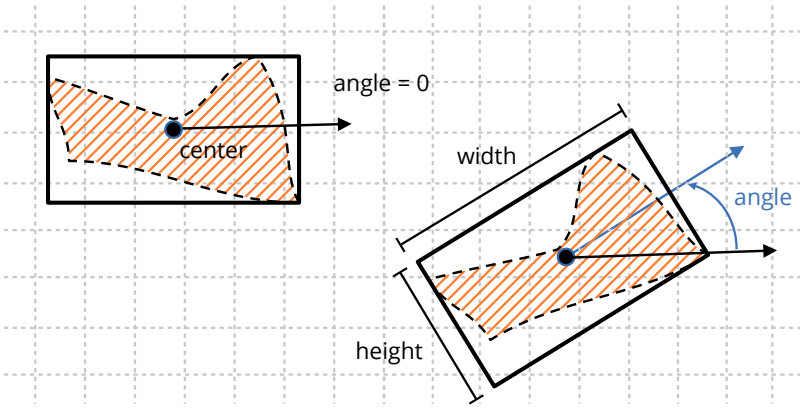


Figura 16.19: Cajas orientadas en 2D.

Podemos obtener parámetros relevantes de un conjunto arbitrario de puntos a partir de la matriz de covarianza (Fórmula 16.6), que geométricamente representa una elipse rotada en el plano y centrada en la media de la distribución (Figura 16.20). Este análisis permite a `obb2d_from_pointsf` calcular la caja 2D asociada a la distribución de una forma bastante aceptable, sin llegar a ser la solución óptima mucho más costosa en términos computacionales.

$$
\begin{aligned}
\Sigma &= \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \\
\sigma_{xx} &= \frac{1}{N}\left[\sum_{i=1}^{N} {x_i}^2\right] - \mu_x^2 \\
\sigma_{yy} &= \frac{1}{N}\left[\sum_{i=1}^{N} {y_i}^2\right] - \mu_y^2 \\
\sigma_{xy} &= \frac{1}{N}\left[\sum_{i=1}^{N} x_i y_i\right] - \mu_x \mu_y \\
\sigma_{yx} &= \sigma_{xy} \\
\mu_x &= \frac{1}{N}\sum_{i=1}^{N} x_i \\
\mu_y &= \frac{1}{N}\sum_{i=1}^{N} y_i
\end{aligned}
$$

Fórmula 16.6: Cálculo de la matriz de covarianza.

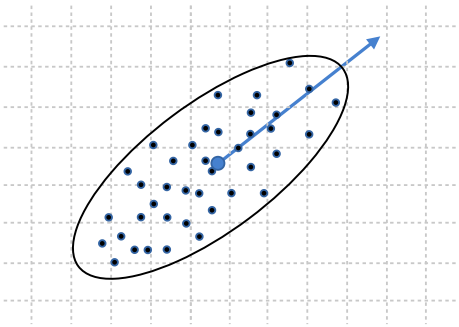


Figura 16.20: La matriz de covarianza representa una elipse rotada en el plano.

Utiliza cajas orientadas (`OBB2Df`) para distribuciones "alargadas" de puntos. En casos redondeados o cuadrados la caja alineada (`Box2Df`) puede proporcionar un volumen de menor área.

16.10. Triángulos 2D

Los triángulos son ampliamente utilizados en geometría computacional, sobre todo a la hora de realizar ciertos cálculos sobre polígonos o superficies. También son la base de la mayoría de APIs gráficas, por lo que en numerosas ocasiones necesitaremos aproximar objetos mediante triángulos. El **centroide** es el punto de equilibrio que se encuentra en la intersección de las medianas (Figura 16.21).

- Utiliza `tri2df` para componer un triángulo.
- Utiliza `tri2d_transformf` para aplicar una transformación.
- Utiliza `tri2d_centroidf` para obtener el centro de masas.
- Utiliza `tri2d_areaf` para calcular el área.

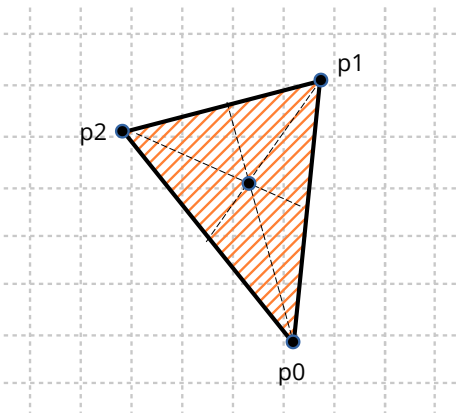


Figura 16.21: Triángulos 2D.

16.11. Polígonos 2D

Los polígonos son figuras ampliamente versátiles, ya que nos permiten definir regiones arbitrarias delimitadas por segmentos rectilíneos. Geom2D soporta los llamados **polígonos simples**, que son aquellos cuyos lados no pueden intersectar entre sí.

- Utiliza `pol2d_createf` para crear un polígono a partir del recorrido que forman sus vértices.
- Utiliza `pol2d_ccwf` para obtener el sentido de giro del recorrido. Ver *"Ángulos CW y CCWÁngulos CW y CCW"* (Página 244).
- Utiliza `pol2d_transformf` para aplicar una transformación al polígono.
- Utiliza `pol2d_areaf` para obtener el área.
- Utiliza `pol2d_boxf` para obtener los límites del polígono.

Podemos clasificar a los polígonos en tres grandes grupos (Figura 16.22):

- **Convexos:** Los más "deseados" desde el punto de vista de la simplicidad de cálculo. Son aquellos donde cualquier segmento que una dos puntos interiores, queda totalmente dentro del polígono.
- **Cóncavos:** O no convexos. Lo opuesto a lo anterior. Es aquel que tiene un ángulo interior de más de 180 grados.
- **Débiles (Weakly):** Es aquel que presenta agujeros por medio de segmentos "de corte" donde dos vértices son duplicados para permitir el acceso y regreso de cada agujero. Es una forma sencilla de vaciar el interior de regiones sin que sean necesarios múltiples ciclos. El cálculo de áreas y colisiones tendrán en cuenta estas cavidades.

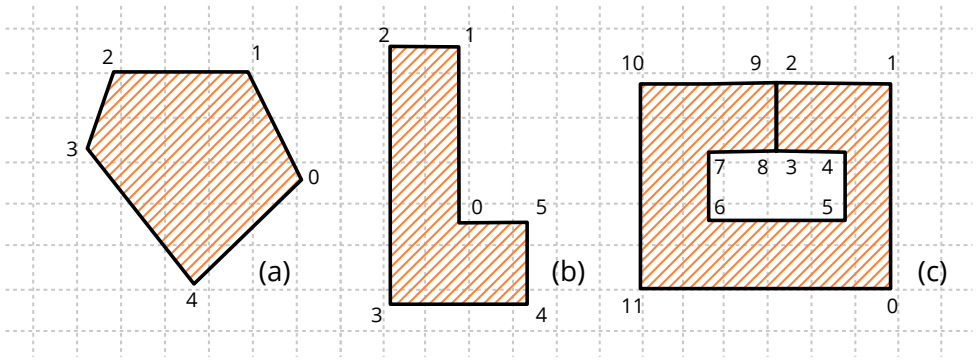


Figura 16.22: Polígonos 2D. (a) Convexo, (b) Cóncavo, (c) Débil. Todos ellos definidos en sentido antihorario *(counter-clockwise)*.

16.11.1. Centro del polígono

Es complicado definir un punto central en una figura tan irregular como puede llegar a ser un polígono. Normalmente interpretaremos como tal el centroide o **centro de masas** pero, en casos no cónvexos, este punto puede localizarse en el exterior del polígono. En tareas de etiquetado se hace necesario disponer de un punto representativo que esté dentro de la figura. Consideramos como **centro visual** a aquel punto dentro del polígono situado a una distancia máxima de cualquier borde (Figura 16.23). En polígonos convexos coincidirá con el centroide.

- Utiliza `pol2d_centroidf` para obtener el centroide del polígono.
- Utiliza `pol2d_visual_centerf` para obtener el centro visual del polígono. Implementa una adaptación del algoritmo **polylabel** del proyecto MapBox[1].

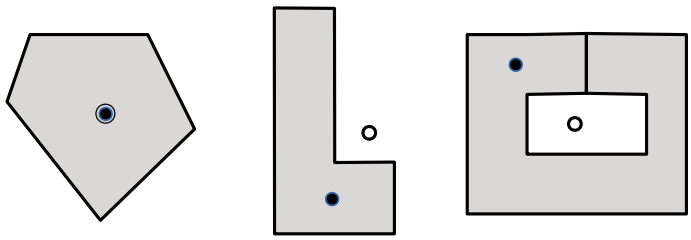


Figura 16.23: Punto "central" de un polígono. Línea: Centroide, Relleno: Centro visual o de etiquetado.

16.11.2. Descomposición de polígonos

Determinados cálculos o tareas de renderizado se pueden optimizar considerablemente si reducimos la complejidad de la geometría a tratar. Descomponer un polígono no es más que obtener una lista de polígonos más simples cuya unión equivale a la figura original (Figura 16.24). Como operación inversa, tendríamos el cálculo de la **envoltura convexa** *Convex Hull*, que es la obtención del polígono convexo que encierra a un conjunto de puntos arbitrarios (Figura 16.25).

- Utiliza `pol2d_trianglesf` para obtener una lista de los triángulos que forman el polígono.
- Utiliza `pol2d_convex_partitionf` para obtener una lista polígonos convexos equivalentes al polígono.
- Utiliza `pol2d_convex_hullf` para crear un polígono convexo que "envuelva" a un conjunto de puntos.

[1] `https://github.com/mapbox/polylabel`

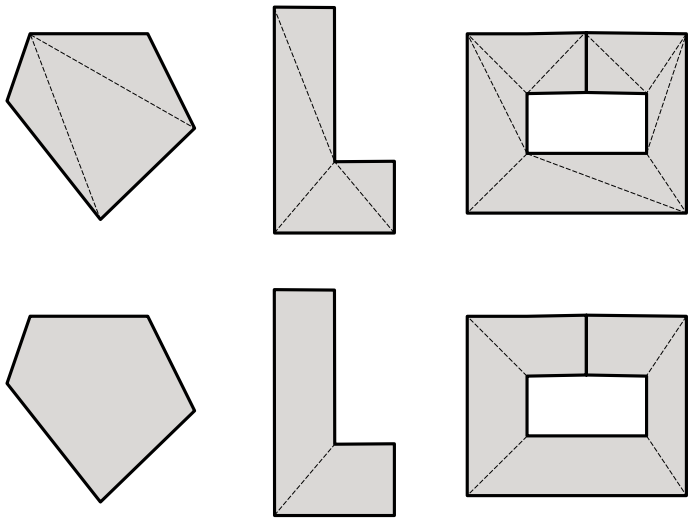


Figura 16.24: Descomposición de un polígono mediante triangulación o componentes convexas.

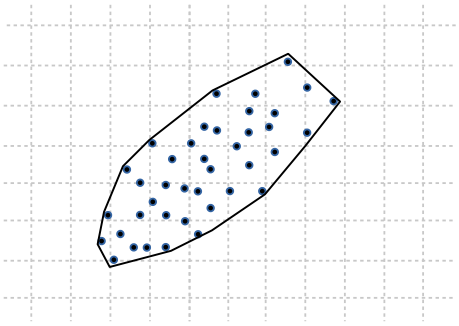


Figura 16.25: Envoltura convexa de un conjunto de puntos.

16.12. Colisiones 2D

La detección de colisiones se encarga de estudiar y desarrollar algoritmos que comprueben si dos objetos geométricos intersectan en algún punto. Como el caso general sería bastante complejo de implementar e ineficiente de evaluar, se definen una serie de **volúmenes de colisión** (Figura 16.26) que encerrarán los conjuntos originales y donde los tests se pueden simplificar notablemente. Al uso de estas formas más elementales normalmente se le conoce como *broad phase collision detection* (Figura 16.27), ya que persigue detectar la "no colisión" lo más rápido posible.

- Utiliza `col2d_poly_obbf` para detectar la colisión entre una caja orientada y un polígono.
- Utiliza `col2d_tri_trif` para detectar la colisión entre dos triángulos.
- Utiliza `col2d_circle_segmentf` para detectar la colisión entre un círculo y un segmento.

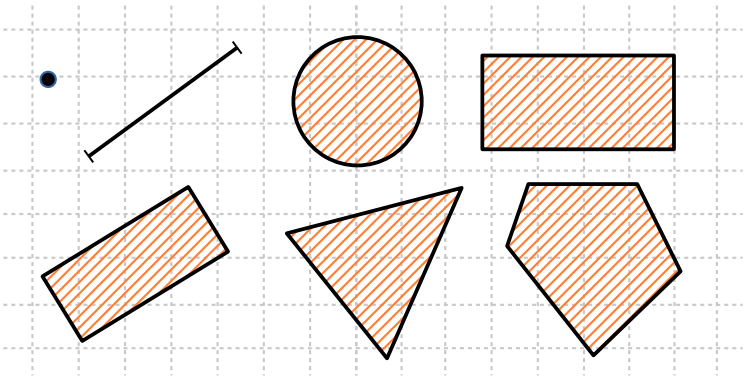


Figura 16.26: Volúmenes de colisión 2D: Punto, segmento, círculo, caja, caja orientada, triángulo y polígono.

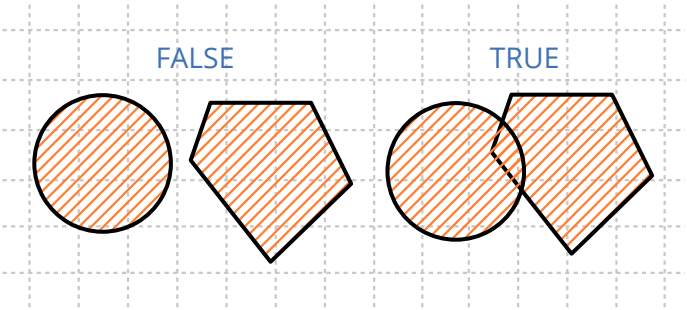


Figura 16.27: Detección de colisiones *broad phase*.

Col2D proporciona funciones para chequear cada par de los volúmenes de colisión presentados con anterioridad. Gran parte de estos métodos utilizan el **Teorema del Eje de Separación** *(Separating Axis Theorem)* (Figura 16.28). Este teorema indica, en esencia, que si es posible encontrar una línea donde las proyecciones de los vértices no intersectan, entonces las figuras no intersectan. En el caso concreto de polígonos convexos, tan solo es necesario evaluar **n líneas**, donde n es el número de lados del polígono.

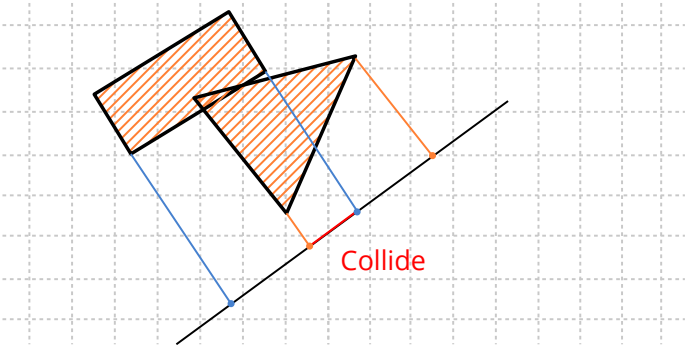


Figura 16.28: Teorema del eje de separación detectando una colisión.

Capítulo 17

Librería Draw2D

17.1. Draw2D

La librería *Draw2D* integra toda la funcionalidad necesaria para crear gráficos vectoriales en dos dimensiones. Depende directamente de *Geom2D* (Figura 17.1) y, como veremos más adelante, dibujar no implica disponer de interfaz gráfica en el programa. Es posible generar imágenes utilizando un búfer interno de memoria, sin necesidad de visualizar el resultado en una ventana.

- *"Contextos 2D"* (Página 263).
- *"Primitivas de dibujo"* (Página 271).
- *"Colores"* (Página 283) y *"Paletas"* (Página 285).
- *"Pixel Buffer"* (Página 286) e *"Imágenes"* (Página 290).
- *"Fuentes tipográficas"* (Página 295).

Esta librería conecta directamente con las tecnologías nativas de cada sistema operativo: **GDI+** en sistemas Windows, **Quartz2D** en macOS y **Cairo** en Linux. En esencia *draw2d* ofrece una interfaz común y ligera con el fin de que el código sea portable, delegando el trabajo final en cada una de ellas. Con esto garantizamos tres cosas:

- Eficiencia: Estas APIs han sido probadas durante años y son mantenidas por los fabricantes de los sistemas.
- Presencia: Están integradas de "serie" en todos los ordenadores, por lo que no se requiere instalar software adicional.
- Rendimiento: Los programas son más pequeños ya que no requieren enlazar con rutinas especiales para el manejo de gráficos, tipografías o imágenes.

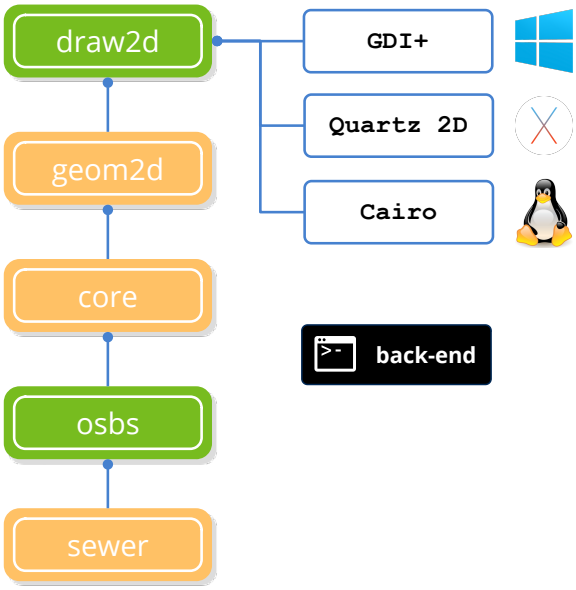


Figura 17.1: Dependencias de *draw2d*. Ver *"NAppGUI API"* (Página 147).

17.2. Contextos 2D

Los gráficos vectoriales se componen a base de primitivas básicas tales como líneas, círculos, texto, etc, utilizando el algoritmo del pintor (Figura 17.2): Las operaciones entrantes se superponen o solapan a las ya existentes. El resultado se va almacenando en un búfer intermedio conocido como *canvas* o *surface*. Esta superficie de dibujo forma parte de un objeto denominado **contexto** que también mantiene ciertos parámetros relacionados con el aspecto de las primitivas: Colores, atributos de línea, sistema de referencia, gradientes, etc.

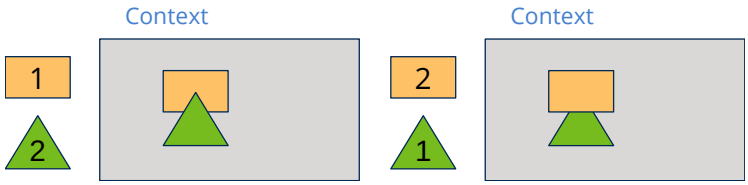


Figura 17.2: Algoritmo del pintor. Los nuevos objetos solaparán a los ya existentes.

Una de las ventajas de trabajar con formas paramétricas, es que pueden realizarse escalados de la imagen sin perdida de calidad (Figura 17.3). Esto es debido a que la conversión a píxeles, proceso denominado rasterización (Figura 17.4), se realiza en tiempo real y se ajusta constantemente al cambio de los vectores. En imágenes bitmap, un aumento de tamaño tiene asociada una pérdida de calidad.

Draw2D permite trabajar con dos tipos de contextos (Figura 17.5).

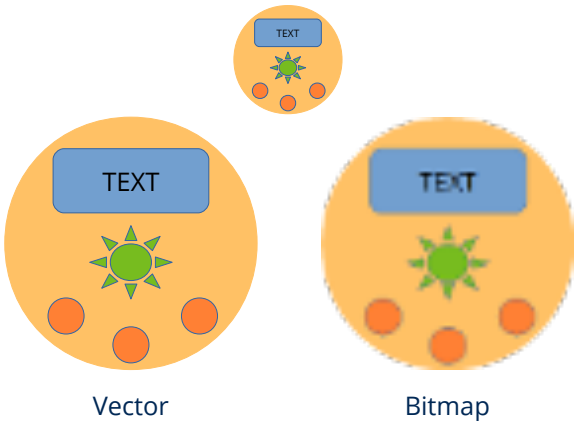


Figura 17.3: Escalado vectorial y escalado bitmap.

Figura 17.4: Rasterización de un círculo.

- Contexto ventana. El destino será un área dentro de la interfaz de usuario gestionada por un control `View`. Este control mantiene su propio contexto de dibujo y lo envía "listo para usar" a través del evento `EvDraw` (Listado 17.1).

Listado 17.1: Dibujar en una ventana.

```
static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);

    draw_clear(p->ctx, color_rgb(200, 200, 200));
    draw_fill_color(p->ctx, color_rgb(0, 128, 0));
    draw_rect(p->ctx, ekFILL, 100, 100, 200, 100);
    draw_fill_color(p->ctx, color_rgb(0, 0, 255));
    draw_circle(p->ctx, ekFILL, 450, 150, 75);
}

View *view = view_create();
view_size(view, s2df(600, 400));
view_OnDraw(view, listener(app, i_OnDraw, App));
```

- Contexto imagen. Aquí los comandos de dibujo se volcarán directamente en memoria para, posteriormente, obtener una imagen con el resultado final (Listado 17.2).

Listado 17.2: Dibujar en una imagen.

```
static i_draw(void)
{
    Image *image = NULL;
    DCtx *ctx = dctx_bitmap(600, 400, ekRGBA32);

    draw_clear(ctx, color_rgb(200, 200, 200));
    draw_fill_color(ctx, color_rgb(0, 128, 0));
    draw_rect(ctx, ekFILL, 100, 100, 200, 100);
    draw_fill_color(ctx, color_rgb(0, 0, 255));
    draw_circle(ctx, ekFILL, 450, 150, 75);

    image = dctx_image(&ctx);
    image_to_file(image, "drawing.png", NULL);
    image_destroy(&image);
}
```

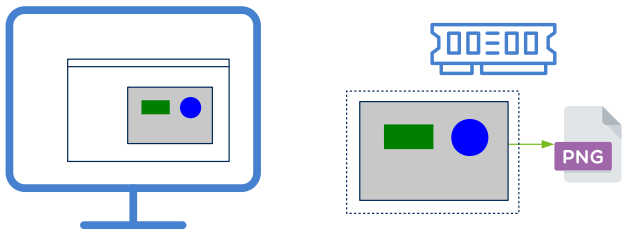


Figura 17.5: Contextos ventana e imagen.

Como podemos observar viendo el código, el dibujo propiamente dicho se realiza de la misma manera, lo único que cambia es como hemos obtenido el contexto (`DCtx`). Esto permite que podamos escribir rutinas gráficas genéricas sin preocuparnos sobre cual será el destino del resultado final. En el ejemplo *DrawImg*[1] tienes un desarrollo práctico paso a paso del uso de contextos. Las imágenes que acompañan el resto del capítulo han sido obtenidas de dicha aplicación.

Debido a que no es imperativo disponer de una ventana para dibujar, Draw2d puede utilizarse en aplicaciones de consola para componer o editar imágenes de forma automatizada.

17.2.1. Sistemas de referencia

El origen de coordenadas del dibujo se sitúa en la esquina superior izquierda (Figura 17.6). Las **X** positivas se desplazan hacia la izquierda y las **Y** positivas hacia abajo. Las unidades se miden en píxeles (o puntos en *"Pantallas retinaPantallas retina"* (Página 270)). Por ejemplo, el comando:

```
draw_circle(ctx, ekSKFILL, 300, 200, 100);
```

[1] https://nappgui.com/es/howto/drawimg.html

dibujará un círculo de 100 píxeles de radio cuyo centro se encuentra a 300 píxeles a la izquierda y 200 píxeles hacia abajo del origen. A este sistema inicial se le denomina **identidad** ya que todavía no ha sido manipulado, como veremos a continuación.

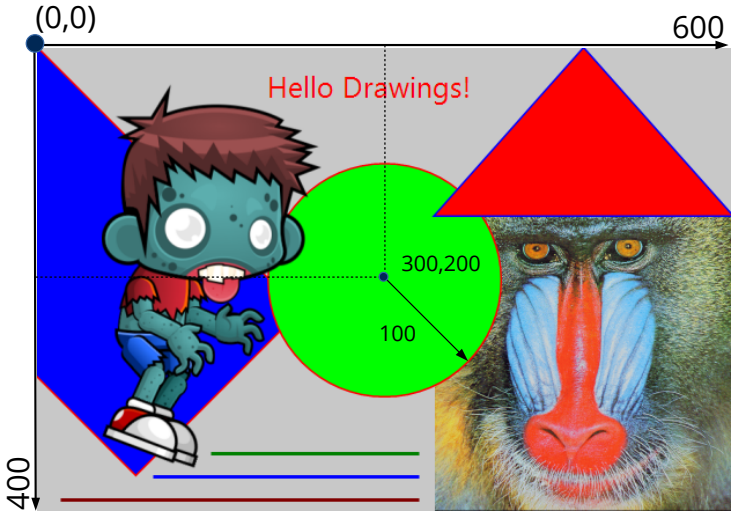


Figura 17.6: Sistema de referencia **identidad** en contextos 2D.

Aunque la escala inicial esté en píxeles, debemos desterrar la idea de que estamos manipulando directamente píxeles al dibujar. Los contextos de dibujo utilizan coordenadas en punto flotante. Por ejemplo, dibujar una línea entre los puntos (0.23, 1.432) y (-45.29, 12.6756) es perfectamente válido. Las transformaciones y el antialiasing pueden alterar ligeramente la posición o el grosor de ciertas líneas. Tampoco debemos esperar resultados "idénticos" a nivel de píxel al migrar las aplicaciones a diferentes plataformas, ya que cada sistema utiliza sus propios algoritmos de rasterización. Debemos pensar que estamos dibujando en el plano real. Para manipular directamente los píxeles de una imagen, consulta "Pixel Buffer" (Página 286).

Este sistema de referencia inicial se puede manipular mediante *"Transformaciones 2D"* (Página 246). Las transformaciones más comunes en gráficos son: Traslaciones (Figura 17.7), Rotaciones (Figura 17.8) y Escalados (Figura 17.9).

- `draw_matrixf` cambiará el sistema de referencia del contexto.

Listado 17.3: Traslación del origen de coordenadas 100 unidades en ambas direcciones.

```
T2Df t2d;
t2d_movef(&t2d, kT2D_IDENTf, 100, 100);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

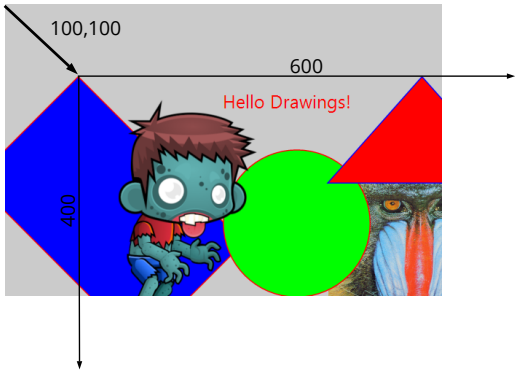


Figura 17.7: Traslación (Listado 17.3).

Listado 17.4: Rotación del origen de coordenadas 15 grados.

```
T2Df t2d;
t2d_rotatef(&t2d, kT2D_IDENTf, 15 * kBMATH_DEG2RADf);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

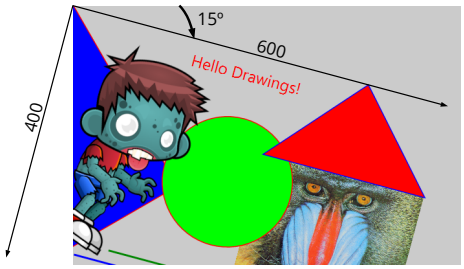


Figura 17.8: Rotación (Listado 17.4).

Listado 17.5: Escalado, reducción del tamaño a la mitad.

```
T2Df t2d;
t2d_scalef(&t2d, kT2D_IDENTf, .5f, .5f);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

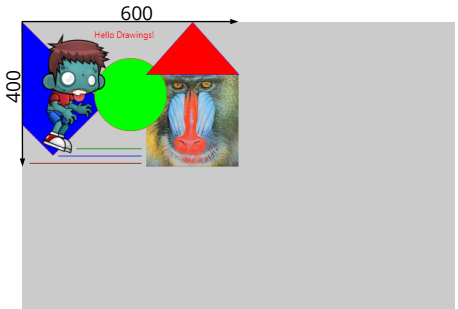


Figura 17.9: Escalado (Listado 17.5).

Las transformaciones se pueden acumular, pero debemos tener presente que no son operaciones conmutativas, sino que el orden en que se aplican influirá en el resultado final. Por ejemplo en (Figura 17.10) observamos que el dibujo se ha desplazado `(100, 50)` píxeles, en lugar de los `(200, 100)`, debido a que la traslación está afectada por el escalado previo. Más detalles en *"Composición de transformacionesComposición de transformaciones"* (Página 249).

Listado 17.6: Composición de transformaciones.

```
T2Df t2d;
t2d_scalef(&t2d, kT2D_IDENTf, .5f, .5f);
t2d_movef(&t2d, &t2d, 200, 100);
t2d_rotatef(&t2d, &t2d, 15 * kBMATH_DEG2RADf);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

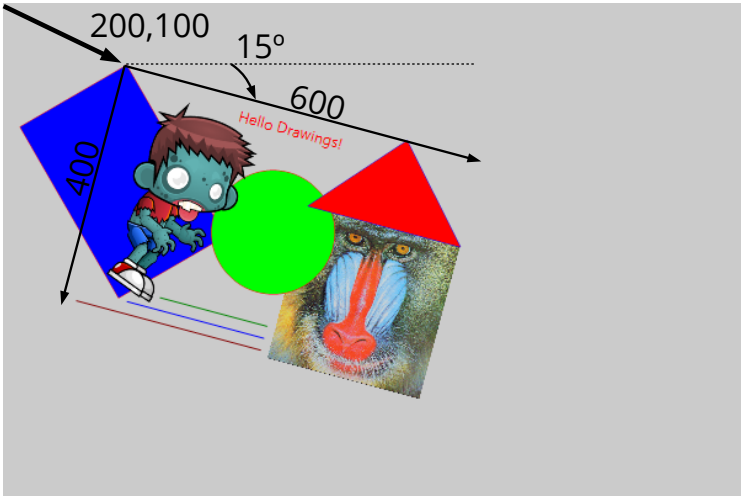


Figura 17.10: Composición de transformaciones (Listado 17.6).

17.2.2. Sistemas cartesianos

Existe una dicotomía al dibujar en 2D: Por un lado, tradicionalmente los sistemas de escritorio y las imágenes digitales sitúan el origen de coordenadas en la esquina superior izquierda con el eje Y creciendo hacia abajo (Figura 17.11). Por otro lado, los sistemas cartesianos utilizados en geometría lo sitúan en la esquina inferior izquierda, con Y creciendo hacia arriba. Esto genera un dilema sobre si un sistema es mejor que otro.

La respuesta es claramente no. Incluso, en el mismo dibujo, es posible que necesitemos combinar ambos en función del elemento que estemos tratando. Para textos e imágenes, el sistema de pantalla es más intuitivo ya que reproduce el papel o lienzo del mundo físico.

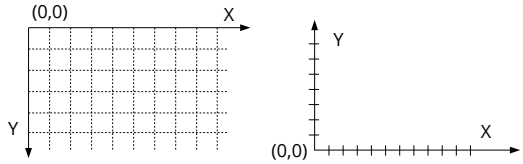


Figura 17.11: Sistema 2D en monitores (izquierda) y cartesiano (derecha).

Para funciones matemáticas, gráficos de barras, planos y otros aspectos relacionados con el mundo técnico, el cartesiano resulta mucho más cómodo y natural.

- `draw_matrix_cartesianf` establece el sistema de referencia del contexto en coordenadas cartesianas. En (Figura 17.12) hemos utilizado un sistema cartesiano de 6x4 unidades mapeado sobre una ventana de 600x400 píxeles.

Listado 17.7: Dibujo en coordenadas cartesianas.

```
T2Df t2d;
draw_line_color(ctx, color_rgb(255, 0, 0));
draw_line_width(ctx, .03);
draw_fill_color(ctx, color_rgb(0, 0, 255));
t2d_scalef(&t2d, kT2D_IDENTf, 100, 100);
draw_matrix_cartesianf(ctx, &t2d);
draw_rect(ctx, ekSKFILL, 1.5f, .1f, 1, 2);
draw_line_color(ctx, color_rgb(0, 128, 0));
draw_line(ctx, 0, 0, 1.5f, 2.1f);
```

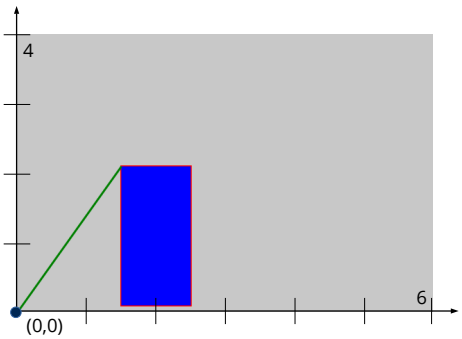


Figura 17.12: Coordenadas cartesianas (Listado 17.7).

17.2.3. Antialiasing

Dada la naturaleza discreta de los monitores e imágenes digitales, se produce un efecto de escalonado (dientes de sierra) al transformar las primitivas vectoriales a píxeles (Figura 17.13). Este efecto se hace menos perceptible a medida que aumenta la resolución de la imagen, pero aún así el "pixelado" sigue estando patente. El **antialiasing**, es una técnica que reduce este efecto de escalón variando ligeramente los colores de los píxeles en el entorno cercano a las líneas y contornos (Figura 17.14). Con esto se consigue engañar al ojo humano difuminando los bordes y generando imágenes de mayor calidad visual. Como

contrapartida tenemos el coste en el rendimiento de aplicarlo, aunque hace años que los cálculos relativos al antialiasing se realizan directamente en hardware (Figura 17.15), por lo que el impacto será mínimo.

- `draw_antialias` permite activar o desactivar los cálculos del antialiasing.

Figura 17.13: Antialiasing desactivado.

Figura 17.14: Antialiasing activado.

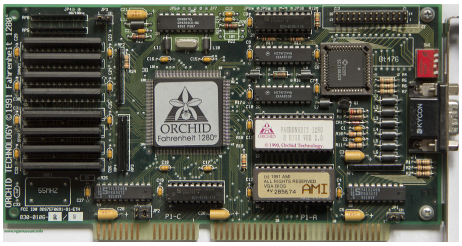


Figura 17.15: Orchid Fahrenheit 1280 (1992). Una de las primeras tarjetas que incorporaban aceleración gráfica 2d.

17.2.4. Pantallas retina

Al final del 2014 Apple presentó su nueva línea iMac con *Retina Display* de alta resolución (5120x2880). Normalmente, estos monitores trabajan en modo **escalado** (2560x1440) lo que permite tener píxeles de doble densidad (Figura 17.16). Apple diferencia entre **puntos** en pantalla, que son los que realmente manipula la aplicación y los píxeles físicos. Por lo tanto, nuestra ventana de 600x400 realmente tendrá 1200x800 píxeles en ordenadores Retina, aunque la aplicación seguirá "viendo" solo 600x400 puntos. El sistema operativo

realiza la conversión de forma transparente. De hecho, no tenemos que hacer nada para adaptar nuestro código, ya que funcionará de la misma forma tanto en iMac normales como en los equipados con monitores Retina.

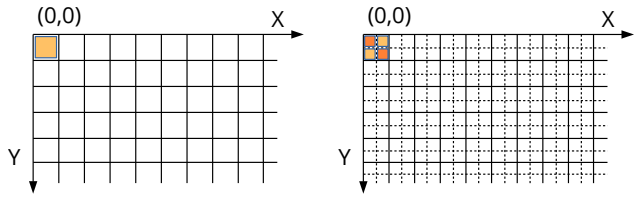


Figura 17.16: Píxeles de doble densidad en *Retina Display* (derecha).

Esta doble densidad la aprovechará el rasterizador para generar imágenes de mayor calidad al disponer de más píxeles en el mismo área de pantalla. En (Figura 17.17) y (Figura 17.18) vemos la calidad extra que aportan estos modelos de monitor.

Figura 17.17: Monitor normal (con antialiasing).

Figura 17.18: *Retina Display* (con antialiasing).

17.3. Primitivas de dibujo

A la hora de dibujar en contextos 2D disponemos de una serie de formas elementales tales como líneas, figuras, texto e imágenes. En *DrawHello*[2] tienes el código fuente de la aplicación que nos acompañará a lo largo de esta sección.

[2] `https://nappgui.com/es/howto/drawhello.html`

17.3.1. Dibujo de líneas

La operación más elemental es dibujar una recta entre dos puntos. En contextos 2d las líneas son objetos sólidos y no una mera hilera de píxeles. Pensemos que estamos utilizando rotuladores de punta gruesa, donde la línea teórica permanecerá siempre en el centro del trazo (Figura 17.19). Podemos cambiar la forma de las terminaciones (linecap), de las uniones (linejoin) y establecer un patrón para líneas discontinuas.

- `draw_line` dibujará una línea.
- `draw_polyline` dibujará varias líneas conectadas.
- `draw_arc` dibujará un arco.
- `draw_bezier` dibujará una curva de Bézier de grado 3 (cúbica).
- `draw_line_color` establecerá el color de línea.
- `draw_line_width` establecerá el ancho de línea.
- `draw_line_cap` establecerá el estilo de los extremos.
- `draw_line_join` establecerá el estilo de las uniones.
- `draw_line_dash` establecerá un patrón de puntos para líneas discontinuas.

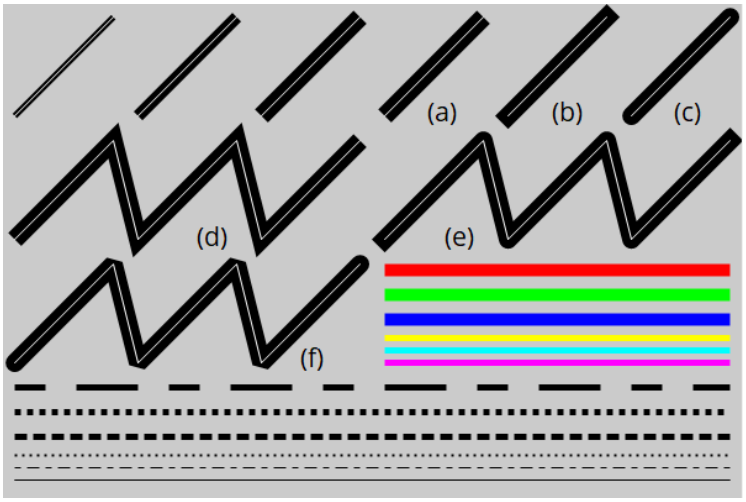


Figura 17.19: Diferentes estilos línea. **(a)** `ekLCFLAT`. **(b)** `ekLCSQUARE`. **(c)** `ekLCROUND`. **(d)** `ekLJMITER`. **(e)** `ekLJROUND`. **(f)** `ekLJBEVEL`. Los patrones: [5, 5, 10, 5], [1, 1], [2, 1], [1, 2], [5, 5, 10, 5], NULL.

17.3.2. Figuras y contornos

Para dibujar figuras o áreas cerradas tenemos varios comandos. Como vemos en (Figura 17.20) podemos dibujar el contorno de la figura, su interior o ambos. Para el contorno se tendrá en cuenta el estilo de línea establecido según hemos visto en el apartado anterior.

- `draw_rect` para rectángulos.
- `draw_rndrect` para rectángulos con bordes redondeados.
- `draw_circle` para círculos.
- `draw_ellipse` para elipses.
- `draw_polygon` para polígonos.
- `draw_fill_color` establece el color de relleno de regiones.

Listado 17.8: Dibujo de figuras (contornos y/o rellenos).

```
draw_fill_color(ctx, kCOLOR_BLUE);
draw_line_color(ctx, kCOLOR_BLACK);
draw_rect(ctx, ekSTROKE, 10, 10, 110, 75);
draw_rndrect(ctx, ekFILL, 140, 10, 110, 75, 20);
draw_circle(ctx, ekSKFILL, 312, 50, 40);
draw_ellipse(ctx, ekFILLSK, 430, 50, 55, 37);
```

Como ya vimos en *"Contextos 2D"* (Página 263), el orden en que se efectúan las operaciones importa. No es lo mismo rellenar y luego dibujar el contorno que viceversa. El centro del trazo coincidirá con el contorno teórico de la figura.

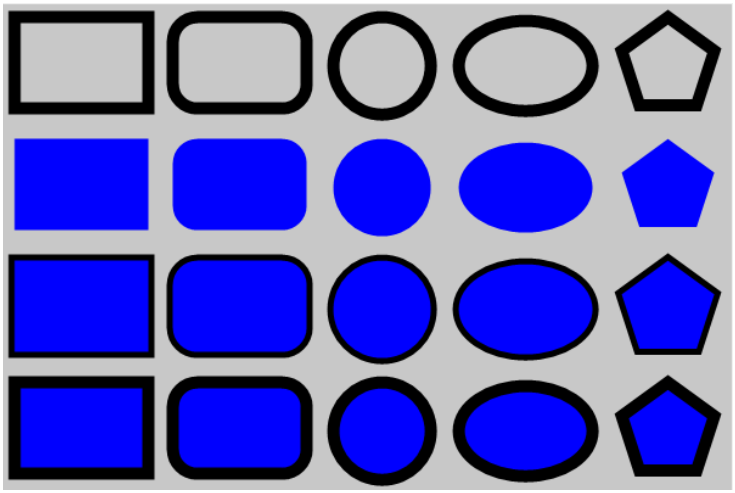


Figura 17.20: Solo contorno `ekSTROKE`. Solo relleno `ekFILL`. Primero contorno, después relleno `ekSKFILL`. Primero relleno, después contorno `ekFILLSK`.

17.3.3. Gradientes

Los gradientes permiten rellenar regiones utilizando un degradado en lugar de un color sólido (Figura 17.21). Se definen varios colores base y su posición relativa a lo largo de un vector. Las posiciones [0, 1] corresponden a los extremos y los valores dentro de este rango a las posibles paradas intermedias. Cada línea perpendicular al vector delimita un color uniforme que se extenderá indefinidamente hasta alcanzar los límites de la figura a rellenar.

- Utiliza `draw_fill_linear` para activar el relleno con degradados.
- Utiliza `draw_fill_color` para volver al relleno de color sólido.

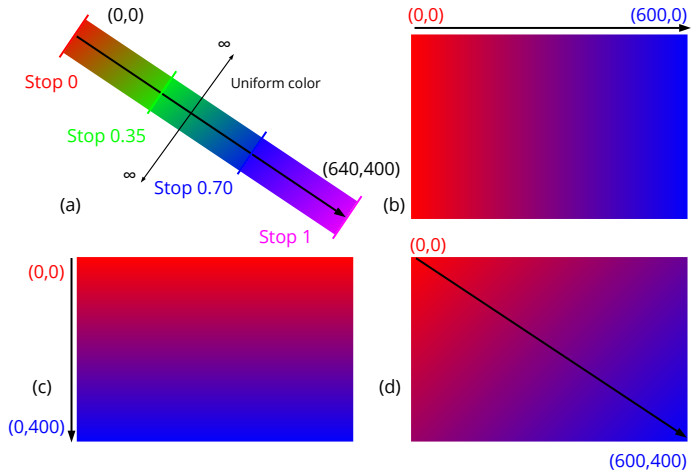


Figura 17.21: Gradientes lineales. El color se interpola a lo largo de un vector.

Listado 17.9: Definición de los gradientes de (Figura 17.21).

```
// (a) Gradient
color_t color[4];
real32_t stop[4] = {0, .35f, .7f, 1};
color[0] = color_rgb(255, 0, 0);
color[1] = color_rgb(0, 255, 0);
color[2] = color_rgb(0, 0, 255);
color[3] = color_rgb(255, 0, 255);
draw_fill_linear(ctx, color, stop, 4, 0, 0, 600, 400);

// (b) Gradient
color_t color[2];
real32_t stop[2] = {0, 1};
color[0] = color_rgb(255, 0, 0);
color[1] = color_rgb(0, 0, 255);
draw_fill_linear(ctx, color, stop, 2, 0, 0, 600, 0);

// (c) Gradient
color_t color[2];
real32_t stop[2] = {0, 1};
```

```
color[0] = color_rgb(255, 0, 0);
color[1] = color_rgb(0, 0, 255);
draw_fill_linear(ctx, color, stop, 2, 0, 0, 0, 400);

// (d) Gradient
color_t color[2];
real32_t stop[2] = {0, 1};
color[0] = color_rgb(255, 0, 0);
color[1] = color_rgb(0, 0, 255);
draw_fill_linear(ctx, color, stop, 2, 0, 0, 600, 400);
```

17.3.4. Transformación del gradiente

Dado que el gradiente se define mediante un vector, es posible establecer una transformación que cambie la forma en la que es aplicado. Esta matriz es totalmente independiente a la que se aplica a las primitivas de dibujo `draw_matrixf`, como vimos en *"Sistemas de referenciaSistemas de referencia"* (Página 265).

- Utiliza `draw_fill_matrix` para establecer la transformación del gradiente. Con esto podemos conseguir varios efectos:
- **Gradiente global:** El degradado se aplicará de forma global al fondo, y las figuras serán recortes del mismo patrón (Figura 17.22). Para ello estableceremos la matriz identidad como transformación del gradiente (Listado 17.10). Es la definida por defecto.

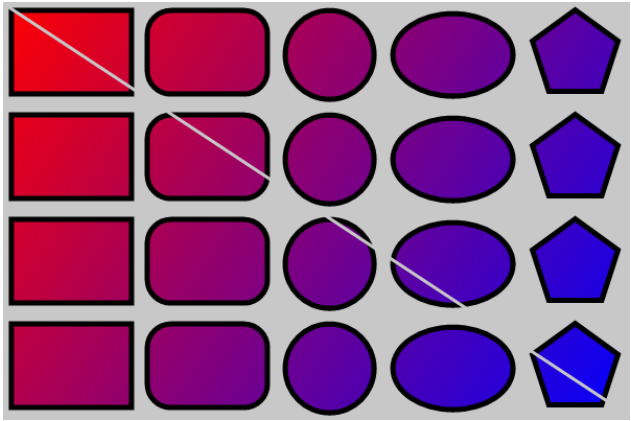


Figura 17.22: Gradiente global. No se pierde la continuidad entre figuras.

Listado 17.10: Matriz del gradiente para todo el dibujo.

```
draw_fill_linear(ctx, c, stop, 2, 0, 0, 600, 400);
draw_fill_matrix(ctx, kT2D_IDENTf);
i_draw_shapes(ctx);
```

- **Gradiente local:** El vector es trasladado al origen de la figura o a un punto de su entorno cercano (Figura 17.23). Con esto, conseguiremos aplicar el degradado de forma local y que solo afecte a una figura concreta. En (Listado 17.11) hemos variado ligeramente la transformación para fijar el origen en una esquina y no en el centro de la elipse. Esto puede variar en función del efecto deseado.

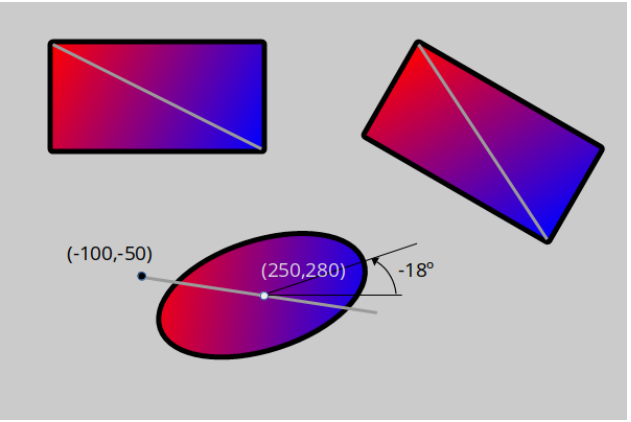


Figura 17.23: Gradiente local. El origen se sitúa en la figura.

Listado 17.11: Matriz del gradiente para la elipse de (Figura 17.23).

```
T2Df t2d;
t2d_movef(&t2d, kT2D_IDENTf, 250, 280);
t2d_rotatef(&t2d, &t2d, - kBMATH_PIf / 10);
draw_matrixf(ctx, &t2d);      // Geometry matrix
draw_fill_linear(ctx, c, stop, 2, 0, 0, 200, 100);
t2d_movef(&t2d, &t2d, -100, -50);
draw_fill_matrix(ctx, &t2d); // Gradient matrix
draw_ellipse(ctx, ekSKFILL, 0, 0, 100, 50);
```

17.3.5. Gradientes en líneas

Además del relleno de regiones, los gradientes también pueden aplicarse a líneas y contornos (Figura 17.24) (Listado 17.12).

- Utiliza `draw_line_fill` para dibujar las líneas con el patrón de relleno actual.
- Utiliza `draw_line_color` para volver al color sólido.

Listado 17.12: Gradientes en líneas.

```
draw_fill_linear(ctx, c, stop, 2, 0, 0, 600, 400);
draw_fill_matrix(ctx, kT2D_IDENTf);
draw_line_fill(ctx);
draw_bezier(ctx, 30, 200, 140, 60, 440, 120, 570, 200);
```

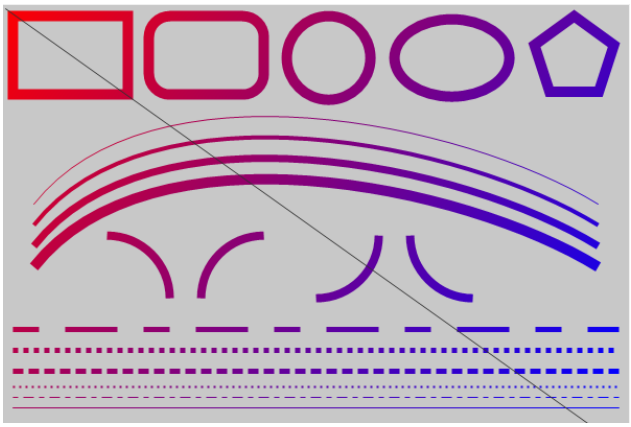


Figura 17.24: Dibujo de líneas utilizando gradientes.

17.3.6. Límites del gradiente

Como hemos comentado, el relleno de color se extenderá de manera uniforme e indefinida por todas las líneas perpendiculares al vector, pero... ¿Que ocurre fuera de los límites de este vector? En (Listado 17.13) (Figura 17.25) el gradiente se ha definido en x=[200, 400], siendo esta medida inferior a la figura a rellenar:

- Utiliza `draw_fill_wrap` para definir el comportamiento fuera de límites.
- `ekFCLAMP` se utiliza el valor del extremo como constante en el área exterior.
- `ekFTILE` se repite el patrón de color.
- `ekFFLIP` se repite el patrón, pero invirtiendo el sentido.

Listado 17.13: Color uniforme fuera de los límites del gradiente (Figura 17.25) **(a)**.

```
draw_fill_linear(ctx, c, stop, 2, 200, 0, 400, 0);
draw_fill_wrap(ctx, ekFCLAMP);
draw_rect(ctx, ekFILLSK, 50, 25, 500, 100);
```

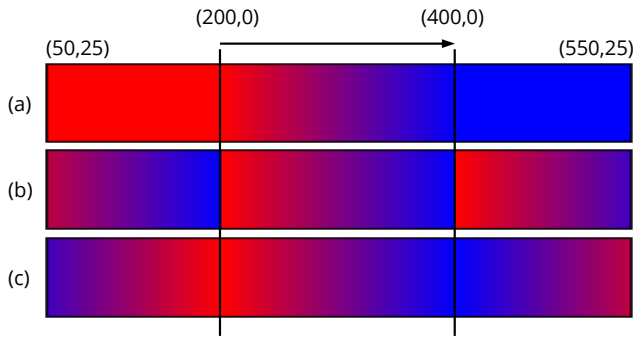


Figura 17.25: Comportamiento en los límites: **(a)** `ekFCLAMP`, **(b)** `ekFTILE`, **(c)** `ekFFLIP`.

17.3.7. Dibujar texto

El renderizado del texto es la parte más importante de la interfaz de usuario. Antiguamente, se utilizaban pequeños *bitmaps* con la imagen de cada carácter, pero a principios de los 90 entraron en juego las fuentes vectoriales basadas en curvas de Bezier. La gran cantidad de tipografías, el inmenso juego de carácteres *"Unicode"* (Página 157) y la posibilidad de escalar, rotar, o maquetar el texto en párrafos supuso un gran reto técnico en aquellos años. Afortunadamente, toda esta casuística está ampliamente resuelta por las APIs de los sistemas operativos modernos, lo que nos permite proporcionar una interfaz simplificada para añadir texto a nuestros dibujos.

- Utiliza `draw_text` para dibujar textos en contextos 2D.
- Utiliza `draw_text_color` para establecer el color del texto.
- Utiliza `draw_font` para establecer la fuente tipográfica.
- Utiliza `draw_text_width` para establecer el ancho máximo de un bloque de texto.
- Utiliza `draw_text_trim` par indicar como será cortado el texto.
- Utiliza `draw_text_align` para establecer la alineación de un bloque de texto.
- Utiliza `draw_text_halign` para establecer la alineación interna del texto.
- Utiliza `draw_text_extents` para obtener el tamaño de un bloque de texto.

Para dibujar textos de una sola línea, tan solo debemos llamar a la función pertinente, pasando una cadena UTF8 (Listado 17.14) (Figura 17.26). Previamente, podemos asignar la fuente, el color y su alineación.

Listado 17.14: Dibujo de una línea de texto.

```
Font *font = font_system(20, 0);
draw_font(ctx, font);
draw_text_color(ctx, kCOLOR_BLUE);
draw_text_align(ctx, ekLEFT, ekTOP);
draw_text(ctx, "Text □□Κείμενο ", 25, 25);
```

Si la cadena a visualizar tiene saltos de línea (carácter '\n') se tendrán en cuenta y el texto se mostrará en varias líneas (Listado 17.15) (Figura 17.27). También es posible obtener su medida en píxeles, útil para evitar solapamientos con otras primitivs.

Listado 17.15: Dibujo de textos con saltos de línea.

```
const char_t *text = "Text new line\□□□□n\Γραμμήη κειμένου";
real32_t w, h;
draw_text(ctx, text, 25, 25);
draw_text_extents(ctx, text, -1, &w, &h);
```

Figura 17.26: Textos de una sola línea, con alineación y transformaciones.

Figura 17.27: Textos con carácter '\n'.

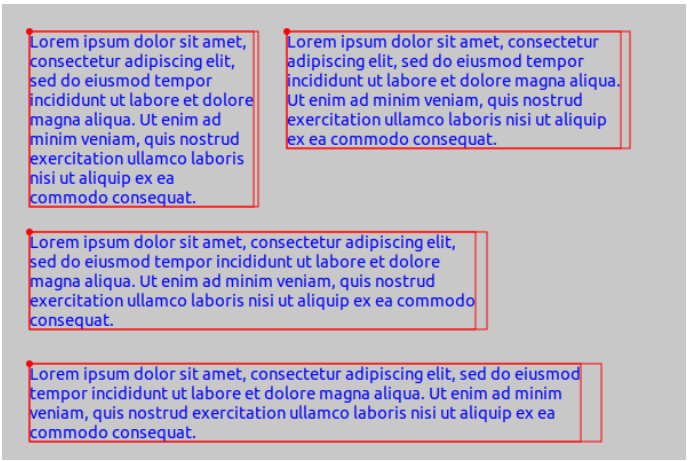
Si el texto no contiene saltos de línea, se dibujará de forma contínua expandiéndose horizontalmente. Esto puede no ser lo más apropiado en párrafos largos, por lo que podemos establecer un ancho máximo, forzando su dibujo en varias líneas (Listado 17.16) (Figura 17.28).

Listado 17.16: Anchura máxima y alineación interna en bloques de texto.

```
const char_t *text = "Lorem ipsum dolor sit amet...consequat";
draw_text_width(ctx, 200);
draw_text_halign(ctx, ekLEFT);
draw_text(ctx, text, 25, 25);
draw_text_extents(ctx, text, 200, &w, &h);
```

Por último, podemos utilizar `draw_text_path` para tratar al texto como cualquier otra región geométrica, resaltando el borde o rellenando con gradientes. En este caso `draw_text_color` no tendrá efecto y se utilizarán los valores de `draw_fill_color`, `draw_fill_linear` y `draw_line_color` (Listado 17.17) (Figura 17.29).

Listado 17.17: Texto con borde punteado y relleno con gradiente.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Figura 17.28: Párrafos de texto con límite de anchura. Se muestra la anchura máxima y la real obtenida con `draw_text_extents`.

```
color_t c[2];
real32_t stop[2] = {0, 1};
real32_t dash[2] = {1, 1};
c[0] = kCOLOR_BLUE;
c[1] = kCOLOR_RED;
draw_line_dash(ctx, dash, 2);
draw_line_color(ctx, kCOLOR_GREEN);
draw_text_extents(ctx, "Gradient dashed text", -1, &w, &h);
draw_fill_linear(ctx, c, stop, 2, 25, 0, 25 + w, 0);
draw_text_path(ctx, ekFILLSK, "Gradient dashed text", 25, 250);
```

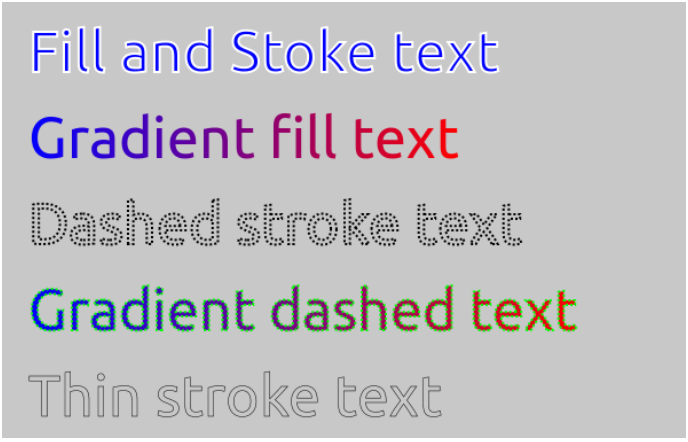


Figura 17.29: Combinando rellenos y bordes.

`draw_text` es mucho más rápida que `draw_text_path`, por lo que debemos limitar el uso de esta última a lo estrictamente necesario.

17.3.8. Dibujar imágenes

Imágenes generadas de forma procedimental o leídas desde disco pueden utilizarse como una primitiva de dibujo más (Listado 17.18) (Figura 17.30). Al igual que ocurre con el texto u otras figuras, la transformación del contexto afectará a la geometría de la imagen.

- Utiliza `draw_image` para dibujar una imagen.
- Utiliza `draw_image_frame` para dibujar una secuencia de una animación (GIF).
- Utiliza `draw_image_align` para establecer la alineación de la imagen.

Listado 17.18: Dibujo de imagen desplazada y rotada.

```
const Image *image = image_from_resource(pack, IMAGE_JPG);
T2Df t2d;
t2d_movef(&t2d, kT2D_IDENTf, 300, 200);
t2d_rotatef(&t2d, &t2d, kBMATH_PIf / 8);
draw_image_align(ctx, ekCENTER, ekCENTER);
draw_matrixf(ctx, &t2d);
draw_image(ctx, image, 0, 0);
```

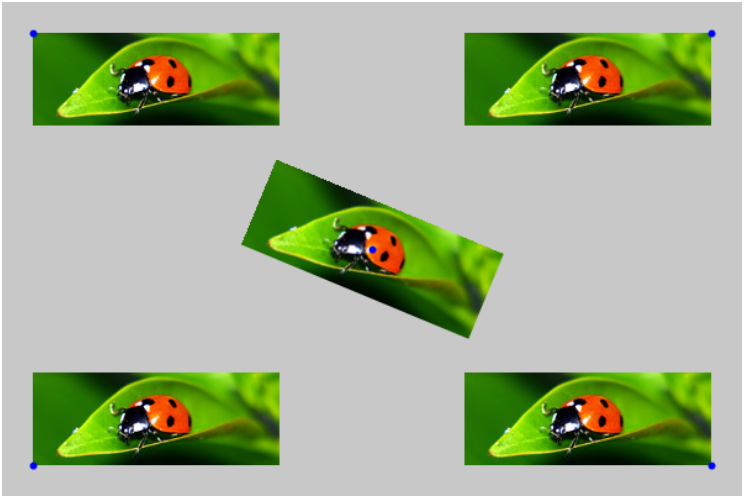


Figura 17.30: Dibujo de imágenes con alineación.

17.3.9. Parámetros por defecto

Cada contexto mantiene ciertos parámetros de estado. Al comienzo del dibujo, bien mediante el método `OnDraw` o tras crear el contexto con `dctx_bitmap` los valores por defecto son los mostrados en (Tabla 17.1):

Parámetro	Valor	Cambiar con
Matrix	Identidad (0,0) Esquina Sup-Izq, píxeles.	`draw_matrixf`
Antialiasing	`TRUE`	`draw_antialias`
LineColor	`kCOLOR_BLACK`	`draw_line_color`
LineWidth	1	`draw_line_width`
Linecap	`ekLCFLAT`	`draw_line_cap`
Linejoin	`ekLJMITER`	`draw_line_join`
LineDash	Sólido	`draw_line_dash`
TextColor	`kCOLOR_BLACK`	`draw_text_color`
FillColor	`kCOLOR_BLACK`	`draw_fill_color`
FillMatrix	Identidad (0,0) Esquina Sup-Izq, píxeles.	`draw_fill_matrix`
Font	Fuente del sistema, tamaño regular.	`draw_font`
Text max width	-1 (no limitado)	`draw_text_width`
Text vertical align	`ekLEFT`	`draw_text_align`
Text horizontal align	`ekTOP`	`draw_text_align`
Text internal align	`ekLEFT`	`draw_text_halign`
Image vertical align	`ekLEFT`	`draw_image_align`
Image horizontal align	`ekTOP`	`draw_image_align`

Tabla 17.1: Valores por defecto en contextos 2D.

17.4. Dibujo de entidades Geom2D

En la sección anterior hemos visto las primitivas básicas para dibujar en 2D. No obstante, *Draw2D* dispone de funciones especializadas para los objetos de *"Geom2D"* (Página 241). Estas nuevas funciones serían totalmente prescindibles, ya que podrías obtener el mismo resultado utilizando `draw_rect`, `draw_polygon`, etc. Se incluyen como un mero atajo, además de ofrecer una versión de las mismas basadas en *"Plantillas matemáticas-Plantillas matemáticas"* (Página 53), muy útiles al desarrollar algoritmos genéricos en C++. Las propiedades de línea y relleno serán las que estén vigentes en cada momento dentro del contexto, debido a: `draw_line_color`, `draw_line_width`, `draw_fill_color`, etc.

- Utiliza `draw_v2df` para dibujar un punto.

- Utiliza `draw_seg2df` para dibujar un segmento.
- Utiliza `draw_cir2df` para dibujar un círculo.
- Utiliza `draw_box2df` para dibujar una caja alineada.
- Utiliza `draw_obb2df` para dibujar una caja orientada.
- Utiliza `draw_tri2df` para dibujar un triángulo.
- Utiliza `draw_pol2df` para dibujar un polígono.

Un ejemplo completo del uso de entidades 2D lo tienes en *Col2DHello*[3] (Figura 17.31). Además de dibujar, en esta aplicación se muestran otros conceptos referentes a gráficos y cálculo geométrico tales como:

- Crear objetos 2D bajo demanda.
- Interactividad *Click+Drag*.
- Detección de colisiones.
- Cálculo de áreas.
- Triangulación de polígonos y descomposición en componentes convexas.
- Cálculo del círculo óptimo que envuelve a un conjunto de puntos.
- Cálculo de la caja orientada (`OBB2Df`) que mejor representa a un conjunto de puntos.
- Cálculo de la envoltura convexa *(Convex Hull)*.

17.5. Colores

Los colores en Draw2D se codifican mediante un entero de 32 bits con los cuatro canales RGBA en Little-Endian: El rojo en el byte 0, verde en el 1, azul en el 2 y alfa (o transparencia) en el 3 (Figura 17.32). Se utiliza el alias `color_t` como equivalencia a `uint32_t`. En el caso particular de que el byte 3 sea igual a 0 (totalmente transparente), los tres primeros bytes no contendrán información RGB, sino un índice a un color predefinido.

- Utiliza `color_rgba` para crear un color mediante sus componentes RGBA.
- Utiliza `color_get_rgba` para obtener las componentes RGBA.
- Utiliza `color_html` para traducir una cadena en formato HTML (`"#RRGGBB"`).
- Utiliza `kCOLOR_BLACK` y similares para acceder a colores básicos predefinidos.

[3] `https://nappgui.com/es/howto/col2dhello.html`

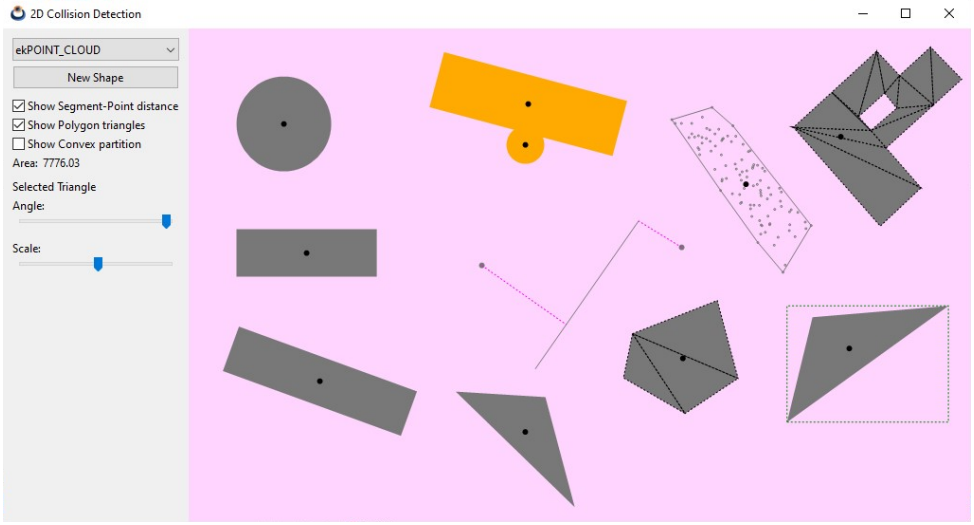


Figura 17.31: Aplicación *Col2dHello*, que ilustra como trabajar con geometría 2D.

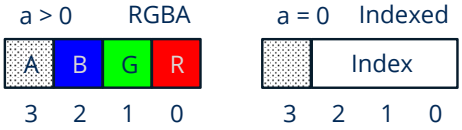


Figura 17.32: Representación de un valor RGBA en 32 bits.

17.5.1. Espacio HSV

La representación RGB se basa en la adición de los tres colores de luz primarios. Es la más extendida dentro de la generación de imágenes por computador sobre todo a la hora de calcular sombreados y reflexiones. También se utiliza en TV, monitores o proyectores donde cada pixel se obtiene combinado la luz de tres emisores. No obstante, es muy poco intuitiva para la edición de colores por parte de los humanos. Por ejemplo, dado un color en RGB, es muy complicado aumentar el brillo o variar el tono (entre rojo y naranja, por ejemplo) manipulando la tripleta (r, g, b). El espacio HSV (*Hue*, *Saturation*, *Value*) también llamado HSB (*Brightness*) soluciona este problema, ya que el efecto de alterar este grupo de valores será altamente predecible (Figura 17.33).

- Utiliza `color_hsbf` para crear un color RGB a partir de sus componentes **H**, **S**, **B**.
- Utiliza `color_to_hsbf` para obtener las componentes **H**, **S**, **B**.
- **Hue (Tono):** Valor continuo cíclico entre 0 y 1. Donde `0`=Rojo, `1/3`=Verde, `2/3`=Azul, `1`=Rojo (Tabla 17.2).
- **Saturación:** Es equivalente a añadir pintura blanca al tono base. Cuando `S=1` no se añade nada de blanco (saturación máxima, color puro). Pero si `S=0` tendremos un blanco puro, independientemente del tono.

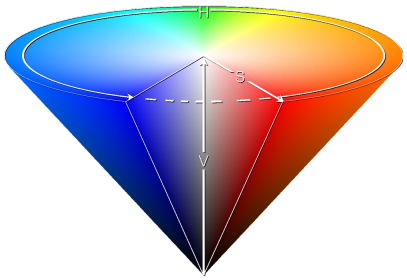


Figura 17.33: Espacio HSV representado por un cono invertido. A medida que se reduzca V, también lo hará la cantidad de colores disponibles.

- **Brillo:** Es equivalente a añadir pintura negra a la combinación `HS`. Si `B=1` no se añade nada de negro (brillo máximo). Si `B=0` tendremos un negro puro, independientemente del tono y la saturación.

RGB			HSV
(0,0,0)		`kCOLOR_BLACK`	(?,?,0)
(1,1,1)		`kCOLOR_WHITE`	(?,0,1)
(1,0,0)		`kCOLOR_RED`	(0,1,1)
(1,1,0)		`kCOLOR_YELLOW`	(1/6,1,1)
(0,1,0)		`kCOLOR_GREEN`	(1/3,1,1)
(0,1,1)		`kCOLOR_CYAN`	(1/2,1,1)
(0,0,1)		`kCOLOR_BLUE`	(2/3,1,1)
(1,0,1)		`kCOLOR_MAGENTA`	(5/6,1,1)

Tabla 17.2: Equivalencia RGB/HSV.

Al contrario que ocurre en RGB, los HSV no son totalmente independientes. A medida que reduzcamos el brillo irá disminuyendo la cantidad de colores del mismo tono hasta llegar a `B=0` donde tendremos negro puro independientemente de H y S. Por otro lado, si `S=0` se anulará H y tendremos los diferentes tonos de gris a medida que cambie B desde 0 (negro) hasta 1 (blanco).

17.6. Paletas

Una paleta no es más que una lista de colores indexada (Figura 17.34), generalmente relacionada con un *"Pixel Buffer"* (Página 286). Su principal utilidad es ahorrar espacio en la representación de imágenes, ya que cada pixel se codifica mediante un índice de 1, 2, 4 u 8 bits en lugar del color real donde son necesarios 24 o 32 bits. Por esta razón lo habitual es tener paletas de 2, 4, 16 o 256 colores.

- Utiliza `palette_create` para crear una paleta.
- Utiliza `palette_colors` para acceder a los colores.

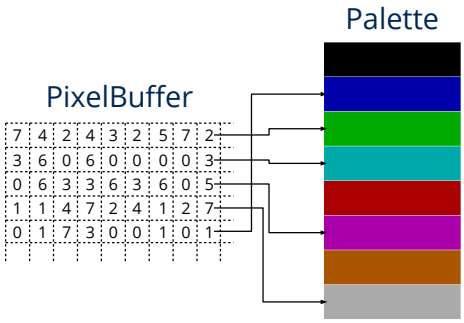


Figura 17.34: Paleta asociada con un pixel buffer indexado.

17.6.1. Paleta predefinida

Tenemos varias paletas predefinidas tanto en color (Figura 17.35) como en tonos de gris (Figura 17.36). La paleta RGB8 se ha creado combinando 8 tonos de rojo (3bits), 8 tonos de verde (3bits) y 4 tonos de azul (2bits). Esto es así porque el ojo humano distingue mucho menos la variación del azul que de los otros dos colores.

- Utiliza `palette_ega4` para crear una paleta predefinida de 16 colores.
- Utiliza `palette_rgb8` para crear una paleta de 256 colores.
- Utiliza `palette_gray4` y similares para crear una paleta en tonos de gris.
- Utiliza `palette_binary` para una paleta de dos colores.

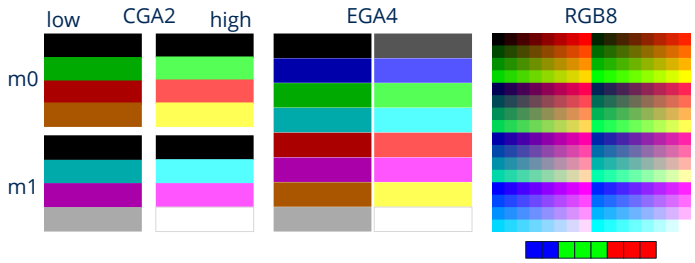


Figura 17.35: Paletas de color predefinidas.

17.7. Pixel Buffer

Un **pixel buffer** (`Pixbuf`) es una área de memoria que representa una rejilla discreta de puntos de color. Permiten el acceso directo a la información pero no están optimizados para el dibujo en pantalla, por lo que deberemos crear un objeto `Image` para poder visualizarlos. Son muy eficientes para la generación procedimental de imágenes o la aplicación de filtros, ya que leer o escribir un valor no requiere más que acceder a su posición dentro del búfer.

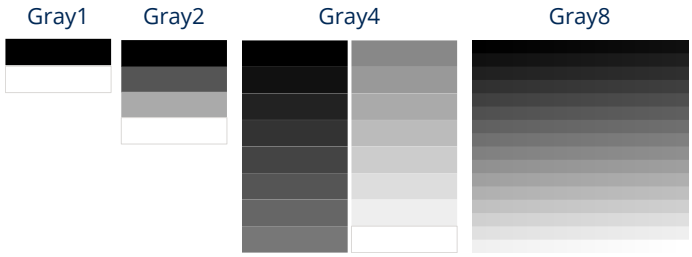


Figura 17.36: Paletas de gris predefinidas.

- Utiliza `pixbuf_create` para crear un píxel búfer.
- Utiliza `pixbuf_trim` para recortar un píxel búfer.
- Utiliza `pixbuf_width` para obtener la anchura de la rejilla.
- Utiliza `pixbuf_height` para obtener la altura de la rejilla.

Todas las operaciones sobre pixel buffers se realizan en la CPU. Son eficientes en la medida que accedemos directamente a memoria, pero no se pueden comparar con alternativas que utilicen la GPU para el procesamiento digital de imágenes.

17.7.1. Formatos de píxel

El formato hace referencia a como se codifica el valor de cada píxel dentro del búfer (Tabla 17.3) (Figura 17.37).

- Utiliza `pixbuf_format` para obtener el formato del píxel.
- Utiliza `pixbuf_format_bpp` para obtener el número de bits queridos por cada píxel.

Valor	Descripción
`ekRGB24`	*True color* +16 Millones simultáneos, 24 bits por píxel.
`ekRGBA32`	*True color* con canal alpha (transparencias), 32 bits por píxel.
`ekGRAY8`	256 tonos de gris, 8 bits por píxel.
`ekINDEX1`	Indexado, 1 bit por pixel.
`ekINDEX2`	Indexado, 2 bits por pixel.
`ekINDEX4`	Indexado, 4 bits por pixel.
`ekINDEX8`	Indexado, 8 bits por pixel.

Tabla 17.3: Formatos de pixel.

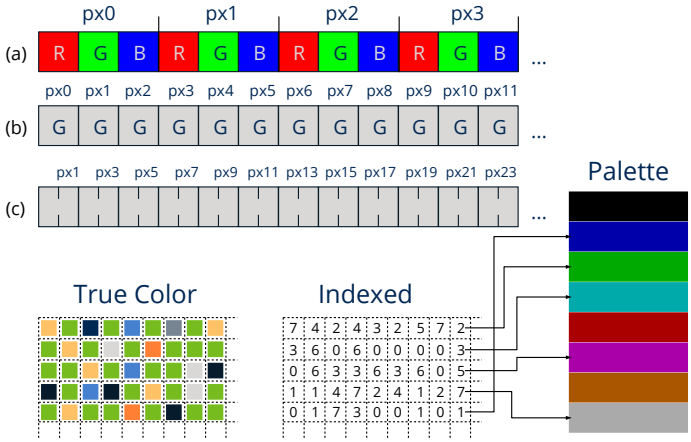


Figura 17.37: Formatos de píxel: **(a)** Color real, **(b)** tonos de gris, **(c)** indexados.

17.7.2. Imágenes procedimentales

Una forma de "rellenar" los buffers es mediante algoritmos que calculen el valor de cada píxel. Un claro ejemplo de imágen procedimental lo puedes encontrar en *Fractals*[4] (Figura 17.38), una aplicación enfocada en la representación de conjuntos fractales, un área de la matemática dedicada al estudio de determinados sistemas dinámicos.

- Utiliza `pixbuf_data` para obtener un puntero al contenido del búfer.
- Utiliza `pixbuf_set` para escribir el valor de un píxel.
- Utiliza `pixbuf_get` para leer el valor de un píxel.

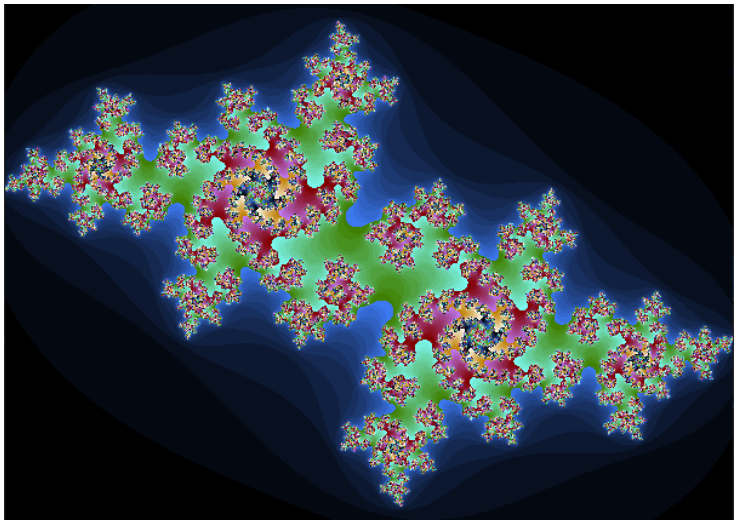


Figura 17.38: Conjunto de Julia. Imagen generada píxel a píxel mediante algoritmos fractales.

[4] `https://nappgui.com/es/demo/fractals.html`

Si bien `pixbuf_set` y `pixbuf_get` permiten la manipulación segura de píxeles, en ocasiones puede ser necesario obtener un extra en cuanto a rendimiento se refiere. En (Listado 17.19) tenemos algunas macros para el acceso directo al área de memoria devuelta por `pixbuf_data`. Utilizalas con sumo cuidado y sabiendo lo que haces, ya que no disponen de métodos de control de errores, por lo que es probable que se produzcan fallos de segmentación si no se usan correctamente.

Listado 17.19: Macros rápidas para la manipulación de un búfer tipo `ekINDEX1` (1 bit por pixel).

```
#define pixbuf_get1(data, x, y, w)\
    (uint32_t)((data[((y)*(w)+(x))/8] >> (byte_t)(((y)*(w)+(x))%8)) & 1)

#define pixbuf_set1(data, x, y, w, v)\
{\
    register byte_t *__ob = data + (((y)*(w))+(x))/8;\
    register byte_t __op = (byte_t)((((y)*(w))+(x))%8);\
    *__ob &= ~(1 << __op);\
    *__ob |= ((v) << __op);\
}
```

17.7.3. Copia y conversión

Durante el procesamiento digital de una imagen es posible que tengamos que encadenar varias operaciones, por lo que será de utilidad poder realizar copias de los búfer o conversiones de formato.

- Utiliza `pixbuf_copy` para realizar una copia.
- Utiliza `pixbuf_convert` para convertir entre formatos (Tabla 17.4).

Desde	Hacia	Observaciones
RGB24	RGB32	Se añade canal alfa con el valor 255
RGB32	RGB24	Se elimina el canal alfa con posible pérdida de información.
RGB(A)	Gray	Se ponderan los canales RGB a razón de 77/255, 148/255, 30/255. Se pierde el canal alfa.
Gray	RGB(A)	Se duplican los canales RGB(gray, gray, gray). Canal alfa a 255.
RGB(A)	Indexed	Se calcula la menor distancia entre cada píxel y la paleta. Posible pérdida de información.
Indexed	RGB(A)	Se utilizará la paleta para obtener cada valor RGBA.
Indexed	Indexed	Si el destino tiene menor número de bits, se aplicará $out = in \,\%\, bpp$ con posible pérdida de información.

Desde	Hacia	Observaciones
Gray	Indexed	El formato Gray8 se considerará indexado a todos los efectos.
Indexed	Gray	El formato Gray8 se considerará indexado a todos los efectos.

Tabla 17.4: Conversión entre formatos.

17.8. Imágenes

Existe una estrecha relación entre los pixel búfer y las imágenes. Si bien los primeros contienen la información de color "en crudo", las segundas son objetos vinculados directamente con la API gráfica de cada sistema, lo que permite dibujarlas en contextos 2d o visualizarlas en una ventana (Figura 17.39).

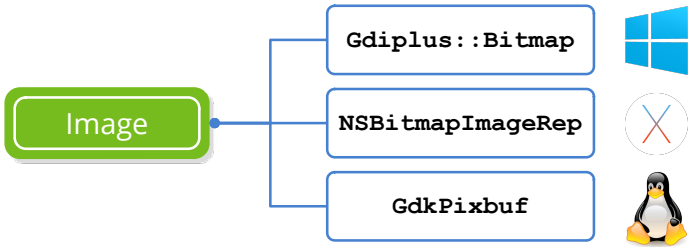


Figura 17.39: Los objetos `Image` tienen una vinculación directa con las APIs gráficas, mientras que los `Pixbuf` no.

La estructura de una imagen digital, también llamada *bitmap* o *raster graphics*, es la misma que la de un píxel búfer. Tenemos una rejilla discreta de puntos de color caracterizada por su resolución (ancho, alto) y profundidad, que es la cantidad de bits necesaria para codificar cada pixel (Figura 17.40). Las imágenes *bitmap* funcionan mejor para captar instantáneas del mundo real, donde es prácticamente imposible describir la escena mediante primitivas geométricas, como vimos en *"Primitivas de dibujo"* (Página 271). Como contrapartida, al estar compuesta por puntos discretos, no se comporta bien ante los cambios de tamaño donde sufrirá una pérdida de calidad.

Figura 17.40: A la izquierda imagen de 64x64 píxeles y 16 colores. A la derecha 256x256 píxeles y 16 millones de colores.

17.8.1. Cargar y visualizar imágenes

En la mayoría de ocasiones, lo único que necesitaremos saber sobre imágenes será como leerlas de disco u otro origen de datos para, posteriormente, visualizarlas en pantalla como parte de la interfaz de usuario (Listado 17.20) (Figura 17.41). Consideramos que las imágenes están almacenadas en alguno de los formatos estándar: JPG, PNG, BMP o GIF.

Listado 17.20: Carga y visualización de imágenes.

```
Image *img = image_from_file("lenna.jpg", NULL);
Image *icon = image_from_resource(pack, ekCANCEL);
...
imageview_image(view, img);
button_image(button, icon);
```

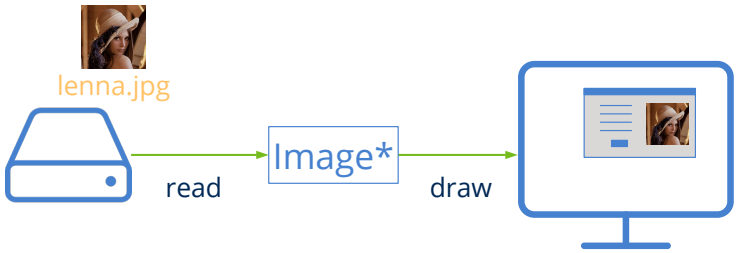


Figura 17.41: Uso elemental de imágenes en la interfaz de usuario.

- Utiliza `image_from_file` para cargar una imagen de disco.
- Utiliza `image_from_data` para crear una imagen desde un búfer de memoria.
- Utiliza `image_from_resource` para obtener una imagen de un paquete de recursos.
- Utiliza `image_read` para crear una imagen a partir de *"Streams"* (Página 197).
- En la demo *UrlImg*[5] tienes un ejemplo de como descargarlas desde un servidor Web.

Una vez cargado el objeto imagen en memoria, disponemos de varias formas de visualizarlo:

- Utiliza `draw_image` para dibujar una imagen en un contexto 2d.
- Utiliza `imageview_image` para asignar una imagen a una vista. Este control soporta animaciones GIF.
- Utiliza `button_image` para asignar una imagen a un botón.
- Utiliza `popup_add_elem` para asignar un texto e icono a una lista desplegable.

[5] `https://nappgui.com/es/howto/urlimg.html`

17.8.2. Generar imágenes

Como ya vimos en *"Contextos 2D"* (Página 263), si fuera necesario podemos crear nuestras propias imágenes a partir de comandos de dibujo para, posteriormente, mostrarlas en la interfaz (Figura 17.42) o guardarlas en disco.

- Utiliza `dctx_image` para crear una imagen a partir de un contexto 2d.

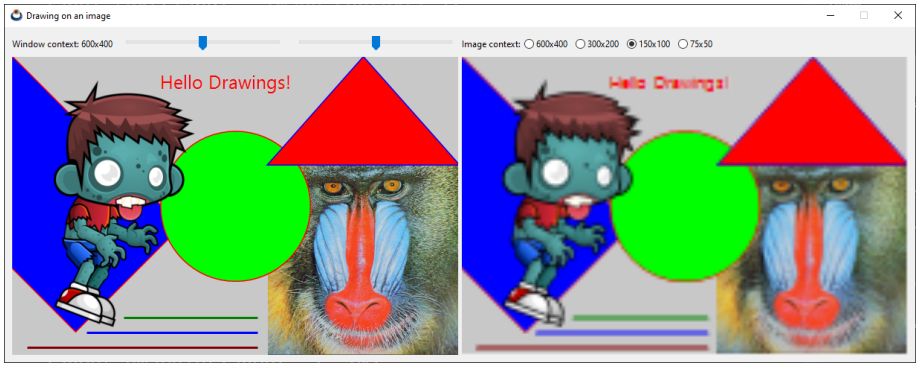


Figura 17.42: Imagen generada mediante comandos de dibujo (a la derecha).

17.8.3. Acceso a píxeles

Las imágenes son **objetos inmutables** optimizados para el dibujo recurrente en pantalla, por lo que se permiten ciertas licencias, tanto en la organización interna de la información de color como en la gestión de posibles copias. Por este motivo **no es posible manipular directamente los píxeles**, sino que debemos acceder a ellos mediante un *"Pixel Buffer"* (Página 286).

- Utiliza `image_from_pixels` para crear una imagen a partir de puntos.
- Utiliza `image_from_pixbuf` para crear una imagen a partir de un pixel búfer.
- Utiliza `image_pixels` para obtener un búfer con los píxeles de la imagen.
- Utiliza `image_width` para obtener la anchura.
- Utiliza `image_height` para obtener la altura.
- Utiliza `image_format` para obtener el formato de píxel.

Documentación técnica de Apple: "Treat NSImage and its image representations as immutable objects. The goal of NSImage is to provide an efficient way to display images on the target canvas. Avoid manipulating the data of an image representation directly, especially if there are alternatives to manipulating the data, such as compositing the image and some other content into a new image object."

Los **pixel buffers** nos permiten de una forma óptima manipular el contenido de la imagen. Para visualizar el resultado o almacenarlo en cualquiera de los formatos soportados, deberemos crear una nueva imagen (Figura 17.43).

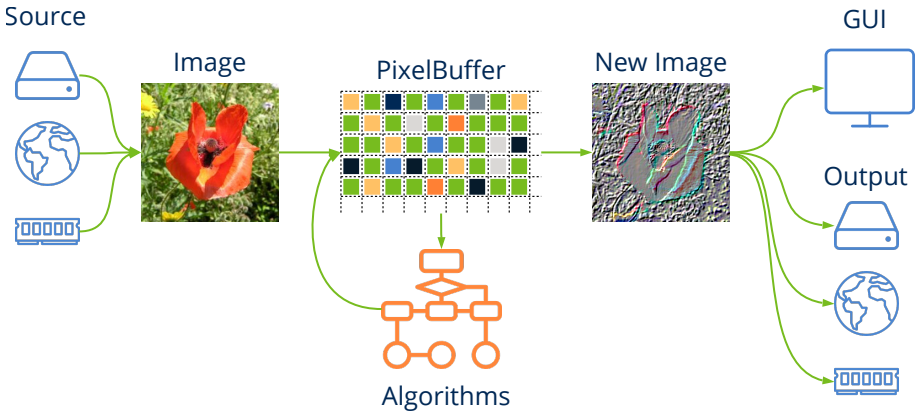


Figura 17.43: Proceso de edición de imágenes.

17.8.4. Guardar imágenes: Codecs

Uno de los mayores problemas que presentan las imágenes digitales es la gran cantidad de memoria que ocupan. Una imagen de tan solo 1024x768 píxeles y 32bits de color necesita 3 megabytes de memoria. Puede que no parezca mucho, pero a finales de los años 80 esto suponía un gran handicap ya que la memoria era muy cara y las transmisiones muy lentas. Por esto se idearon varios sistemas de codificación (compresión) que reducían la cantidad de memoria necesaria y que se consolidaron con el auge de Internet (Figura 17.44).

- Utiliza `image_get_codec` para obtener el *codec* asociado a la imagen.
- Utiliza `image_codec` para cambiar el *codec* asociado a la imagen.
- Utiliza `image_to_file` para guardarla en disco.
- Utiliza `image_write` para escribirla en un `Stream`.

Figura 17.44: Formatos de imagen soportados por NAppGUI.

Draw2D no soporta de forma nativa otros formatos diferentes a los mencionados. En caso necesario, deberás encontrar la forma de crear un `Pixbuf` a partir de los datos específicos de tu formato, con el fin de integrar dichas imágenes en la interfaz de usuario.

- **JPEG:** *Joint Photographic Experts Group* es un formato con una muy buena tasa de compresión basada en la Transformada de Fourier. Ideal para captar instantáneas del mundo real, aunque restará algo de calidad a la captura original (compresión con pérdida).
- **PNG:** *Portable Network Graphics* surgió como respuesta a los problemas legales con el formato GIF. Soporta compresión LZ77/Deflate sin pérdida y formatos de píxel indexado. Ideal para esquemas, gráficos o imágenes generadas por computador.
- **GIF:** *Graphics Interchange Format*. Utiliza el algoritmo de compresión propietario LZW, aunque la patente expiró en 2003. Ha sobrevivido a PNG gracias a que puede incluir animaciones en un solo archivo, algo que ninguno de los dos formatos anteriores soporta.
- **BMP:** *BitMaP*. Formato nativo de Windows ampliamente superado por los otros tres. Aunque soporta un tipo especial de compresión denominado *Run-Length encoding*, lo cierto es que la mayoría de archivos se guardan sin comprimir. Los archivos BMP ocupan mucho más espacio, por este motivo se utiliza muy poco en Internet y casi nada en máquinas ajenas a Windows. Es soportado por casi todos los programas y sistemas gracias a que es muy sencillo y rápido de interpretar.

Para poder visualizarse en pantalla, la imagen debe ser descomprimida (de-codificada), proceso que se realiza automáticamente al leer la imagen. Al guardarla a disco o enviarla por la red se realiza el proceso contrario, se comprime o codifica utilizando el algoritmo asociado a la misma (Tabla 17.5), pero se puede cambiar.

Constructor	Codec
`image_from_file`	El codec original.
`image_from_data`	El codec original.
`image_from_resource`	El codec original.
`image_from_pixels`	¿Transparencias? Sí:`ekPNG` No:`ekJPG`.
`dctx_image`	`ekPNG`.

Tabla 17.5: Codecs de imagen por defecto.

Por lo general, el soporte que brindan GDI+, NSImage o GdkPixbuf para la configuración de los codecs es bastante limitado. Por ejemplo, no es posible generar archivos PNG indexados, algo muy útil a la hora de reducir el tamaño de las imágenes para la Web. Si la aplicación requiriera mayor control sobre la exportación, no tendremos más remedio que utilizar libpng, libjpeg o cualquier otra solución de terceros.

17.9. Fuentes tipográficas

Las fuentes tipográficas son objetos gráficos (archivos) que contienen los caracteres y símbolos que vemos en un monitor. Recordamos que una cadena *"Unicode"* (Página 157) únicamente almacena el código de los caracteres *(codepoints)* sin ningún tipo de información sobre como deben ser dibujados. Se conoce como **glifo** al gráfico asociado a un carácter y, en un fichero de fuente, hay tantos glifos como *codepoints* pueda representar la tipografía. El emparejamiento entre *codepoints* y sus correspondientes glifos lo lleva a cabo el subsistema gráfico del sistema operativo (Listado 17.21) (Figura 17.45).

Listado 17.21: Dibujo de una cadena de texto.

```
Font *font = font_create("Comic Sans MS" 28, 0);
draw_font(ctx, font);
draw_text(ctx, "Hello World", 200, 250);
font_destroy(&font);
```

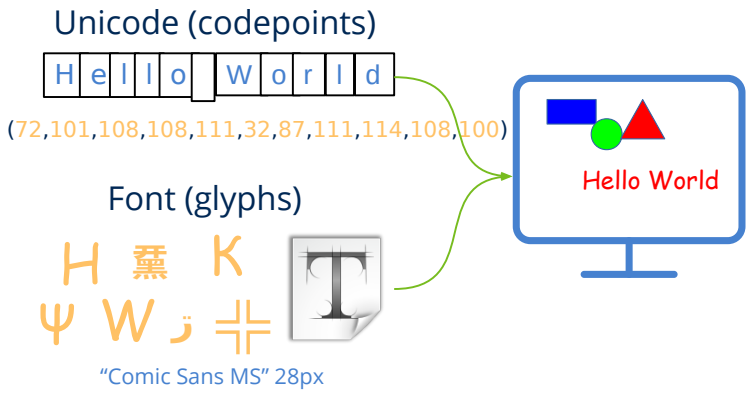


Figura 17.45: Representación de textos: *codepoints* + glifos.

17.9.1. Crear fuentes

A la hora de mostrar textos en interfaces gráficas es necesario establecer una tipografía, de lo contrario el sistema no sabría como renderizarla. Siempre existirá alguna fuente definida por defecto, pero podemos cambiarla a la hora de personalizar la apariencia de nuestros textos.

- Utiliza `font_create` para crear una nueva fuente.
- Utiliza `font_family` para obtener el tipo de letra de la fuente.
- Utiliza `draw_font` para establecer la fuente en contextos 2D.
- Utiliza `label_font` para cambiar la fuente asociada a un control `Label`.

La característica más representativa del diseño de una tipografía es la familia a la que pertenece (*font family* o *typeface*) (Figura 17.46). Cada computador tiene instaladas una serie de familias que no tienen porqué coincidir con las incorporadas en otra máquina. Este es un hecho importante a tener en cuenta ya que, en pro de la portabilidad, no debemos asumir que una determinada familia tipográfica vaya a estar presente en todas las máquinas que ejecuten el programa. Sentencias del tipo:

```
Font *font = font_create("Comic Sans MS", 28, 0);
```

no serán del todo portables, ya que no tenemos la certeza de que la tipografía *Comic Sans MS* esté instalada en todos los ordenadores. Tenemos dos alternativas para garantizar la existencia de una determinada fuente:

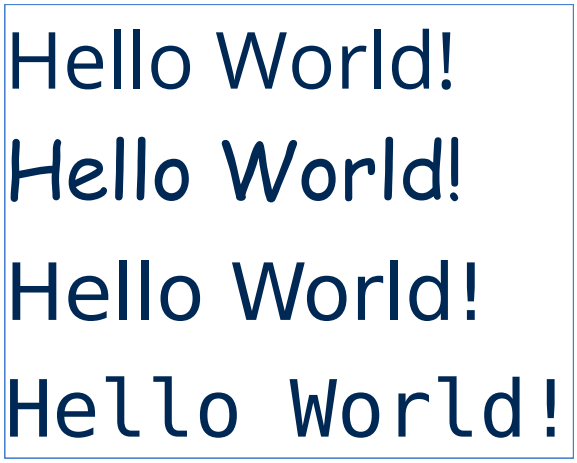


Figura 17.46: Diferentes familias tipográficas.

- Utiliza `font_system` para obtener la fuente por defecto del sistema operativo. Siempre estará disponible pero su apariencia será diferente según el sistema operativo.
- Utiliza `font_regular_size` para obtener el tamaño predefinido para botones y otros controles.
- Utiliza `font_installed_families` para obtener la lista de familias instaladas en la máquina y elegir la que más se adapte a nuestros propósitos.

17.9.2. Fuente del sistema

Como acabamos de mencionar, siempre hay una fuente por defecto asociada al entorno de ventanas y que, en cierta manera, le otorga parte de su personalidad. Utilizar esta fuente nos garantiza la correcta integración de nuestro programa en todos los sistemas donde se ejecute, haciendo nuestro código totalmente portable (Figura 17.47). Los controles de interfaz como `Button` o `Label` tienen asociada por defecto la fuente del sistema en tamaño regular. La correspondencia de *font_system* en las diferentes plataformas es:

- **Segoe UI**: Windows Vista, 7, 8, 10.
- **Tahoma**: Windows XP.
- **San Francisco**: macOS Mojave, High Sierra, Sierra, Mac OSX El Capitan.
- **Helvetica Neue**: Mac OSX Yosemite.
- **Lucida Grande**: Mac OSX Mavericks, Mountain Lion, Lion, Snow Leopard.
- **Ubuntu**: Linux Ubuntu.
- **Piboto**: Linux Raspbian.

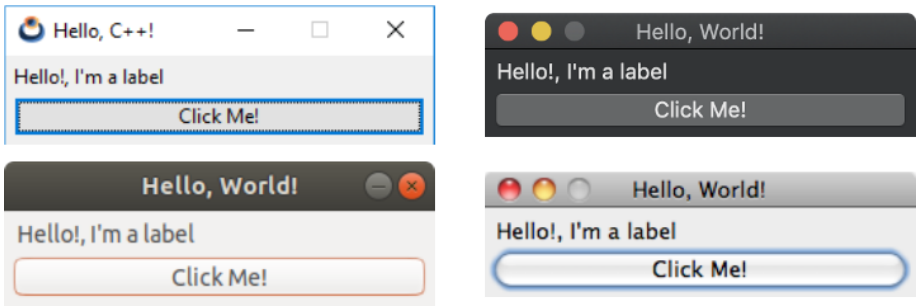


Figura 17.47: Uso de la fuente del sistema.

Además de la fuente del sistema tenemos disponible otra fuente **monoespacio** por defecto (Figura 17.48). Estas tipografías imitan a las máquinas de escribir antiguas, donde todos los caracteres ocupan el mismo espacio. Suelen utilizarse para documentos técnicos o archivos de código fuente.

- Utiliza `font_monospace` para crear una tipografía monoespacio genérica.

Figura 17.48: Fuente proporcional (ancho variable) y monoespacio (ancho fijo).

17.9.3. Características de las fuentes

Además de la familia, podremos ajustar el tamaño y el estilo de la fuente. El tamaño hace referencia a la altura promedio (en píxeles) de los caracteres que componen la tipográfica, donde no se tiene en cuenta los márgenes ni los desplazamientos en relación al *baseline* (Figura 17.49). A la **altura total de una línea de texto** se la conoce como *cell height* y, por norma general, será algo superior al tamaño de la fuente *char height*.

Figura 17.49: Altura del carácter *(char height = font size)*.

También podemos cambiar el estilo del texto, estableciendo sus atributos a través del parámetro `style` combinando los valores de `fstyle_t` (Figura 17.50).

- `ekFBOLD`. Negrita.
- `ekFITALIC`. Itálica.
- `ekFUNDERLINE`. Subrayado.
- `ekFSTRIKEOUT`. Tachado.

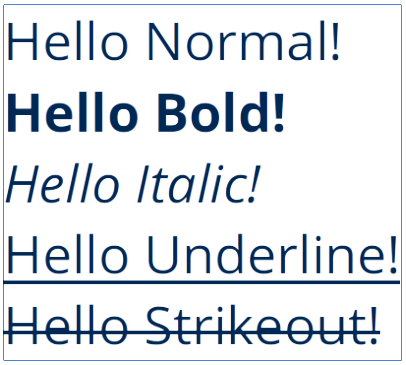


Figura 17.50: Estilo del texto.

17.9.4. Tamaño en puntos

Por defecto, el tamaño de la fuente se expresa en píxeles, pero puede cambiarse añadiendo `ekFPOINTS` al parámetro `style`. Esta unidad está relacionada con las fuentes impresas en papel. Aquí aparece el concepto DPI (*dots per inch*) que indica la cantidad de gotas de tinta aisladas que un dispositivo de impresión puede emitir por pulgada métrica. En tipografía se establece el criterio de 72 DPI's, por tanto, el tamaño de un punto es 0.35mm aproximadamente. De esta forma es sencillo calcular el tamaño de letra a partir de los puntos: 12pt=4.2mm, 36pt=12.7mm ó 72pt=25.4mm (1 pulgada). Esta es la unidad utilizada en **procesadores de texto**, que ya estos trabajan en base a un tamaño de página de impresión. El problema llega cuando queremos representar fuentes expresadas en puntos

en una pantalla, ya que no existe una correspondencia exacta entre píxeles y milímetros. El tamaño final del pixel depende de la resolución y del tamaño físico del monitor. Se precisa un convenio de conversión entre píxeles y pulgadas, lo que da lugar al término PPI (*pixels per inch*). Tradicionalmente, en sistemas Windows se establece 96 PPI mientras que en los iMac de Apple es de 72 PPI. Esto provoca que las fuentes expresadas en puntos sean un 33 % más grandes en Windows (Figura 17.51). Además en el sistema de Microsoft es posible configurar el PPI por el usuario, lo que añade más incertidumbre sobre el tamaño final de los textos en pantalla.

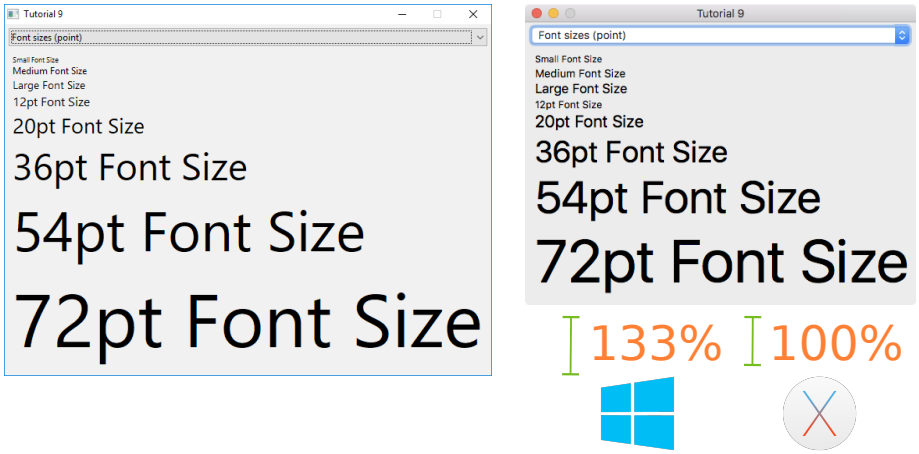


Figura 17.51: La unidad `ekFPOINTS` no es aconsejable para pantallas.

17.9.5. Fuentes Bitmap y Outline

En los primeros computadores las tipografías se creaban como gráficos de raster *Bitmap Fonts* (Figura 17.52). Cada carácter se ajustaba a una celda de tamaño fijo donde se marcaban aquellos píxeles que lo componían. El mayor problema es que no escalan bien. A medida que hacemos más grande el texto en pantalla más se hace patente el efecto dentado de los píxeles.

Figura 17.52: Fuentes bitmap.

En 1982 Adobe lanza el formato PostScript que incluía las conocidas como *Outline Fonts* (Figura 17.53). Este formato contiene una descripción geométrica de cada símbolo basada en líneas y curvas de Bezier. De esta forma se evita el efecto pixelado de las bitmap, ya que al escalar el carácter se vuelven a computar los píxeles que lo componen en un proceso conocido como **rasterización**. A finales de los 80's Apple lanza el formato *TrueType* y le vende una licencia a Microsoft que lo incorpora en Windows 3.1, abriendo la puerta del mercado masivo a las fuentes vectoriales. Hoy en día todos los sistemas trabajan con fuentes escalables, teniendo en *TrueType* y *OpenType* los más claros representantes.

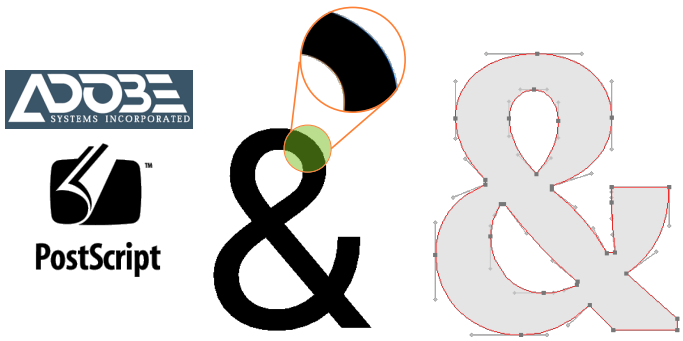


Figura 17.53: Fuentes outline, en las que se basan los formatos *TrueType* y *OpenType*.

17.9.6. Unicode y glifos

Unicode es una tabla muy extensa. En la versión 11 (Junio 2018) hay registrados 137.374 *codepoints* y este número va creciendo en cada nueva revisión del estándar. Si la aplicación necesita de símbolos especiales (por encima del BMP-*Basic Multilingual Plane)* debemos cerciorarnos que las tipografías seleccionadas contengan glifos para ellos. Para ver la relación entre codepoints y glifos podemos utilizar la aplicación BabelMap (Figura 17.54), y dentro de ella la opción Font Analysis. A partir de un bloque Unicode mostrará aquellas fuentes instaladas que incluyen glifos para dicho rango. En macOS tenemos una aplicación similar llamada *Character Viewer* y en Ubuntu otra denominada *Character Map*.

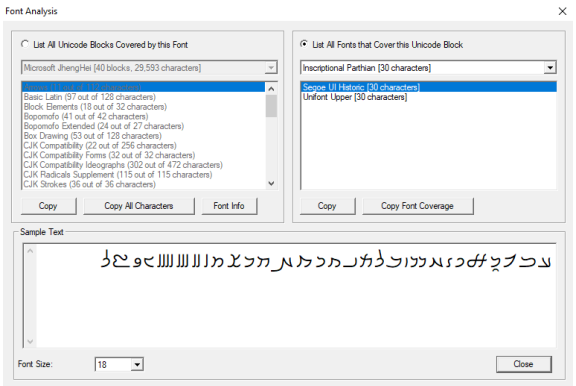


Figura 17.54: BabelMap Font Analysis nos proporciona información sobre los glifos incluidos en cada tipografia.

Capítulo 18

Librería Gui

18.1. Gui

La librería *Gui* permite crear interfaces gráficas de usuario de forma sencilla e intuitiva. Es el primer paquete especializado en aplicaciones de escritorio (Figura 18.1), al contrario de lo que ocurre con el resto de librerías que pueden utilizarse también en aplicaciones por línea de comandos.

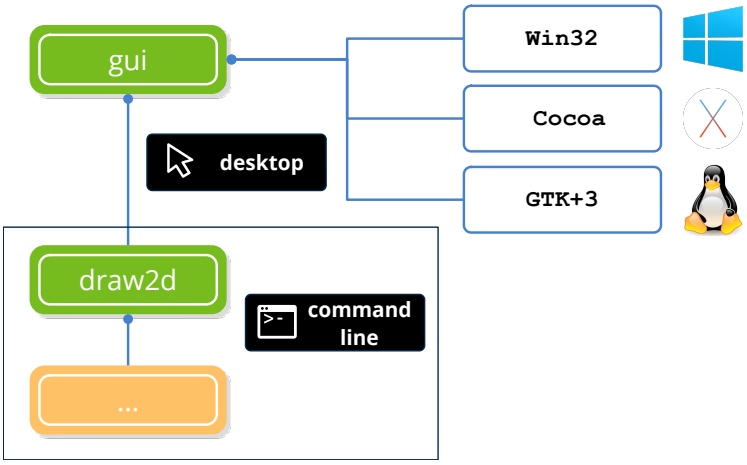


Figura 18.1: Dependencias de *Gui*. Ver *"NAppGUI API"* (Página 147).

Al igual que *"Draw2D"* (Página 262) y *"Osbs"* (Página 168) *Gui* se apoya en las APIs de cada sistema operativo. Además de las ventajas ya comentadas en estos dos casos, el acceso nativo provocará que nuestros programas estén totalmente integrados en el entorno de ventanas y acordes al tema visual presente en cada máquina (Figura 18.2).

18.1.1. Composición declarativa

La librería *Gui* se aleja del concepto de tratar las ventanas (o cuadros de diálogo) como un recurso externo del programa. Por el contrario, estas se crean directamente desde el código fuente evitando maquetarlas mediante editores visuales (Figura 18.3). Debemos tener en cuenta que cada plataforma utiliza diferentes tipografías y *skins*, por lo que especificar posiciones y tamaños concretos para los controles no será del todo portable (Figura 18.4). Por el contrario, en *Gui* los elementos de interfaz se ubican en una rejilla virtual denominada `Layout`, que los dimensionará en tiempo de ejecución en función de la máquina donde esté corriendo el programa (Figura 18.5).

Además, otro hecho relevante es que las interfaces son objetos vivos sujetos a cambios constantes. Un claro ejemplo son las traducciones, que alteran la ubicación de los elementos debido a la nueva dimensión del texto (Figura 18.6). *Gui* se adaptará a estos eventos de

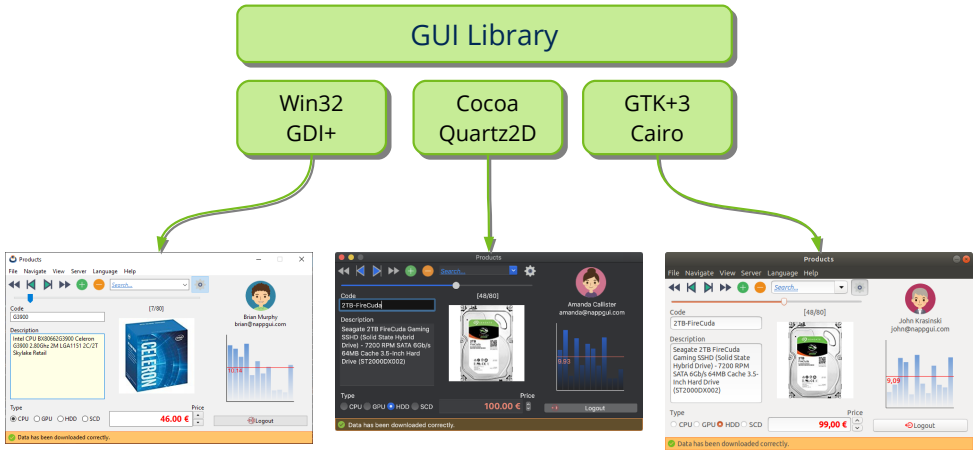


Figura 18.2: Las interfaces creadas con *Gui* se adaptarán al estilo de cada entorno de ventanas.

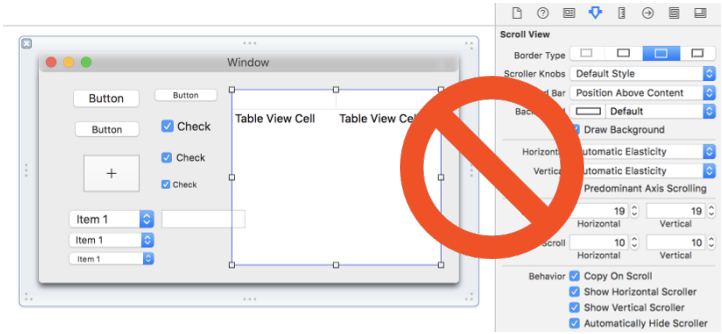


Figura 18.3: Los editores de recursos no son buenos aliados para crear complejas interfaces dinámicas. Menos aún si queremos portarlas entre plataformas.

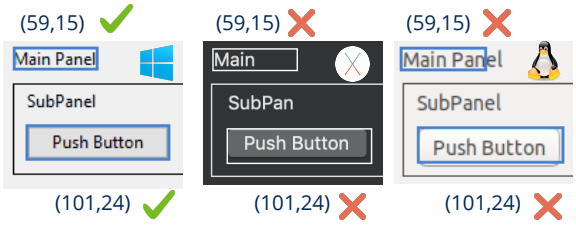


Figura 18.4: Utilizar medidas fijas para los controles no se adaptará bien al migrar el programa.

forma automática, recalculando posiciones y tamaños para mantener una maquetación coherente.

18.1.2. Anatomía de una ventana.

En (Figura 18.7) tenemos las partes principales de una ventana. Los **controles** son los elementos finales con los que interactúa el usuario para introducir datos o lanzar acciones. Las **vistas** son regiones rectangulares de tamaño relativamente grande donde se representa información mediante texto y gráficos, pudiendo responder a los eventos de teclado o ratón.

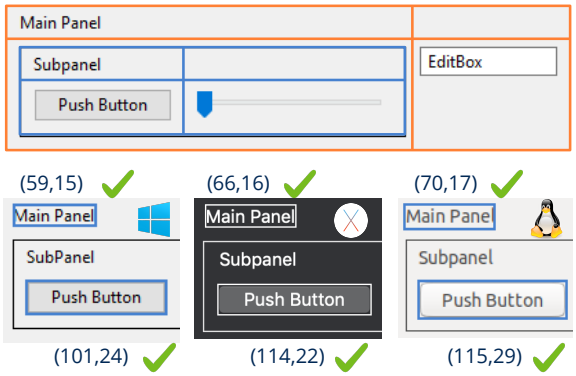


Figura 18.5: El Layout calcula la posición y tamaño de los componentes en tiempo de ejecución.

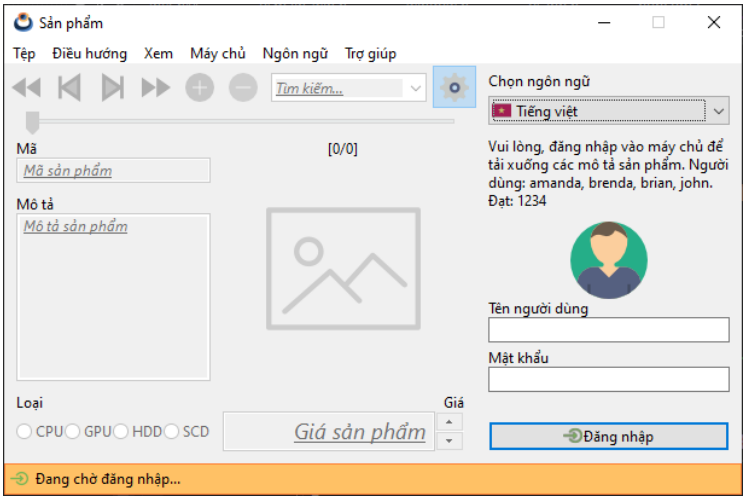


Figura 18.6: Las ventanas se adaptan automáticamente a cambios en tiempo de ejecución. Animación en *https://nappgui.com/img/gui/translate.gif*.

Por último, todos estos elementos se agruparán en **paneles** y se maquetarán mediante **layouts**.

- *"Label"* (Página 309). Pequeños bloques de texto descriptivo.
- *"Button"* (Página 310). Botones de pulsación, casillas de verificación o radio.
- *"PopUp"* (Página 313). Botón con lista desplegable.
- *"Edit"* (Página 313). Cuadro de edición de texto.
- *"Combo"* (Página 315). Cuadro de edición con lista desplegable.
- *"ListBox"* (Página 316). Cuadro de lista.
- *"UpDown"* (Página 317). Botones de incremento y decremento.

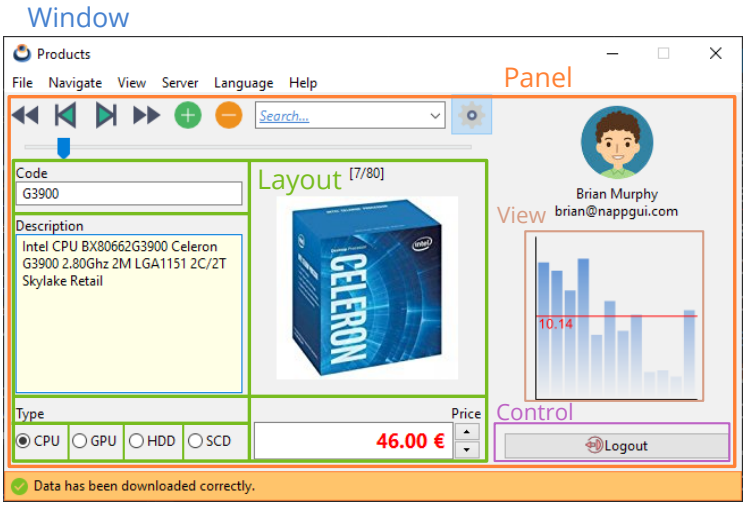


Figura 18.7: Partes destacables en una ventana de interfaz.

- *"Slider"* (Página 317). Barra deslizadora.
- *"Progress"* (Página 318). Barra de progreso.
- *"View"* (Página 319). Vista genérica donde se puede dibujar libremente.
- *"TextView"* (Página 323). Vista para mostrar y editar textos, en múltiples formatos.
- *"ImageView"* (Página 325). Vista para mostrar imágenes.
- *"TableView"* (Página 326). Vista para mostrar información en filas y columnas.
- *"SplitView"* (Página 332). Vista dividida en dos partes redimensionables.
- *"Layout"* (Página 335). Rejilla virtual e invisible donde se ubicarán los controles.
- *"Panel"* (Página 344). Sub-ventana dentro de la principal con sus propios controles.
- *"Window"* (Página 350). Ventana principal con barra de título y marco.
- *"Menu"* (Página 366). Lista desplegable con opciones.
- *"MenuItem"* (Página 367). Cada uno de los elementos del menú.

18.1.3. Eventos GUI

Las aplicaciones de escritorio están dirigidas por eventos, lo que significa que están continuamente esperando a que el usuario realice alguna acción sobre la interfaz: Pulsar un botón, arrastrar un *slider*, escribir un texto, etc. Cuando esto ocurre, el gestor de ventanas

detecta el evento y lo notifica a la aplicación (Figura 18.8), la cual debe proveer un **manejador del evento** con el código a ejecutar. Por ejemplo, en (Listado 18.1) definimos un manejador para responder a la pulsación de un botón. Evidentemente, si no hay manejador asociado, la aplicación ignorará el evento.

- Utiliza `event_params` para obtener los parámetros asociados al evento. Cada tipo de evento tiene sus propios parámetros. Ver (Tabla 18.1).
- Utiliza `event_result` para escribir la respuesta al evento. Muy pocos eventos requieren enviar una respuesta.

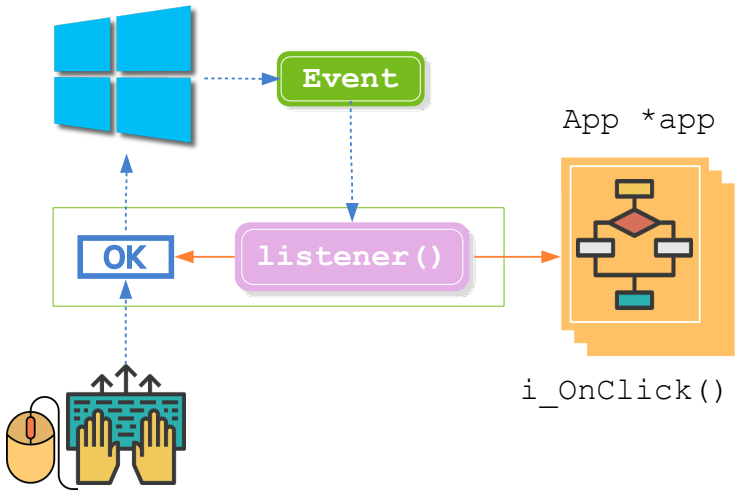


Figura 18.8: Notificación de un evento a través del manejador.

Listado 18.1: Asignar un manejador para la pulsación de un botón.

```
static void i_OnClick(App *app, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    if (p->state == ekGUI_ON)
        create_new_file(app);
}

Button *button = button_check();
button_OnClick(button, listener(app, i_OnClick, App));
```

Evento	Manejador	Parámetros	Respuesta
Clic en label	`label_OnClick`	`EvText`	-
Clic en botón	`button_OnClick`	`EvButton`	-

Evento	Manejador	Parámetros	Respuesta
Selección en PopUp	`popup_OnSelect`	`EvButton`	-
Selección en ListBox	`listbox_OnSelect`	`EvButton`	-
Pulsación de tecla en Edit	`edit_OnFilter`	`EvText`	`EvTextFilter`
Fin de edición en Edit	`edit_OnChange`	`EvText`	-
Pulsación de tecla en Combo	`combo_OnFilter`	`EvText`	`EvTextFilter`
Fin de edición en Combo	`combo_OnChange`	`EvText`	-
Movimiento de un Slider	`slider_OnMoved`	`EvSlider`	-
Clic en UpDown	`updown_OnClick`	`EvButton`	-
Dibujar el contenido de una vista	`view_OnDraw`	`EvDraw`	-
El tamaño de una vista ha cambiado	`view_OnSize`	`EvSize`	-
El ratón entra en el área de una vista	`view_OnEnter`	`EvMouse`	-
El ratón sale del área de una vista	`view_OnExit`	-	-
El ratón se mueve sobre una vista	`view_OnMove`	`EvMouse`	-
Se ha pulsado un botón del ratón	`view_OnDown`	`EvMouse`	-
Se ha liberado un botón del ratón	`view_OnUp`	`EvMouse`	-
Clic sobre una vista	`view_OnClick`	`EvMouse`	-
Arrastrando sobre una vista	`view_OnDrag`	`EvMouse`	-
Ruedecilla del ratón sobre una vista	`view_OnWheel`	`EvWheel`	-
Pulsar tecla sobre una vista	`view_OnKeyDown`	`EvKey`	-
Liberar tecla sobre una vista	`view_OnKeyUp`	`EvKey`	-
Vista ha recibido el foco del teclado	`view_OnFocus`	`bool_t`	-
Cierre de una ventana	`window_OnClose`	`EvWinClose`	`bool_t`
Ventana moviéndose	`window_OnMoved`	`EvPos`	-
Ventana se está re-dimensinando	`window_OnResize`	`EvSize`	-
Clic sobre un menú item	`menuitem_OnClick`	`EvMenu`	-
Cambio de color	`comwin_color`	`color_t`	-

Tabla 18.1: Lista de todos los eventos de interfaz.

18.2. Label

Los controles **Label** sirven para insertar pequeños bloques de texto en ventanas y formularios. Son de formato uniforme, es decir, los atributos de fuente y color se aplicarán a la totalidad del texto. En la mayoría de ocasiones el contenido estará limitado a una sola línea, aunque es posible mostrar bloques que se extiendan en varias de ellas. El tamaño del control se ajustará al del texto que contiene (Figura 18.9). En *"¡Hola Label!¡Hola Label!"* (Página 497) tienes el código de ejemplo.

- Utiliza `label_create` para crear un control de texto.
- Utiliza `label_multiline` para crear un control multilínea.
- Utiliza `label_align` para establecer la alineación interna del texto.
- Utiliza `label_font` para establecer la fuente.

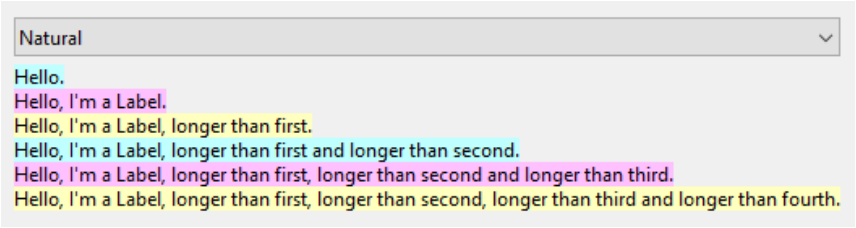


Figura 18.9: Controles Label.

En el caso que la columna del `Layout` tenga un ancho menor que el del propio texto, se mostrarán unos puntos (elípsis) en el punto de corte (Figura 18.10), salvo en labels multilínea, que se expandirán verticalmente para dar cabida a todo el texto (Figura 18.11).

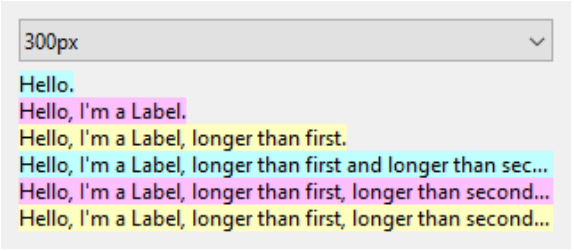


Figura 18.10: Ajuste del texto al reducir el ancho del control.

En (Figura 18.12) tenemos un ejemplo del uso de *Label* en formularios. Si fuera necesario, podemos hacer que los textos sean sensibles al ratón variando su estilo y colores (Figura 18.13).

- Utiliza `label_style_over` para cambiar el estilo de la fuente.
- Utiliza `label_color_over` para cambiar color del texto.
- Utiliza `label_bgcolor_over` para cambiar color del fondo.

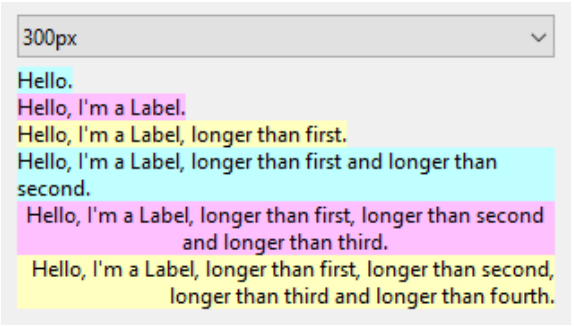


Figura 18.11: Los Labels multilínea se expandirán verticalmente para dar cabida a todo el texto.

- Utiliza `label_OnClick` para responder a un clic en el texto.

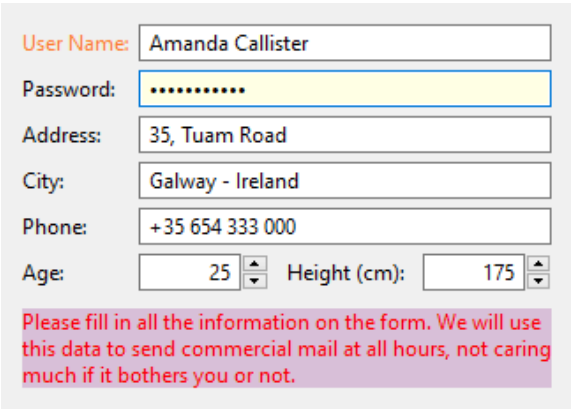


Figura 18.12: Uso de *Label* simples y multilínea en formularios.

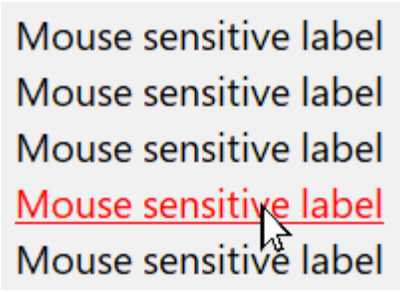


Figura 18.13: Controles *Label* sensibles al ratón.

18.3. Button

Los botones son otro elemento clásico en interfaces gráficas, donde distinguimos cuatro tipos: El botón de pulsación, casilla de verificación *(checkbox)*, botón de radio *radiobutton* y botón plano típico de las barras de herramientas (Figura 18.14). En *"¡Hola Button!¡Hola Button!"* (Página 502) tienes un ejemplo de uso.

- Utiliza `button_push` para crear un botón de pulsación.

- Utiliza `button_check` para crear una casilla de verificación.
- Utiliza `button_check3` para crear una casilla con tres estados.
- Utiliza `button_radio` para crear un botón de radio.
- Utiliza `button_flat` para crear un botón plano.
- Utiliza `button_flatgle` para crear un botón plano con estado.
- Utiliza `button_text` para asignar un texto.
- Utiliza `button_OnClick` para responder a pulsaciones.

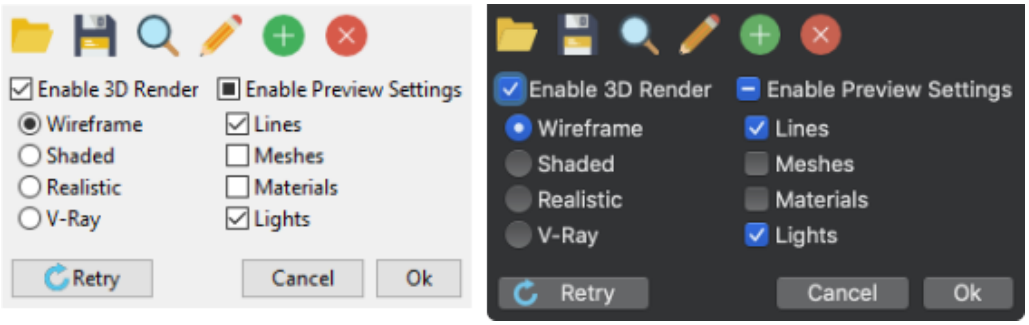


Figura 18.14: Botones en diferentes plataformas.

Además de capturar el evento y notificar a la aplicación, los *checkbox* y *flatgle* mantienen un estado (pulsado/check o liberado/uncheck).

- Utiliza `button_state` para establecer el estado del botón.
- Utiliza `button_get_state` para obtener el estado del botón.

18.3.1. RadioGroup

Mención especial requieren los botones de radio, que solo tienen sentido cuando aparecen en grupo ya que se utilizan para seleccionar una única opción dentro de un conjunto. Los grupos se forman a nivel de `Layout`, es decir, todos los *radiobutton* de un mismo layout se considerarán del mismo grupo, donde solo uno de ellos podrá estar seleccionado. Si necesitamos varios sub-grupos, deberemos crear varios sub-layout, como se muestra en (Figura 18.15) (Listado 18.2). A la hora de capturar el evento, el campo `index` de `EvButton` indicará el índice del botón que ha sido pulsado.

Listado 18.2: Grupos de botones de radio.

```
Button *button1 = button_radio();
Button *button2 = button_radio();
Button *button3 = button_radio();
Button *button4 = button_radio();
```

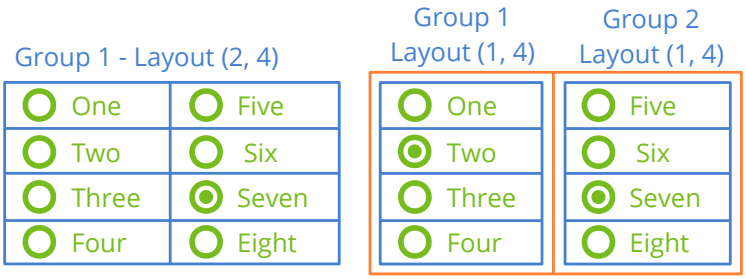


Figura 18.15: Radio grupos vinculados a diferentes layouts.

```
Button *button5 = button_radio();
Button *button6 = button_radio();
Button *button7 = button_radio();
Button *button8 = button_radio();
button_text(button1, "One");
button_text(button2, "Two");
button_text(button3, "Three");
button_text(button4, "Four");
button_text(button5, "Five");
button_text(button6, "Six");
button_text(button7, "Seven");
button_text(button8, "Eight");

// One group - One layout
Layout *layout = layout_create(2, 4);
layout_button(layout, button1, 0, 0);
layout_button(layout, button2, 0, 1);
layout_button(layout, button3, 0, 2);
layout_button(layout, button4, 0, 3);
layout_button(layout, button5, 1, 0);
layout_button(layout, button6, 1, 1);
layout_button(layout, button7, 1, 2);
layout_button(layout, button8, 1, 3);

// Two groups - Two sub-layouts
Layout *layout1 = layout_create(2, 1);
Layout *layout2 = layout_create(1, 4);
Layout *layout3 = layout_create(1, 4);
layout_button(layout2, button1, 0, 0);
layout_button(layout2, button2, 0, 1);
layout_button(layout2, button3, 0, 2);
layout_button(layout2, button4, 0, 3);
layout_button(layout3, button5, 0, 0);
layout_button(layout3, button6, 0, 1);
layout_button(layout3, button7, 0, 2);
layout_button(layout3, button8, 0, 3);
layout_layout(layout, layout1, 0, 0);
layout_layout(layout, layout2, 1, 0);
```

18.4. PopUp

Los **PopUps** son botones que tienen asociado un menú desplegable (Figura 18.16). Aparentemente parecen *pushbuttons* que al pulsarse muestran una lista de opciones. En *"¡Hola PopUp y Combo!¡Hola PopUp y Combo!"* (Página 505) tienes un ejemplo de uso.

- Utiliza `popup_create` parar crear un popup.
- Utiliza `popup_add_elem` para añadir un elemento a la lista.
- Utiliza `popup_OnSelect` para responder a la selección.

Figura 18.16: PopUps en Windows, macOS y Linux.

18.5. Edit

Los **EditBox** son pequeñas cajas de texto con capacidades de edición. Al igual que los `Label` son de formato uniforme: La tipografía y colores afectarán a la totalidad del texto (Figura 18.17). Suelen utilizarse para editar campos en formularios, normalmente restringidos a una sola línea, aunque también pueden extenderse a varias de ellas. Para editar textos con múltiples atributos utilizar `TextView`. En *"¡Hola Edit y UpDown!¡Hola Edit y UpDown!"* (Página 507) tienes un ejemplo de uso.

- Utiliza `edit_create` para crear una caja de edición.
- Utiliza `edit_multiline` para crear una caja de edición multilínea.
- Utiliza `edit_passmode` para ocultar el texto del control.
- Utiliza `edit_phtext` para establecer un *placeholder*.
- Utiliza `edit_autoselect` para seleccionar automáticamente todo el texto.

18.5.1. Filtrar textos

En función del valor que estemos editando, es posible que sea necesario validar el texto introducido. Esto podemos hacerlo al terminar de editar o mientras estamos escribiendo.

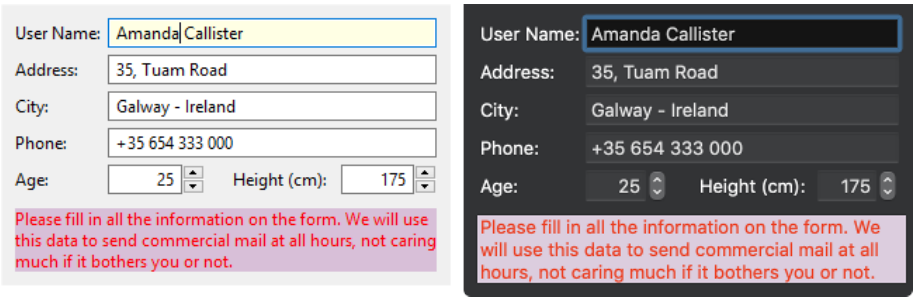


Figura 18.17: Cajas de edición en diferentes plataformas.

Para el primer caso utilizaremos el evento `edit_OnChange` que llamará al manejador cuando el control haya perdido el foco del teclado (Figura 18.18). Si queremos implementar filtros más elaborados, que corrijan el texto mientras se escribe utilizaremos el evento `edit_OnFilter`. Por ejemplo, en (Listado 18.3) tenemos un sencillo filtro que solo permite carácteres numéricos (Figura 18.19).

- Utiliza `edit_OnChange` para validar el texto final.
- Utiliza `edit_OnFilter` para detectar y corregir cada pulsación del usuario.

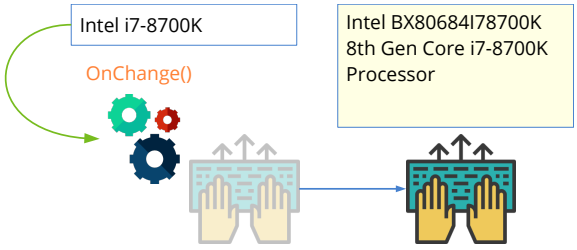


Figura 18.18: El evento *OnChange* se llama cuando el control pierde el foco.

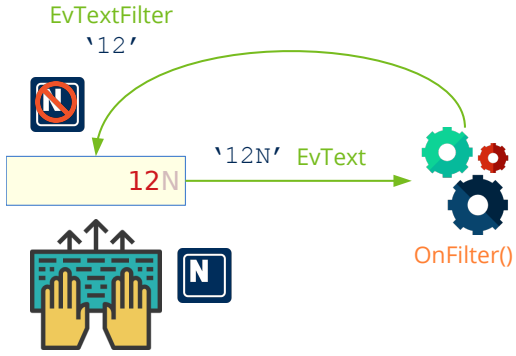


Figura 18.19: El evento *OnFilter* se llama tras cada pulsación de tecla.

Listado 18.3: Filtro que solo permite carácteres numéricos.

```
static void OnFilter(void *noused, Event *e)
{
    const EvText *params = event_params(e, EvText);
```

```
    EvTextFilter *result = event_result(e, EvTextFilter);
    uint32_t i = 0, j = 0;
    while (params->text[i] != '\0')
    {
        if (params->text[i] >= '0' && params->text[i] <= '9')
        {
            result->text[j] = params->text[i];
            j += 1;
        }

        i += 1;
    }

    result->text[j] = '\0';
    result->apply = TRUE;
}
...
edit_OnFilter(edit1, listener(NULL, i_OnFilter, void));
```

18.6. Combo

Los **ComboBox** son cajas de edición de texto con lista desplegable (Figura 18.20). Por tanto, funcionarán de la misma forma que los controles `Edit` sobre los que se añaden métodos para la gestión de la lista. En *"¡Hola PopUp y Combo!¡Hola PopUp y Combo!"* (Página 505) tienes un ejemplo de uso.

- Utiliza `combo_create` parar crear un combo.
- Utiliza `combo_text` para establecer el texto de edición.
- Utiliza `combo_color` para establecer el color del texto.
- Utiliza `combo_bgcolor` para establecer el color del fondo.
- Utiliza `combo_add_elem` para añadir un elemento a la lista.

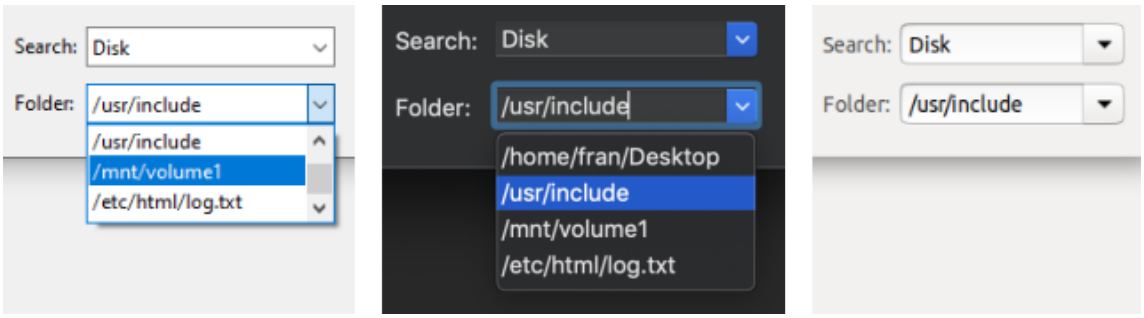


Figura 18.20: Combos en Windows, macOS y Linux.

18.7. ListBox

Los **ListBox** son controles que muestran una serie de elementos a modo de lista (Figura 18.21), (Figura 18.22), (Figura 18.23). Dependiendo de como se configure podremos seleccionar uno o varios elementos o visualizar *checkboxes* para marcarlos. El control habilita barras de scroll cuando sea necesario y permite la navegación con el teclado. En *"¡Hola ListBox!¡Hola ListBox!"* (Página 511) tienes un ejemplo de uso.

- Utiliza `listbox_create` parar crear un control de lista.
- Utiliza `listbox_add_elem` para añadir un elemento.
- Utiliza `listbox_multisel` para habilitar la selección múltiple.
- Utiliza `listbox_checkbox` para habilitar los checkboxes.
- Utiliza `listbox_OnSelect` para responder a la selección.

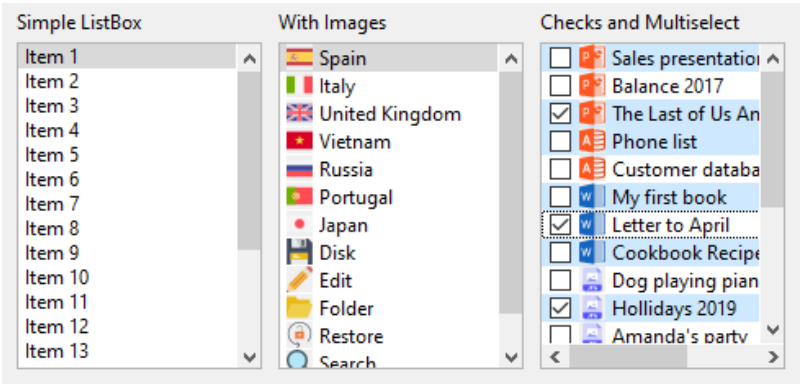


Figura 18.21: Controles ListBox en Windows.

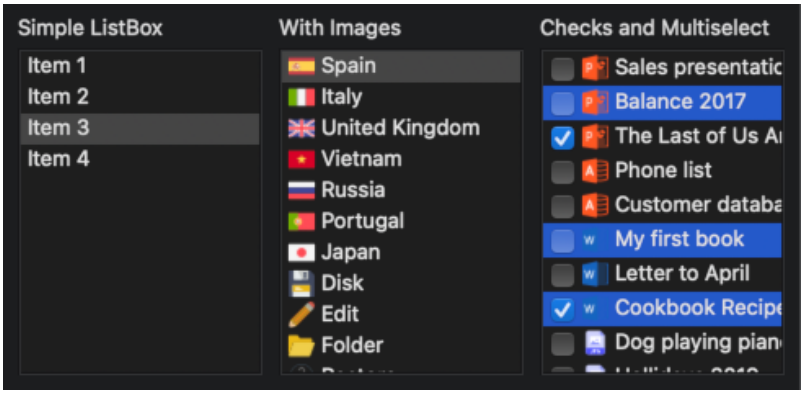


Figura 18.22: Controles ListBox en macOS.

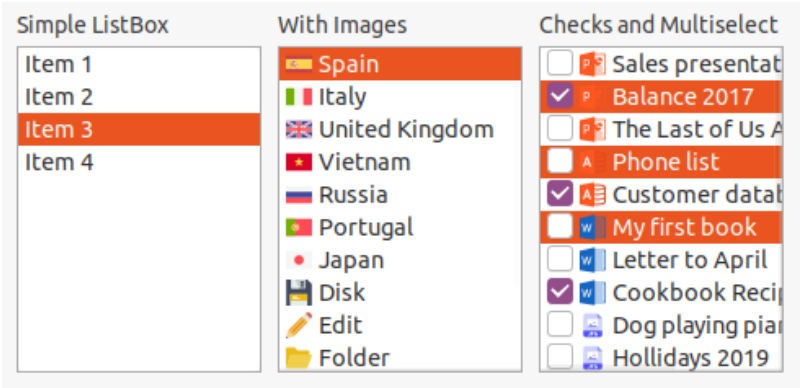


Figura 18.23: Controles ListBox en Linux.

18.8. UpDown

Los **UpDown** son controles tipo botón divididos horizontalmente dos partes (Figura 18.24). Cada parte tiene impresa una pequeña flecha y se utilizan, normalmente, para realizar incrementos discretos en valores numéricos asociados a controles *"Edit"* (Página 313).

- Utiliza `updown_create` para crear un botón updown.
- Utiliza `updown_OnClick` para responder a pulsaciones.

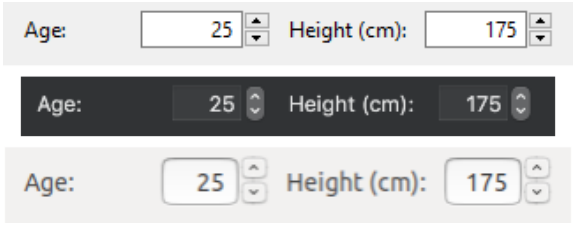


Figura 18.24: UpDown en Windows, macOS y Linux.

18.9. Slider

Los **Sliders** son controles deslizantes se que utilizan normalmente para editar valores numéricos continuos y acotados (Figura 18.25). A medida que el control se desplaza, se van produciendo eventos *OnMoved* que devuelven un valor entre 0 y 1. En *"¡Hola Slider y Progress!¡Hola Slider y Progress!"* (Página 513) tienes un ejemplo de uso.

- Utiliza `slider_create` para crear un slider horizontal.
- Utiliza `slider_vertical` para crear un slider vertical.
- Utiliza `slider_OnMoved` para responder al desplazamiento.

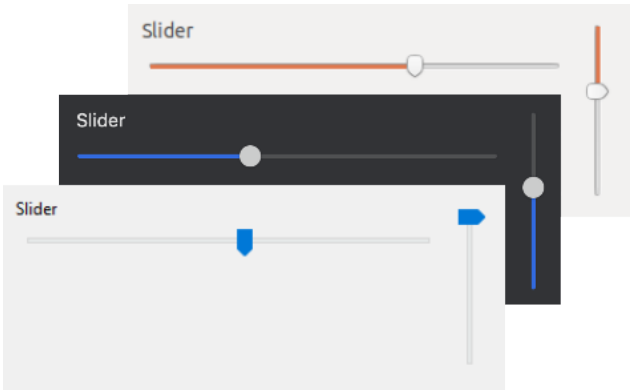


Figura 18.25: Sliders en Windows, macOS y Linux.

18.10. Progress

Las barras de progreso son controles pasivos que muestran el tiempo restante para completar una determinada tarea (Figura 18.26). A medida que transcurra el tiempo deberemos actualizar el control. El estado indefinido mostrará una animación sin indicar estado, lo que será útil cuando no podamos determinar el tiempo requerido.

- Utiliza `progress_create` para crear una barra de progreso.
- Utiliza `progress_undefined` para establecer la barra como indefinida.
- Utiliza `progress_value` para ir actualizando el progreso de la tarea.

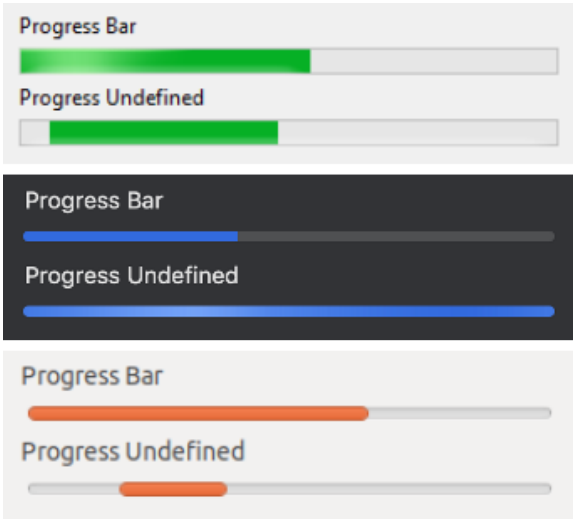


Figura 18.26: ProgressBar en Windows, macOS y Linux.

18.11. View

Los controles **View** o vistas personalizadas (Figura 18.27) son áreas en blanco dentro de la ventana que nos permiten implementar nuestros propios componentes. Tendremos total libertad para dibujar y capturar los eventos del teclado o ratón que nos permitan interactuar con la misma.

- Utiliza `view_create` para crear una vista.
- Utiliza `view_data` para vincular un objeto.
- Utiliza `view_get_data` para obtener este objeto.
- Utiliza `view_size` para establecer el tamaño por defecto.

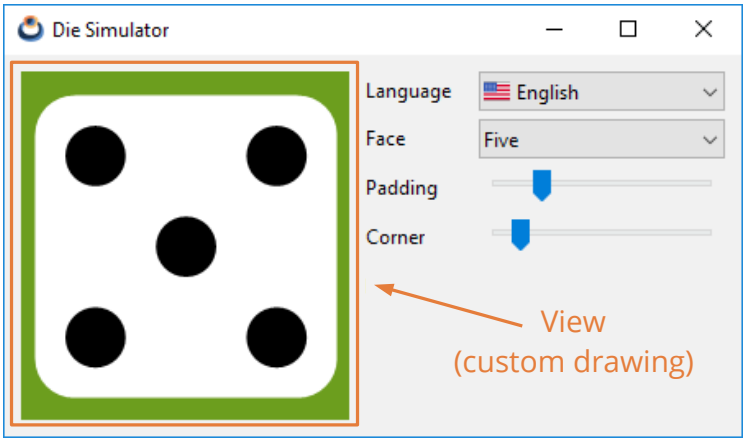


Figura 18.27: Control de vista personalizada.

18.11.1. Dibujar en vistas.

No podemos actualizar el área de dibujo cuando queramos. Esta puede verse afectada por otras ventanas del entorno, por lo que el *framebuffer* es gestionado directamente por el sistema operativo. Este último mandará una notificación cada vez que el control deba refrescar su contenido.

- Utiliza `view_OnDraw` para establecer la función de dibujo.
- Utiliza `view_update` para forzar la actualización del contenido.

En Die[1] tienes una sencilla aplicación de ejemplo que implementa el dibujado de vistas personalizadas. En ella se representa la figura de un dado, permitiéndonos editar ciertos

[1] `https://nappgui.com/es/demo/die.html`

parámetros del dibujo. Esta interacción irá lanzando una serie de eventos que requerirán el re-dibujado de nuestra figura. El ciclo completo puede resumirse en estos pasos (Figura 18.28):

- Se produce algún evento que requiere actualizar el contenido de la vista.
- La aplicación llama al método `view_update` para notificar que la vista debe ser actualizada.
- En el momento oportuno, el sistema mandará un evento `OnDraw` con un contexto `DCtx` listo para dibujar. Aquí puedes aplicar todo lo visto en *"Draw2D"* (Página 262) y utilizar líneas, figuras o textos para representar el contenido del control.

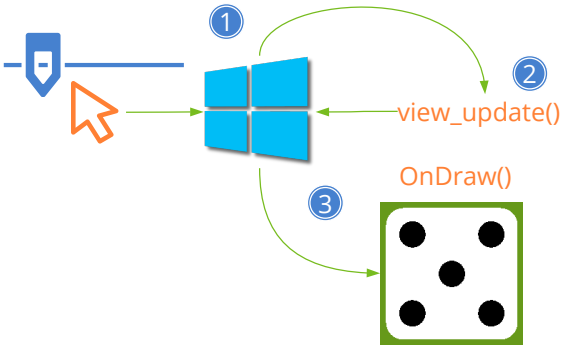


Figura 18.28: Ciclo de refresco de una vista personalizada.

El sistema operativo puede lanzar eventos `OnDraw` en cualquier momento sin que hallamos llamado previamente a `view_update`.

18.11.2. Vistas con scroll

Es posible que la "escena" a representar sea mucho más grande que el propio control, por lo que este mostrará solo un pequeño fragmento de la misma (Figura 18.29). En estos casos diremos que la vista es un *viewport* de la escena. Podemos gestionarlo de dos maneras:

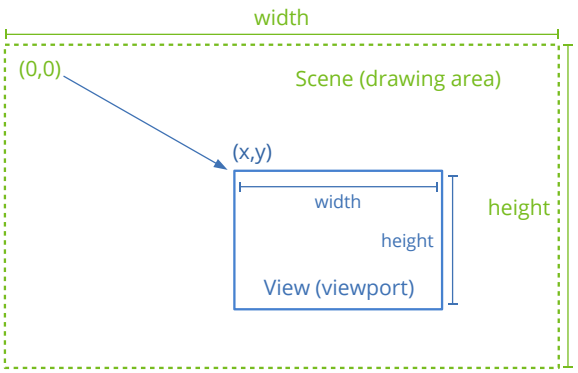


Figura 18.29: Escena y vista (viewport).

- Utilizar `draw_matrixf` para indicar la transformación que integra el desplazamiento, zoom y posible rotación del viewport con respecto a la escena. Todo ello debe ser gestionado por la aplicación y no tenemos que hacer nada especial, salvo llamar a `view_update()` cuando sea necesario.
- Utilizar barras de scroll que permitan al usuario desplazarse libremente por el contenido. En este caso la gestión de la vista se complica un poco más. Esto es lo que debemos tener en cuenta:
  - Utilizar `view_scroll` para crear la vista.
  - Utilizar `view_content_size` para indicar las medidas de la escena, con el fin de que se dimensionen correctamente las barras laterales.
  - Utilizar `view_scroll_x`, `view_scroll_y` para mover el origen del área visible.
  - Utilizar `view_viewport` para obtener la posición y dimensiones del área visible.

Algo muy importante es evitar dibujar elementos no visibles, sobre todo en escenas muy grandes o con multitud de objetos. El sistema operativo mandará sucesivos eventos `OnDraw()` a medida que el usuario manipule las scrollbars, indicando en la estructura `EvDraw` los parámetros del área visible. En *https://nappgui.com/es/howto/drawbig.html* tienes una aplicación de ejemplo que muestra como gestionar correctamente este tipo de casos.

Es posible que las dimensiones del viewport recibido en `OnDraw()` sean algo superiores a las medidas reales devueltas por `view_viewport()`. Esto es debido a que determinados sistemas (macOS, Linux) obligan a dibujar en ciertas zonas no visibles cercanas a los bordes, con el fin de evitar parpadeos en desplazamientos muy rápidos.

18.11.3. Uso del ratón

Para poder interactuar con el control es necesario definir manejadores para los diferentes eventos del ratón (Figura 18.30). El sistema operativo notificará las acciones del usuario con el fin de que la aplicación pueda lanzar las acciones pertinentes. No es necesario utilizarlos todos, tan solo los imprescindibles en cada caso.

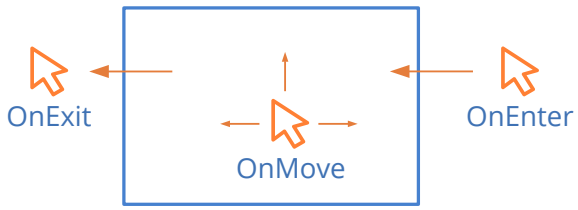


Figura 18.30: Eventos de posición con respecto a la vista.

- Utiliza `view_OnEnter` para saber cuando el cursor entre en la vista.

- Utiliza `view_OnExit` para saber cuando el cursor salga de la vista.
- Utiliza `view_OnMove` para saber cuando el cursor se está moviendo por la vista.
- Utiliza `view_OnDown` para saber cuando se pulse un botón dentro de la vista.
- Utiliza `view_OnUp` para saber cuando se libere un botón dentro de la vista.
- Utiliza `view_OnClick` para identificar un click (Up + Down rápido).
- Utiliza `view_OnDrag` para desplazamientos del cursor con un botón pulsado.
- Utiliza `view_OnWheel` para el uso de la ruedecilla del ratón.

Si la vista utiliza barras de scroll, la posición (x,y) del cursor pasada a `EvMouse` en cada evento, hace referencia a los coordenadas globales de la escena, teniendo en cuenta el desplazamiento. En vistas sin barras de scroll, son las coordenadas locales del control.

18.11.4. Uso del teclado

Para poder recibir eventos de teclado, es necesario que la vista sea capaz de obtener el foco, algo que por defecto está desactivado.

- Utiliza `layout_tabstop` para incluir la vista en el *tab-list* de la ventana y permitir que reciba el foco del teclado mediante la tecla `[TAB]` o al hacer clic sobre ella.
- Utiliza `view_OnKeyDown` para detectar cuando se pulse una tecla.
- Utiliza `view_OnKeyUp` para detectar cuando se libere una tecla.
- Utiliza `view_OnFocus` para notificar a la aplicación cada vez que la vista reciba (o pierda) el foco del teclado. En (Figura 18.31), la vista cambia el color de la celda activa cuando tiene el foco.

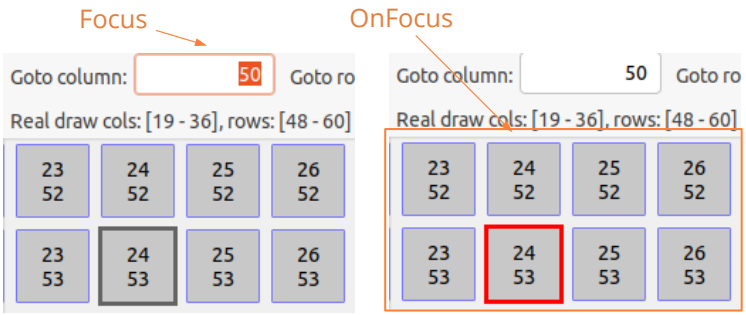


Figura 18.31: Vista sin el foco del teclado (izquierda) y con él (derecha).

En los eventos `KeyDown` y `KeyUp` se recibirá un `vkey_t` con el valor de la tecla pulsada. En (Figura 18.32) y (Figura 18.33) se muestran la correspondencia de estos códigos.

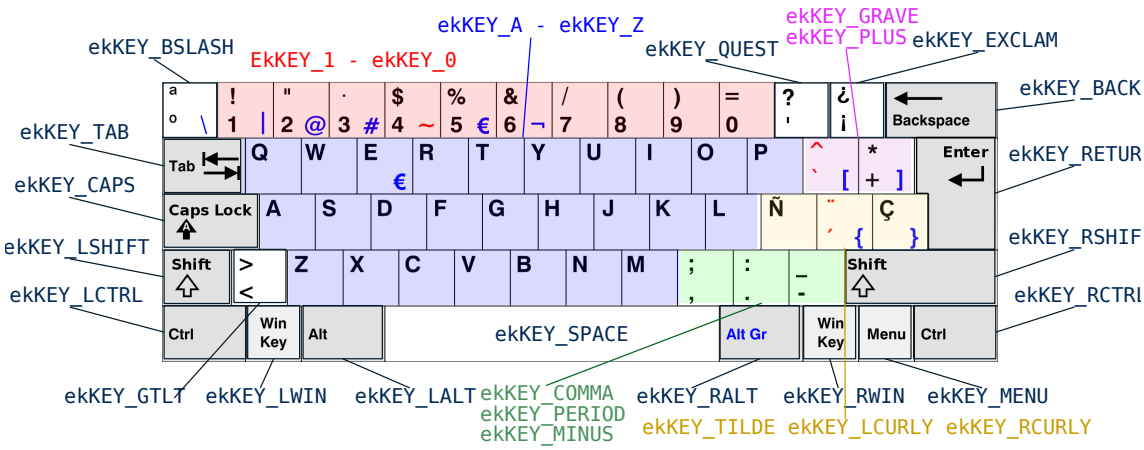


Figura 18.32: Códigos del teclado.

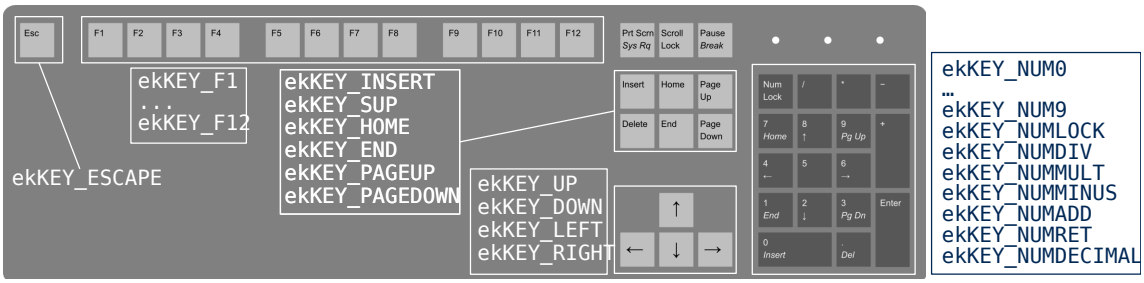


Figura 18.33: Códigos extendidos del teclado.

En *"Aplicaciones síncronasAplicaciones síncronas"* (Página 377) es posible que necesitemos saber si una tecla está pulsada o no durante el ciclo de actualización (síncrono) donde no tenemos acceso a los eventos `OnKeyDown` y `OnKeyUp` (asíncronos). Esto es posible hacerlo asignando a la vista un búfer de teclado mediante `view_keybuf`, que capturá los eventos asociados a cada tecla y nos permitirá consultar su estado en cualquier momento de forma cómoda.

18.12. TextView

Los **TextView** son vistas diseñadas para trabajar con bloques de texto enriquecido, donde pueden combinarse tipografías, tamaños y colores (Figura 18.34). Podemos considerarlas como la base de un editor de texto. En *"¡Hola TextView!¡Hola TextView!"* (Página 515) tienes un ejemplo de uso.

- Utiliza `textview_create` para crear una vista de texto.

- Utiliza `textview_writef` para añadir texto a la vista.
- Utiliza `textview_printf` para añadir texto con el formato de `printf`.
- Utiliza `textview_rtf` para añadir contenido en formato **RTF** de Microsoft.
- Utiliza `textview_clear` para borrar todo el texto.

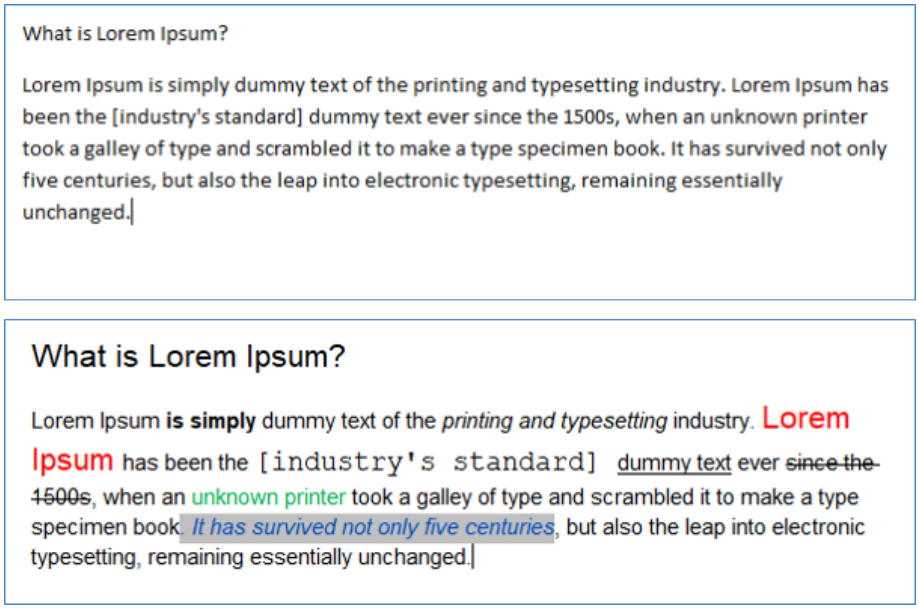


Figura 18.34: Texto plano y texto enriquecido.

18.12.1. Formato de carácter

Una de las ventajas del texto enriquecido con respecto al texto plano, es la posibilidad de combinar diferentes formatos de carácter dentro del mismo párrafo (Figura 18.35). Los cambios se aplicarán al nuevo texto que se añada al control.

Utiliza `textview_family` para cambiar la tipografía.

Utiliza `textview_fsize` para cambiar el tamaño del carácter.

Utiliza `textview_fstyle` para cambiar el estilo.

Utiliza `textview_color` para cambiar el color del texto.

Utiliza `textview_bgcolor` para cambiar el color de fondo del texto.

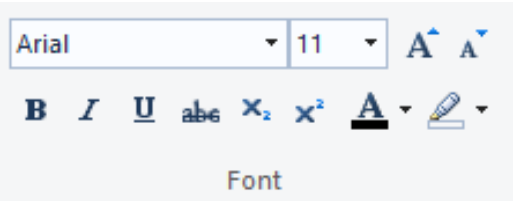


Figura 18.35: Controles típicos del formato de carácter.

18.12.2. Formato de párrafo

Se pueden establecer también atributos por párrafo (Figura 18.36). El carácter nueva línea `'\n'` se considera cierre o final del párrafo.

Utiliza `textview_halign` para establecer a alineación del párrafo.

Utiliza `textview_lspacing` para establecer la separación entre líneas (interlineado).

Utiliza `textview_bfspace` para indicar el espacio vertical antes del párrafo.

Utiliza `textview_afspace` para indicar el espacio vertical después del párrafo.

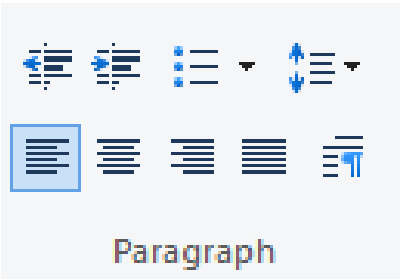


Figura 18.36: Controles típicos del formato de párrafo.

18.12.3. Formato del documento

Por último disponemos de varios atributos que afectan a la totalidad del documento o del control.

Utiliza `textview_units` para establecer la unidades del texto.

Utiliza `textview_pgcolor` para establecer el color del fondo del control (página).

18.13. ImageView

Los **ImageView** son vistas especializadas en visualizar imágenes y animaciones **GIF**.

- Utiliza `imageview_create` para crear un control de imagen.
- Utiliza `imageview_image` para establecer la imagen que mostrará el control.
- Utiliza `imageview_scale` para establecer el modo de ajuste de la imagen.

Figura 18.37: ImageView en un panel.

18.14. TableView

Las **TableView** son vistas de datos que muestran información tabulada organizada en filas y columnas (Figura 18.38), (Figura 18.39), (Figura 18.40). El control habilita barras de scroll y permite la navegación mediante el teclado. En *"¡Hola TableView!¡Hola TableView!"* (Página 518) tienes un ejemplo de uso.

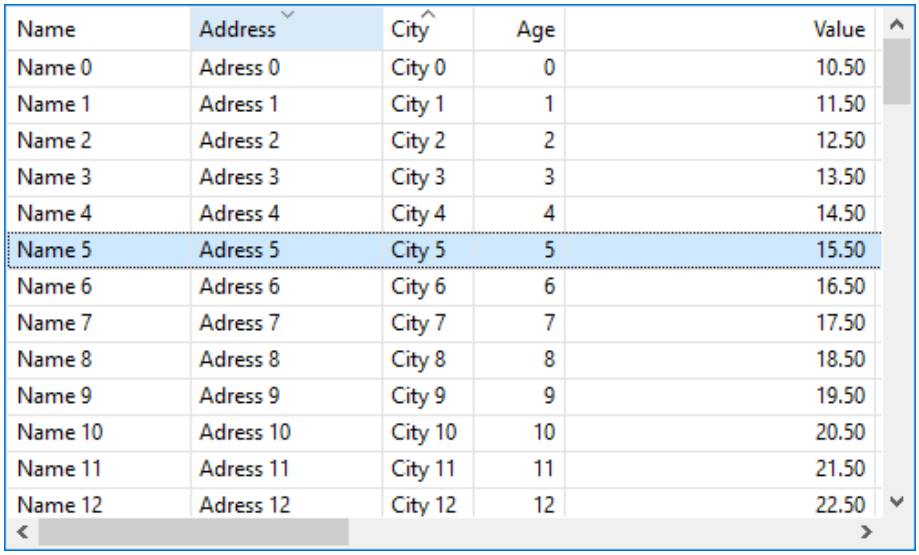
- Utiliza `tableview_create` parar crear un control de tabla.
- Utiliza `tableview_new_column_text` para añadir una columna.
- Utiliza `tableview_size` para establecer el tamaño por defecto.

18.14.1. Conexión de datos

Pensemos que una tabla puede contener miles de registros y estos pueden cambiar en cualquier momento desde diferentes fuentes de datos (disco, red, SGBDs, etc). Por este motivo, el TableView **no mantendrá ninguna caché interna**. Se ha diseñado con el objetivo de realizar un rápida visualización de los datos, pero sin entrar en la gestión de los mismos. Es la aplicación, en última instancia, la que debe suministrar esta información de una forma fluida.

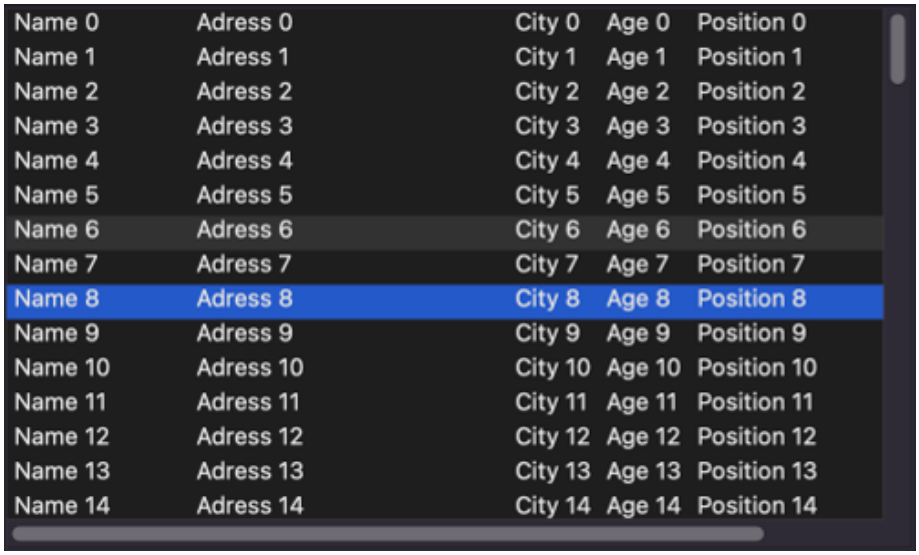
- Utiliza `tableview_OnData` para vincular la tabla con el origen de datos.
- Utiliza `tableview_update` para forzar la actualización de los datos de la tabla.

Cuando una tabla deba dibujar su contenido, como respuesta a un evento `OnDraw`, en primer lugar pedirá a la aplicación el número total de registros mediante una notificación `ekGUI_EVENT_TBL_NROWS`. Con ello podrá calcular el tamaño del documento y configurar las barras de scroll (Figura 18.41). Posteriormente irá lanzando sucesivos even-



Name	Address	City	Age	Value
Name 0	Adress 0	City 0	0	10.50
Name 1	Adress 1	City 1	1	11.50
Name 2	Adress 2	City 2	2	12.50
Name 3	Adress 3	City 3	3	13.50
Name 4	Adress 4	City 4	4	14.50
Name 5	Adress 5	City 5	5	15.50
Name 6	Adress 6	City 6	6	16.50
Name 7	Adress 7	City 7	7	17.50
Name 8	Adress 8	City 8	8	18.50
Name 9	Adress 9	City 9	9	19.50
Name 10	Adress 10	City 10	10	20.50
Name 11	Adress 11	City 11	11	21.50
Name 12	Adress 12	City 12	12	22.50

Figura 18.38: Control TableView en Windows.



Name 0	Adress 0	City 0	Age 0	Position 0
Name 1	Adress 1	City 1	Age 1	Position 1
Name 2	Adress 2	City 2	Age 2	Position 2
Name 3	Adress 3	City 3	Age 3	Position 3
Name 4	Adress 4	City 4	Age 4	Position 4
Name 5	Adress 5	City 5	Age 5	Position 5
Name 6	Adress 6	City 6	Age 6	Position 6
Name 7	Adress 7	City 7	Age 7	Position 7
Name 8	Adress 8	City 8	Age 8	Position 8
Name 9	Adress 9	City 9	Age 9	Position 9
Name 10	Adress 10	City 10	Age 10	Position 10
Name 11	Adress 11	City 11	Age 11	Position 11
Name 12	Adress 12	City 12	Age 12	Position 12
Name 13	Adress 13	City 13	Age 13	Position 13
Name 14	Adress 14	City 14	Age 14	Position 14

Figura 18.39: Control TableView en macOS.

tos `ekGUI_EVENT_TBL_CELL`, donde pedirá a la aplicación el contenido de cada celda (Figura 18.42). Todos estos requerimientos se realizarán a través de la función *callback* establecida en `tableview_OnData` (Listado 18.4).

TableView solo pedirá el contenido de la parte visible en cada momento.

Name	Address	City	Age	Value
Name 0	Adress 0	City 0	0	10.50
Name 1	Adress 1	City 1	1	11.50
Name 2	Adress 2	City 2	2	12.50
Name 3	Adress 3	City 3	3	13.50
Name 4	Adress 4	City 4	4	14.50
Name 5	Adress 5	City 5	5	15.50
Name 6	Adress 6	City 6	6	16.50
Name 7	Adress 7	City 7	7	17.50
Name 8	Adress 8	City 8	8	18.50
Name 9	Adress 9	City 9	9	19.50
Name 10	Adress 10	Cit...	10	20.50
Name 11	Adress 11	Cit...	11	21.50

Figura 18.40: Control TableView en Linux.

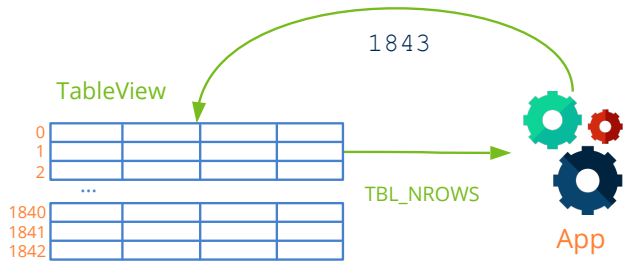


Figura 18.41: Petición del número de filas del conjunto de datos.

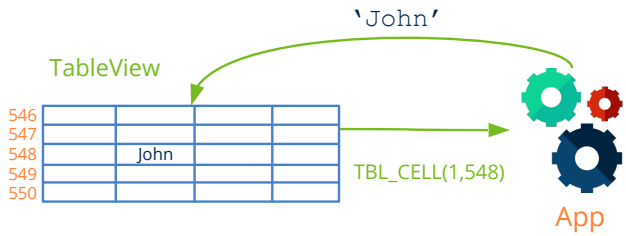


Figura 18.42: Petición de los datos de una celda.

Listado 18.4: Ejemplo de conexión de datos.

```
static void i_OnTableData(App *app, Event *e)
{
    uint32_t etype = event_type(e);
    unref(app);

    switch(etype) {
    case ekGUI_EVENT_TBL_NROWS:
    {
        uint32_t *n = event_result(e, uint32_t);
```

```
        *n = app_num_rows(app);
        break;
    }

    case ekGUI_EVENT_TBL_CELL:
    {
        const EvTbPos *pos = event_params(e, EvTbPos);
        EvTbCell *cell = event_result(e, EvTbCell);

        switch(pos->col) {
        case 0:
            cell->text = app_text_column0(app, pos->row);
            break;

        case 1:
            cell->text = app_text_column1(app, pos->row);
            break;

        case 2:
            cell->text = app_text_column2(app, pos->row);
            break;
        }

        break;
    }

    }
}

TableView *table = tableview_create();
tableview_OnData(table, listener(app, i_OnTableData, App));
tableview_update(table);
```

18.14.2. Caché de datos

Como ya hemos comentado, en cada instante la tabla solo mostrará una pequeña porción del conjunto de datos. Con el fin de suministrar estos datos de la manera más rápida posible, la aplicación puede mantener una caché con aquellos que vayan a ser mostrados a continuación. Para ello, antes de comenzar a dibujar la vista, la tabla mandará un evento tipo `ekGUI_EVENT_TBL_BEGIN` donde indicará el rango de filas y columnas que necesitan actualización (Figura 18.43). Este evento precederá a cualquier `ekGUI_EVENT_TBL_CELL` visto en la sección anterior. De igual forma, una vez actualizadas todas las celdas visibles, se enviará el evento `ekGUI_EVENT_TBL_END`, donde la aplicación podrá liberar los recursos en caché (Listado 18.5).

Listado 18.5: Ejemplo de empleo de caché de datos.

```
static void i_OnTableData(App *app, Event *e)
{
```

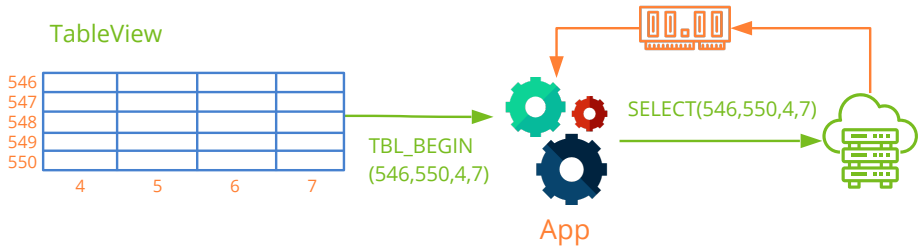


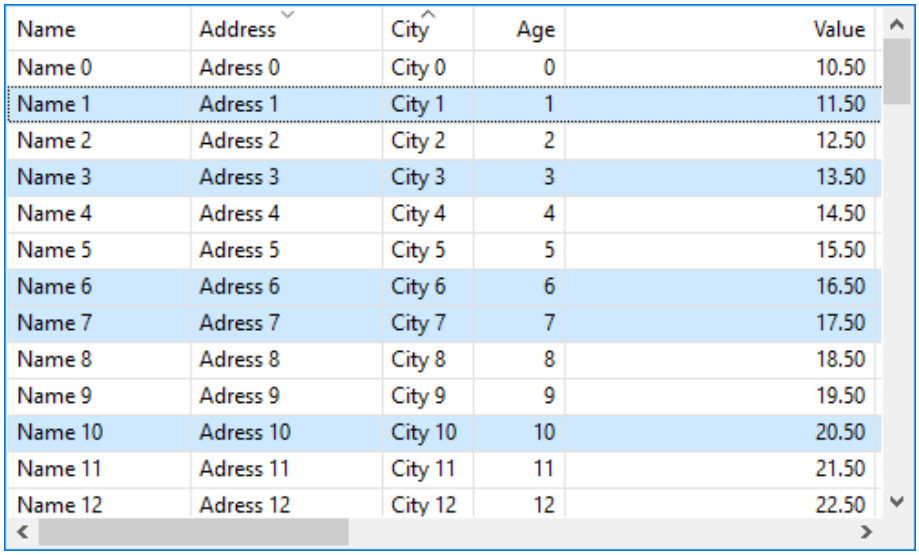
Figura 18.43: Uso de caché de datos.

```
    uint32_t etype = event_type(e);
    unref(app);

    switch(etype) {
    case ekGUI_EVENT_TBL_NROWS:
    {
        uint32_t *n = event_result(e, uint32_t);
        *n = app_num_rows(app);
        break;
    }

    case ekGUI_EVENT_TBL_BEGIN:
    {
        const EvTbRect *rect = event_params(e, EvTbRect);
        app->cache = app_fill_cache(app, rect->strow, rect->edrow, rect->stcol,
            ↪ rect->edcol);
        break;
    }

    case ekGUI_EVENT_TBL_CELL:
    {
        const EvTbPos *pos = event_params(e, EvTbPos);
        EvTbCell *cell = event_result(e, EvTbCell);
        cell->text = app_get_cache(app->cache, pos->row, pos->col);
        break;
    }

    case ekGUI_EVENT_TBL_END:
        app_delete_cache(app->cache);
        break;
}

TableView *table = tableview_create();
tableview_OnData(table, listener(app, i_OnTableData, App));
tableview_update(table);
```

18.14.3. Selección múltiple

Cuando navegamos por un `TableView` podemos activar la selección múltiple, que permitirá marcar más de una fila de la tabla (Figura 18.44).

- Utiliza `tableview_multisel` para activar o desactivar la selección múltiple.
- Utiliza `tableview_selected` para obtener las filas seleccionadas.
- Utiliza `tableview_select` para seleccionar un conjunto de filas.
- Utiliza `tableview_deselect` para de-seleccionar.
- Utiliza `tableview_deselect_all` para desmarcar todas las filas.
- Utiliza `tableview_OnSelect` para recibir un evento cuando cambie la selección.



Name	Address	City	Age	Value
Name 0	Adress 0	City 0	0	10.50
Name 1	Adress 1	City 1	1	11.50
Name 2	Adress 2	City 2	2	12.50
Name 3	Adress 3	City 3	3	13.50
Name 4	Adress 4	City 4	4	14.50
Name 5	Adress 5	City 5	5	15.50
Name 6	Adress 6	City 6	6	16.50
Name 7	Adress 7	City 7	7	17.50
Name 8	Adress 8	City 8	8	18.50
Name 9	Adress 9	City 9	9	19.50
Name 10	Adress 10	City 10	10	20.50
Name 11	Adress 11	City 11	11	21.50
Name 12	Adress 12	City 12	12	22.50

Figura 18.44: TableView con selección múltiple.

La navegación por un `TableView` funciona igual que otros controles similares, como el explorador de archivos.

- `[UP]/[DOWN]` para desplazar verticalmente.
- `[LEFT]/[RIGHT]` para desplazar horizontalmente.
- `[PAGEUP]/[PAGEDOWN]` avanzar o retroceder una página.
- `[HOME]` va al inicio de la tabla.
- `[END]` va al final de la tabla.
- `[CTRL]+clic` selección múltiple con el ratón.

- `[SHIFT]+[UP]/[DOWN]` selección multiple con el teclado.

En selección múltiple, se producirá una **de-selección automática de las filas** siempre que hagamos clic liberando `[CTRL]` o pulsemos cualquier tecla de navegación liberando `[SHIFT]`. Si queremos navegar sin perder la selección previa, deberemos activar el flag `preserve` en `tableview_multisel`.

18.14.4. Configurar columnas

- Utiliza `tableview_column_width` para establecer el ancho de una columna.
- Utiliza `tableview_column_limits` para establecer limites en el ancho.
- Utiliza `tableview_column_resizable` para permitir estirar o contraer la columna.
- Utiliza `tableview_header_visible` para mostrar u ocultar la cabecera.
- Utiliza `tableview_OnHeaderClick` para notificar el clic sobre la cabecera.
- Utiliza `tableview_column_freeze` para fijar una o varias columnas (Figura 18.45).

Name	Address	a 1	Extra Data 2
Name 0	Adress 0	1 0	Extra Data 2 0
Name 1	Adress 1	1 1	Extra Data 2 1
Name 2	Adress 2	1 2	Extra Data 2 2
Name 3	Adress 3	1 3	Extra Data 2 3
Name 4	Adress 4	1 4	Extra Data 2 4
Name 5	Adress 5	1 5	Extra Data 2 5
Name 6	Adress 6	1 6	Extra Data 2 6
Name 7	Adress 7	1 7	Extra Data 2 7
Name 8	Adress 8	1 8	Extra Data 2 8
Name 9	Adress 9	1 9	Extra Data 2 9
Name 10	Adress 10	1 10	Extra Data 2 10
Name 11	Adress 11	1 11	Extra Data 2 11
Name 12	Adress 12	1 12	Extra Data 2 12

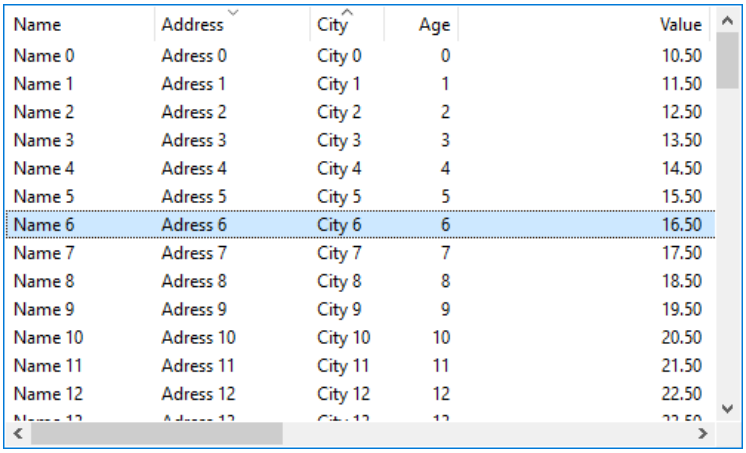
Figura 18.45: Columnas 0 y 1 congeladas.

18.14.5. Dibujo de rejilla

- Utiliza `tableview_grid` para mostrar u ocultar las líneas interiores (Figura 18.46), (Figura 18.47).

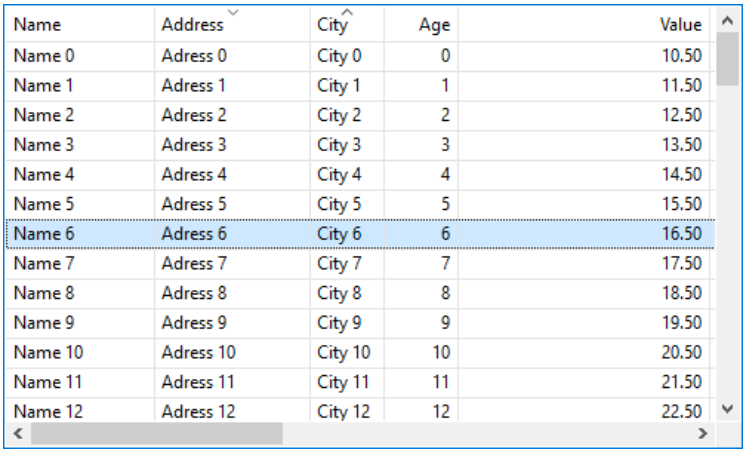
18.15. SplitView

Las **SplitView** son vistas divididas en dos partes, donde en cada una de ellas emplazamos otra vista o un panel. La línea divisoria es desplazable, lo que permite redimensionar



Name	Address	City	Age	Value
Name 0	Adress 0	City 0	0	10.50
Name 1	Adress 1	City 1	1	11.50
Name 2	Adress 2	City 2	2	12.50
Name 3	Adress 3	City 3	3	13.50
Name 4	Adress 4	City 4	4	14.50
Name 5	Adress 5	City 5	5	15.50
Name 6	Adress 6	City 6	6	16.50
Name 7	Adress 7	City 7	7	17.50
Name 8	Adress 8	City 8	8	18.50
Name 9	Adress 9	City 9	9	19.50
Name 10	Adress 10	City 10	10	20.50
Name 11	Adress 11	City 11	11	21.50
Name 12	Adress 12	City 12	12	22.50

Figura 18.46: TableView sin líneas interiores.



Name	Address	City	Age	Value
Name 0	Adress 0	City 0	0	10.50
Name 1	Adress 1	City 1	1	11.50
Name 2	Adress 2	City 2	2	12.50
Name 3	Adress 3	City 3	3	13.50
Name 4	Adress 4	City 4	4	14.50
Name 5	Adress 5	City 5	5	15.50
Name 6	Adress 6	City 6	6	16.50
Name 7	Adress 7	City 7	7	17.50
Name 8	Adress 8	City 8	8	18.50
Name 9	Adress 9	City 9	9	19.50
Name 10	Adress 10	City 10	10	20.50
Name 11	Adress 11	City 11	11	21.50
Name 12	Adress 12	City 12	12	22.50

Figura 18.47: TableView con líneas interiores.

ambas mitades, repartiendo el tamaño total del control entre los hijos (Figura 18.48), (Figura 18.49), (Figura 18.50). En *"¡Hola SplitView!¡Hola SplitView!"* (Página 525) tienes un ejemplo de uso.

- Utiliza `splitview_horizontal` para crear una vista partida.
- Utiliza `splitview_size` para establecer el tamaño inicial.

18.15.1. Añadir controles

Existen varias funciones para añadir controles "hijo" al splitview. La primera llamada a cualquiera de ellas emplazará la vista o panel en la lado izquierdo o superior. La segunda llamada lo hará en el lado derecho o inferior. Sucesivas llamadas generarán un error.

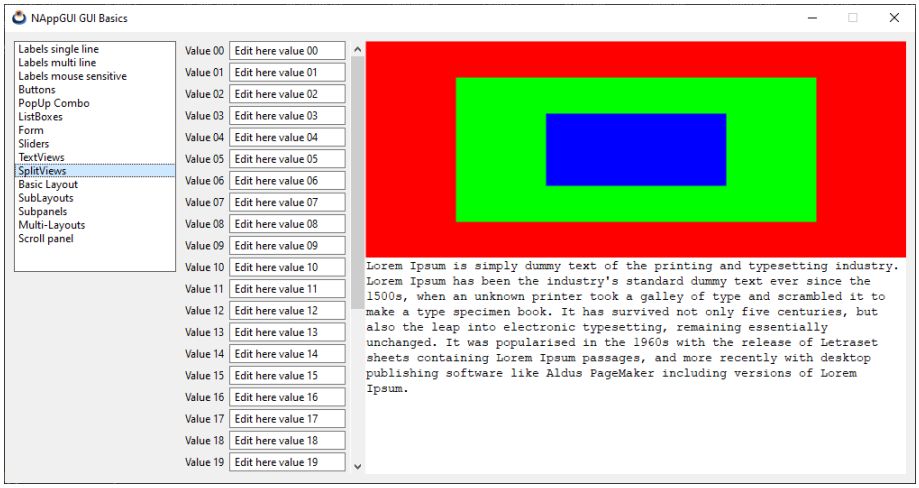


Figura 18.48: SplitView en Windows.

Figura 18.49: SplitView en macOS.

- Utiliza `splitview_view` para añadir una vista personalizada.
- Utiliza `splitview_panel` para añadir un panel.

18.15.2. Modos de división

Tenemos dos modos de comportamiento de la barra divisoria y ambos se activan desde esta función:

- Utiliza `splitview_pos` para establecer la modalidad del separador.

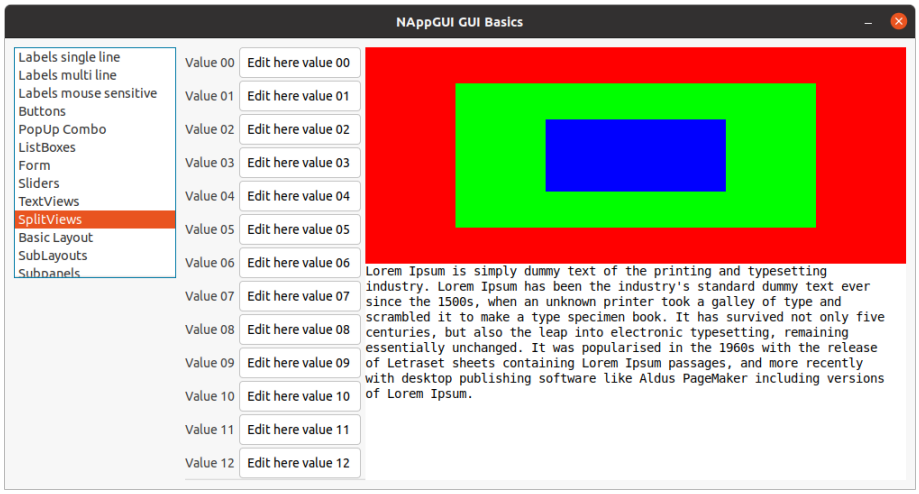


Figura 18.50: SplitView en Linux.

- Modo proporcional: La posición de la divisoria siempre se mantendrá constante con respecto al tamaño del splitview. Es decir, un valor 0.3 significa que la vista izquierda siempre ocupará 1/3 del tamaño total y la derecha 2/3. Para ello indicar un valor entre 0 y 1 en el parámetro `pos`.
- Modo fijo: Los cambios de tamaño del splitview siempre dejarán una de las partes con un tamaño constante. Si `pos > 1` la vista izquierda/superior mantendrá el número de píxeles indicados. Por el contrario, si `pos < 0` ocurrirá lo mismo con la vista derecha/inferior.

La proporción o valor cambiarán si el usuario arrastra la línea divisoria, pero no lo hará el modo de funcionamiento.

18.16. Layout

Un **Layout** es una rejilla virtual y transparente vinculada siempre con un `Panel` que sirve para ubicar los diferentes elementos de la interfaz (Figura 18.51). Sus celdas interiores tienen la capacidad de dimensionarse automáticamente en función de su contenido, con lo que se consigue una gran portabilidad debido a que no es necesario indicar coordenadas ni tamaños específicos para los controles. Para ilustrar el concepto, vamos a simplificar ligeramente el código de *"¡Hola Edit y UpDown!¡Hola Edit y UpDown!"* (Página 507) (Listado 18.6), cuyo resultado podemos verlo en (Figura 18.52).

- Utiliza `layout_create` para crear un nuevo layout.
- Utiliza `layout_label` y similares para ubicar controles en las diferentes celdas.

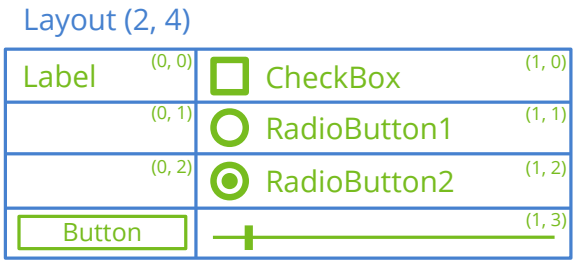


Figura 18.51: Un layout sirve para ubicar controles en el área del panel.

Listado 18.6: Layout de dos columnas y cinco filas.

```
Layout *layout = layout_create(2, 5);
Label *label1 = label_create();
Label *label2 = label_create();
Label *label3 = label_create();
Label *label4 = label_create();
Label *label5 = label_create();
Edit *edit1 = edit_create();
Edit *edit2 = edit_create();
Edit *edit3 = edit_create();
Edit *edit4 = edit_create();
Edit *edit5 = edit_create();
label_text(label1, "User Name:");
label_text(label2, "Password:");
label_text(label3, "Address:");
label_text(label4, "City:");
label_text(label5, "Phone:");
edit_text(edit1, "Amanda Callister");
edit_text(edit2, "aQwe56nhjJk");
edit_text(edit3, "35, Tuam Road");
edit_text(edit4, "Galway - Ireland");
edit_text(edit5, "+35 654 333 000");
edit_passmode(edit2, TRUE);
layout_label(layout, label1, 0, 0);
layout_label(layout, label2, 0, 1);
layout_label(layout, label3, 0, 2);
layout_label(layout, label4, 0, 3);
layout_label(layout, label5, 0, 4);
layout_edit(layout, edit1, 1, 0);
layout_edit(layout, edit2, 1, 1);
layout_edit(layout, edit3, 1, 2);
layout_edit(layout, edit4, 1, 3);
layout_edit(layout, edit5, 1, 4);
```

18.16.1. Dimensionado natural

El resultado de (Figura 18.52), si bien no es muy estético, es lo que denominamos **dimensionado natural** que es la maquetación por defecto aplicada en función del contenido

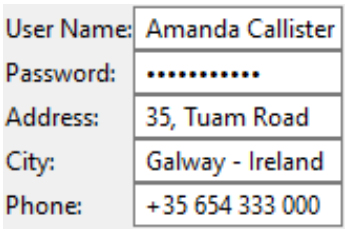


Figura 18.52: Resultado de (Listado 18.6).

de las celdas. En (Tabla 18.2) tienes las medidas por defecto de cada control. El ancho de columna se fija al del elemento más ancho y de igual forma se calcula la altura de las filas. El tamaño final del layout será la suma de las medidas tanto de columnas como de filas.

Control	Anchura	Altura
`Label`	Ajustado al texto.	Ajustado al texto considerando `'\n'`.
`Button` (push)	Ajustado al texto + márgen.	Según el tema del S.O.
`Button` (check/radio)	Ajustado al texto + icono.	Ajustado al icono.
`Button` (flat)	Ajustado al icono + margen.	Ajustado al icono + margen.
`PopUp`	Ajustado al texto mas largo.	Según el tema del S.O.
`Edit`	100 Unidades (px).	Ajustado al texto + margen.
`Combo`	100 Unidades (px).	Según el tema del S.O.
`ListBox`	128 px o `listbox_size`.	128 px o `listbox_size`.
`UpDown`	Según el tema del S.O.	Según el tema del S.O.
`Slider` (horizontal)	100 Unidades (px).	Según el tema del S.O.
`Slider` (vertical)	Según el tema del S.O.	100 Unidades (px).
`Progress`	100 Unidades (px).	Según el tema del S.O.
`View`	128 px o `view_size`.	128 px o `view_size`.
`TextView`	256 px o `textview_size`.	144 px o `textview_size`.
`ImageView`	64 px o `imageview_size`.	64 px o `imageview_size`.
`TableView`	256 px o `tableview_size`.	128 px o `tableview_size`.
`SplitView`	128 px o `splitview_size`.	128 px o `splitview_size`.
`Panel`	Dimensión natural.	Dimensión natural.
`Panel` (con scroll)	256 px o `panel_size`.	256 px o `panel_size`.

Tabla 18.2: Dimensionado natural de controles.

Los márgenes y constantes aplicados a los controles son los necesarios para cumplir con las **human guidelines** de cada gestor de ventanas. Esto quiere decir que un *PushButton* con el texto `"Hello"` no tendrá las mismas dimensiones en WindowsXP que en macOS Mavericks o Ubuntu 16.

Las celdas vacías se dimensionarán con tamaño 0 y no afectarán a la composición.

18.16.2. Márgenes y formato

El dimensionado natural que acabamos de ver ajusta el panel al tamaño mínimo necesario para albergar correctamente todos los controles, pero no siempre resulta estético. Podemos darle forma añadiendo márgenes o forzando un tamaño determinado para filas y columnas (Listado 18.7) (Figura 18.53).

- Utiliza `layout_hsize` para forzar la anchura de una columna.
- Utiliza `layout_vsize` para forzar la altura de una fila.
- Utiliza `layout_hmargin` para establecer un margen inter-columnas.
- Utiliza `layout_vmargin` para establecer un margen inter-filas.
- Utiliza `layout_margin` para establecer un margen en el borde del layout.

Listado 18.7: Aplicando formato a (Listado 18.6).

```
layout_hsize(layout, 1, 235);
layout_hmargin(layout, 0, 5);
layout_vmargin(layout, 0, 5);
layout_vmargin(layout, 1, 5);
layout_vmargin(layout, 2, 5);
layout_vmargin(layout, 3, 5);
layout_margin(layout, 10);
```

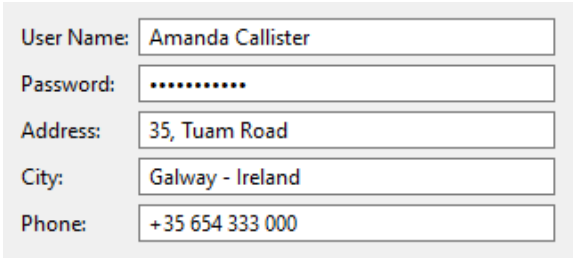


Figura 18.53: Resultado de (Listado 18.7).

18.16.3. Alineación

Es habitual que la anchura un control sea inferior a la anchura de la columna que lo contiene, bien porque se ha forzado un ancho fijo o bien porque existan elementos más anchos en la misma columna. En estos casos, podemos indicar la alineación horizontal o vertical del control respecto a la celda (Figura 18.54). En (Tabla 18.3) tienes las alineaciones por defecto.

- Utiliza `layout_halign` para cambiar la alineación horizontal de una celda.
- Utiliza `layout_valign` para cambiar la alineación vertical de una celda.

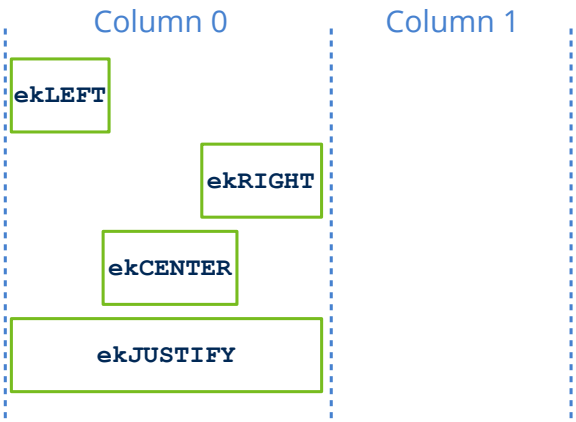


Figura 18.54: Alineación horizontal.

Control	Horizontal	Vertical
`Label`	`ekLEFT`	`ekCENTER`
`Button` (push)	`ekJUSTIFY`	`ekCENTER`
`Button` (resto)	`ekLEFT`	`ekCENTER`
`PopUp`	`ekJUSTIFY`	`ekCENTER`
`Edit`	`ekJUSTIFY`	`ekTOP`
`Edit` (multiline)	`ekJUSTIFY`	`ekJUSTIFY`
`Combo`	`ekJUSTIFY`	`ekCENTER`
`ListBox`	`ekJUSTIFY`	`ekJUSTIFY`
`UpDown`	`ekJUSTIFY`	`ekJUSTIFY`
`Slider` (horizontal)	`ekJUSTIFY`	`ekCENTER`
`Slider` (vertical)	`ekCENTER`	`ekJUSTIFY`

Control	Horizontal	Vertical
`Progress`	`ekJUSTIFY`	`ekCENTER`
`View`	`ekJUSTIFY`	`ekJUSTIFY`
`TextView`	`ekJUSTIFY`	`ekJUSTIFY`
`ImageView`	`ekJUSTIFY`	`ekJUSTIFY`
`TableView`	`ekJUSTIFY`	`ekJUSTIFY`
`SplitView`	`ekJUSTIFY`	`ekJUSTIFY`
`Layout` (sublayout)	`ekJUSTIFY`	`ekJUSTIFY`
`Panel`	`ekJUSTIFY`	`ekJUSTIFY`

Tabla 18.3: Alineación por defecto de controles.

18.16.4. Sub-layouts

Consideremos ahora el panel de (Figura 18.55). No es difícil darse cuenta que esta disposición no encaja de ninguna manera en una rejilla rectangular, por lo que ha llegado el momento de utilizar **sublayouts**. Además de controles individuales, una celda admite también otro layout, por lo que podemos dividir el panel original en tantas partes como sean necesarias hasta conseguir la maquetación deseada. El layout principal irá dimensionando cada sublayout de forma recursiva e integrándolo en la composición final. En *"¡Hola Sublayout!¡Hola Sublayout!"* (Página 543) tienes el código que genera este ejemplo.

- Utiliza `layout_layout` para asignar un layout completo a una celda de otro layout.

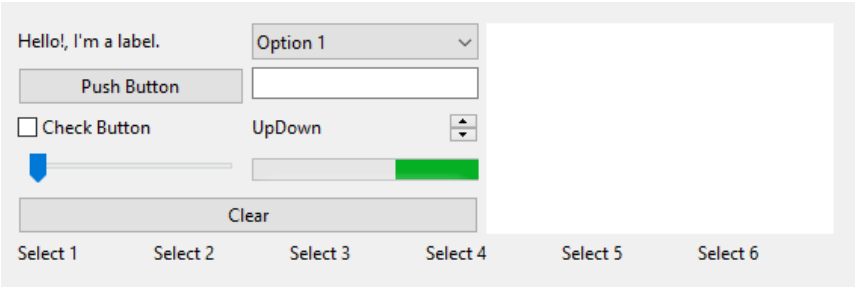


Figura 18.55: Composición de panel compleja.

En este caso hemos aplicado la filosofía del divide y vencerás, hasta asegurar que cada parte encaje en una rejilla individual (Figura 18.56). Cada sublayout lo hemos codificado en una función independiente para darle mayor consistencia al código, aplicando márgenes y formato de forma individual dentro de cada una de ellas (Listado 18.8).

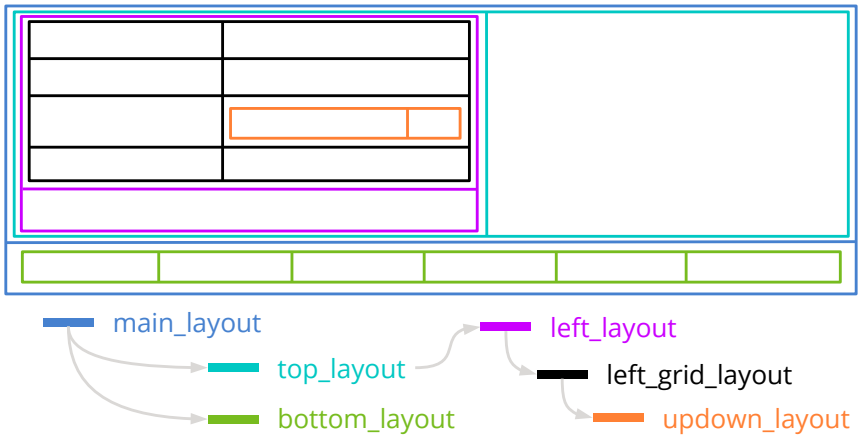


Figura 18.56: Sublayouts necesarios para componer el panel de (Figura 18.55).

Listado 18.8: Integración de sublayouts (parcial).

```
static Layout *i_main_layout(void)
{
    Layout *layout1 = layout_create(1, 2);
    Layout *layout2 = i_top_layout();
    Layout *layout3 = i_bottom_layout();
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 0, 1);
    layout_margin(layout1, 5);
    layout_vmargin(layout1, 0, 5);
    return layout1;
}
```

18.16.5. Expansión de celdas

En ciertas ocasiones, el tamaño de un layout viene forzado por condiciones externas. Esto ocurre cuando tenemos un sublayout en una celda con alineación `ekJUSTIFY` (expansión interna) o cuando el usuario cambia el tamaño de una ventana re-dimensionable (expansión externa). Esto producirá un "exceso de píxeles" entre el dimensionado natural y el tamaño real de la celda (Figura 18.57). Esta situación se resuelve repartiendo el sobrante por igual entre todas las columnas del sublayout, que a su vez, se irán expandiendo de forma recursiva hasta llegar a una celda vacía o un control individual. Podemos cambiar este reparto equitativo mediante estas funciones:

- Utiliza `layout_hexpand` para expandir una única celda y dejar el resto con su tamaño por defecto.
- Utiliza `layout_hexpand2` para expandir dos celdas indicando la proporción de crecimiento de cada una.

- Utiliza `layout_hexpand3` para expandir tres celdas.

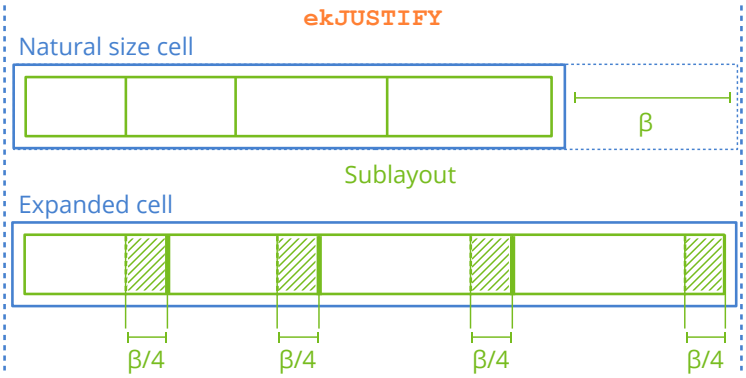


Figura 18.57: Cuando el tamaño del sublayout viene dado por condiciones externas, se reparte por igual el exceso de píxeles entre las columnas (expansión horizontal) y filas (expansión vertical).

La expansión vertical funciona exactamente igual, repartiendo el espacio sobrante entre las filas del layout.

18.16.6. Tabstops

Es una práctica habitual en la mayoría de sistemas utilizar la tecla `[TAB]` y la combinación `[SHIFT]+[TAB]` para navegar por los diferentes controles de una ventana o formulario. Términos como **taborder** o **tabstop** hacen referencia tanto al orden de navegación como a la pertenencia (o no) de un elemento a dicha lista. Si bien es posible organizar los elementos de una *tab-list* aleatoriamente, los layouts proporcionan un orden natural coherente basado en la ubicación de los controles. Por defecto, cada layout crea una *tab-list* recorriendo todas sus celdas por filas , pero podemos cambiarlo:

- Utiliza `layout_taborder` para organizar la *tab-list* por filas o columnas.
- Utiliza `layout_tabstop` para añadir o quitar controles de la *tab-list*.

No todas las celdas de un layout tienen porqué ser un *tabstop*, ya que no tiene sentido que controles estáticos como `Label` reciban el foco del teclado. En (Tabla 18.4) tienes que controles se incluyen por defecto en dicha lista. Con `layout_tabstop` podrás añadir o quitar controles de la *tab-list*. En la mayoría de ocasiones la utilizaremos que permitir que ciertas vistas personalizadas `View` reciban el foco del teclado.

Control	Incluido
`Label`	NO

Figura 18.58: Taborder por filas en layouts y sublayouts.

Control	Incluido
`Button`	SÍ
`PopUp`	SÍ
`Edit`	SÍ
`Combo`	SÍ
`ListBox`	SÍ
`UpDown`	NO
`Slider`	SÍ
`Progress`	NO
`View`	NO
`TextView`	NO
`ImageView`	NO
`TableView`	SÍ
`SplitView`	SÍ
`Layout` (sublayout)	SÍ
`Panel`	NO

Tabla 18.4: Controles incluidos por defecto en el *tab-list*.

Cuando el taborder entra en un sublayout, seguirá el orden local de este último. Cuando salga del sublayout continuará con el orden principal.

Por otro lado, puede ser útil mover el foco del teclado desde el propio código y no esperar a la pulsación de `[TAB]` por parte del usuario. En *"¡Hola IP-Input!¡Hola IP-*

Input!" (Página 552) tienes varios `Edit` que pasan al siguiente control cuando se introducen exactamente tres números.

- Utiliza `layout_next_tabstop` para saltar al siguiente control de la *tab-list*.
- Utiliza `layout_previous_tabstop` para saltar al anterior control de la *tab-list*.
- Utiliza `cell_focus` para establecer el foco en una celda concreta.

El control del foco del teclado solo tendrá efecto en layouts vinculados con una ventana activa.

18.17. Cell

Las celdas son los elementos internos de un *"Layout"* (Página 335) y albergarán un control o un sublayout (Figura 18.59).

- Utiliza `layout_cell` para obtener la celda.
- Utiliza `cell_button` para obtener el control interior.
- Utiliza `cell_layout` para obtener el sublayout interior.
- Utiliza `cell_enabled` para activar o desactivar los controles.
- Utiliza `cell_visible` para mostrar y ocular el contenido.
- Utiliza `cell_focus` para asignar el foco del teclado.
- Utiliza `cell_padding` para establecer los márgenes interiores (Figura 18.60).

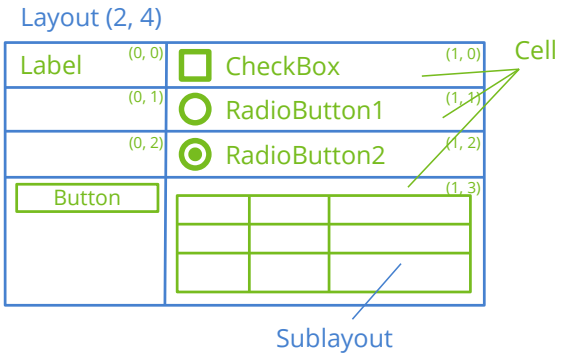


Figura 18.59: Celdas en el interior de un Layout

18.18. Panel

Un **Panel** es un control dentro de una ventana que agrupa otros controles. Define su propio sistema de referencia, es decir, si desplazamos un panel todos sus descendientes se

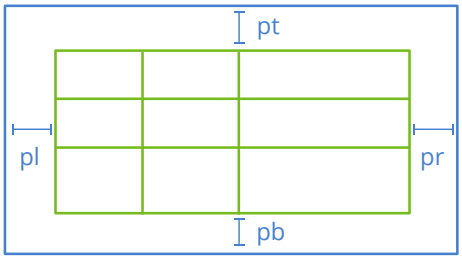


Figura 18.60: Padding interior de una celda.

moverán al unísono ya que sus ubicaciones serán relativas al origen del mismo. Admitirá otros (sub)-paneles como descendientes, lo que permite conformar una **Jerarquía de ventana** (Figura 18.61). En pro de la portabilidad, esta librería **Gui** no permite indicar directamente coordenadas ni tamaños concretos para los elementos dentro de un panel, sino que la asociación se lleva a cabo mediante un objeto `Layout` que se encarga de calcular en tiempo de ejecución las ubicaciones finales de los controles en función de la plataforma y gestor de ventanas. En *"¡Hola Subpanel!¡Hola Subpanel!"* (Página 547) tienes un ejemplo elemental del uso de paneles.

- Utiliza `panel_create` para crear un nuevo panel.
- Utiliza `panel_scroll` para crear un nuevo panel con barras de scroll.
- Utiliza `panel_layout` para añadir controles al panel.
- Utiliza `panel_size` para establecer el tamaño por defecto del panel.

Cada panel admite varios layouts y permite alternar entre ellos en tiempo de ejecución (Figura 18.62). Esto permite crear interfaces dinámicas responsivas con muy poco esfuerzo, ya que el propio panel se encarga de vincular y dimensionar los controles en función del layout activo en cada caso. En *"¡Hola Multi-layout!¡Hola Multi-layout!"* (Página 548) tienes un ejemplo.

- Utiliza `panel_visible_layout` para alternar entre layouts.

Debido a que los layout son estructuras lógicas fuera de la jerarquía de ventana, pueden compartir controles al estar vinculados con el mismo panel (Figura 18.63). Lo que no está permitido es utilizar los mismos objetos en diferentes paneles, por el propio concepto de jerarquía.

18.18.1. Entendiendo el dimensionado de paneles

Vamos a mostrar, por medio de un ejemplo, la lógica que se esconde detrás de la composición y dimensionado de paneles. Empezamos con (Listado 18.9) donde creamos un panel relativamente extenso en altura.

Listado 18.9: Composición de un panel con varias filas de edición.

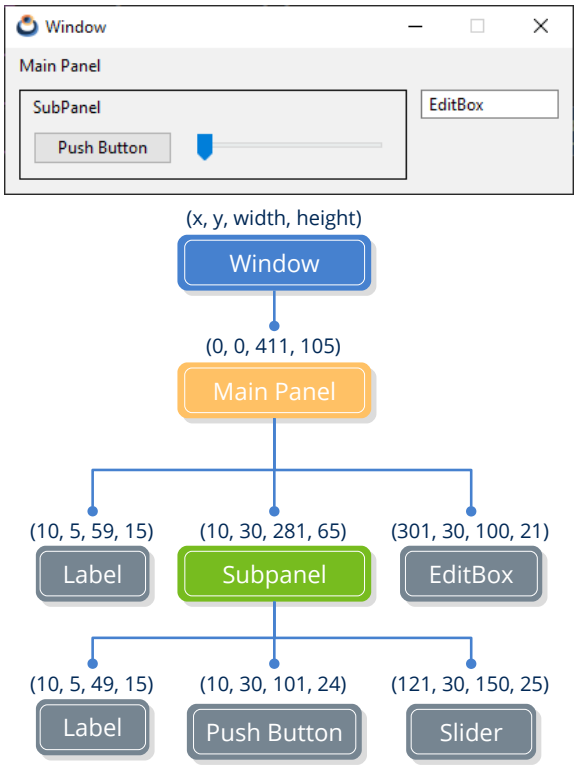


Figura 18.61: Jerarquía de ventana.

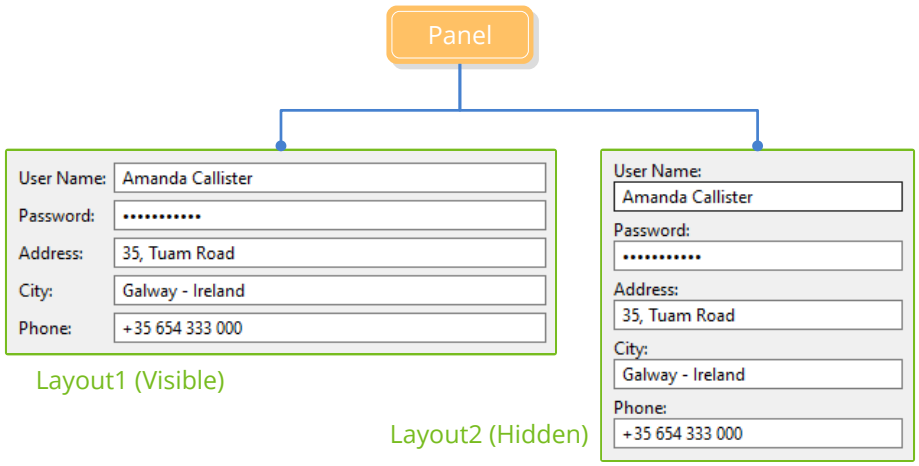


Figura 18.62: Panel con dos organizaciones diferentes para los mismos controles.

```
static Window *i_window(void)
{
    uint32_t i, n = 20;
    Window *window_create(ekWINDOW_STDRES);
    Panel *panel = panel_create();
```

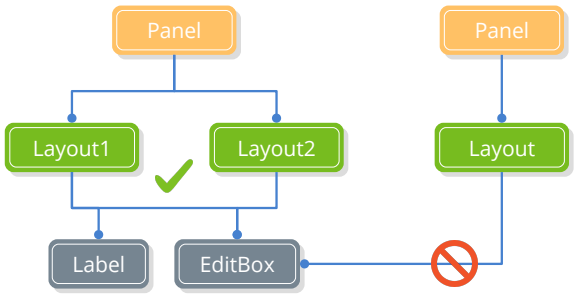


Figura 18.63: Es posible reutilizar los mismos componentes entre layouts del mismo panel.

```
    Layout *layout = layout_create(2, n);

    for (i = 0; i < n; ++i)
    {
        char_t text[64];
        Label *label = label_create();
        Edit *edit = edit_create();
        bstd_sprintf(text, sizeof(text), "Value %02d", i);
        label_text(label, text);
        bstd_sprintf(text, sizeof(text), "Edit here value %02d", i);
        edit_text(edit, text);
        layout_label(layout, label, 0, i);
        layout_edit(layout, edit, 1, i);
    }

    for (i = 0; i < n - 1; ++i)
        layout_vmargin(layout, i, 3);

    layout_hmargin(layout, 0, 5);
    layout_margin4(layout, 10, 10, 10, 10);
    panel_layout(panel, layout);
    window_panel(window, panel);
    return window;
}
```

- Las líneas 3-6 crean la ventana, el panel y el layout.
- El bucle 8-19 añade varias etiquetas y cajas de edición al layout.
- El bucle 21-22 establece una pequeña separación entre filas.
- Las líneas 24-25 establecen una separación entre columnas y un margen en el borde.
- Las líneas 26-27 vinculan el layout con el panel y este con la ventana.

El resultado de este código es el *"Dimensionado naturalDimensionado natural"* (Página 336) del panel (Figura 18.64), que establece por defecto una anchura de 100 píxeles para los controles de edición. Las etiquetas se ajustan al texto que contienen. También se han aplicado las separaciones y márgenes.

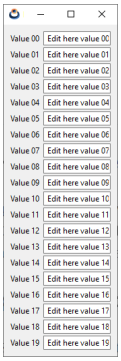


Figura 18.64: Dimensionado natural del panel definido en (Listado 18.9).

En este caso es posible redimensionar la ventana, ya que hemos utilizado el flag `ekWINDOW_STDRES` al crearla (Figura 18.65).

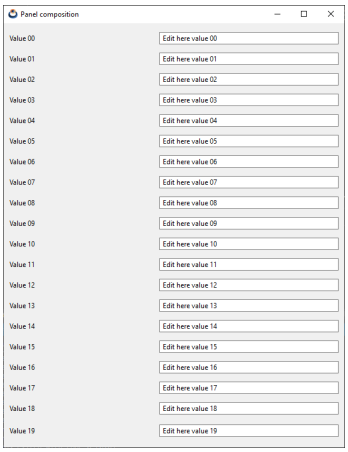


Figura 18.65: Comportamiento del panel cuando crece la ventana.

Este comportamiento puede que no sea el más apropiado para el caso que nos ocupa. Por defecto, el layout realiza la *"Expansión de celdasExpansión de celdas"* (Página 341) de manera proporcional. Pero lo que realmente buscamos es "estirar" los controles de edición y que las filas mantengan su altura por defecto (Listado 18.10).

Listado 18.10: Cambio en la expansión horizontal y vertical.

```
Layout *layout = layout_create(2, n + 1);
...
layout_hexpand(layout, 1);
layout_vexpand(layout, n);
```

Las líneas anteriores provocan que la expansión horizontal recaiga exclusivamente sobre la columna 1 (la de los EditBox). Por otro lado, se ha creado una fila extra vacía, volcando en ella toda la expansión vertical (Figura 18.66).

Si bien el panel se comporta ahora correctamente ante el crecimiento de la ventana, te-

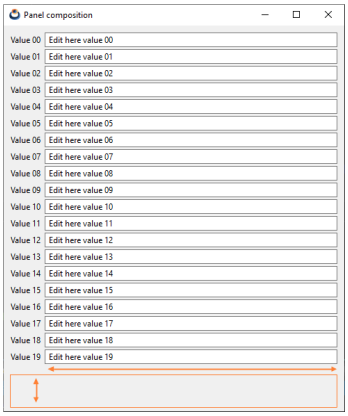


Figura 18.66: Comportamiento deseado, ante la expansión de la ventana.

nemos dificultades cuando queremos "encogerla" por debajo de cierto límite (Figura 18.67). Esto es debido a que el dimensionado natural impone un tamaño mínimo, ya que llega un momento que es imposible reducir los controles asociados al layout.

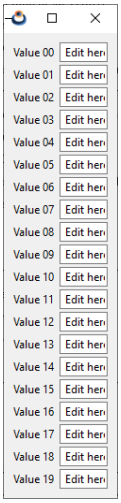


Figura 18.67: Tamaño mínimo del panel.

Esto puede suponer un problema, ya que es posible que tengamos paneles suficientemente grandes que excedan incluso el tamaño del monitor y no puedan mostrarse en su totalidad. Para solucionarlo podemos establecer un tamaño por defecto para todo el panel (Listado 18.11), que será el que muestre al iniciar la ventana (Figura 18.68).

Listado 18.11: Tamaño por defecto del panel.

```
...
panel_size(panel, s2df(400, 300));
...
```

Este comando desvincula, en cierta manera, el tamaño del panel del tamaño de su

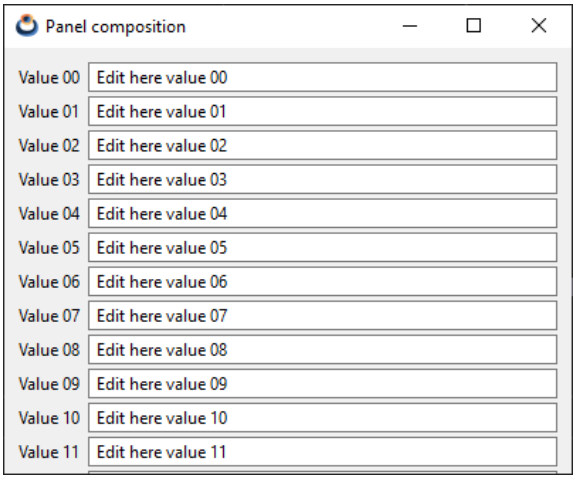


Figura 18.68: Dimensionado natural, forzado a 400x300.

contenido. De esta forma, el Layout tiene libertad para reducir el tamaño de la vista, independientemente de si puede o no mostrar la totalidad del contenido (Figura 18.69).

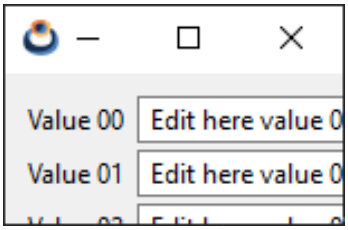


Figura 18.69: Reducción de los límites del panel.

Y ya por último, si queremos, podemos crear el panel con barras de scroll (Listado 18.12) y desplazarnos por el contenido no visible (Figura 18.70).

Listado 18.12: Panel con barras de scroll.

```
...
Panel *panel = panel_scroll(TRUE, TRUE);
...
```

Y, por supuesto, todo lo dicho funcionará de la misma forma en cualquier plataforma .

18.19. Window

Los objetos **Window** son los contenedores de más alto nivel dentro de la interfaz de usuario (Figura 18.72). Están compuestos por la barra de título, donde se ubican los botones de cerrar, maximizar y minimizar, la zona interior y el marco. Si la ventana admite redimensionado, dicho marco podrá ser arrastrado con el ratón para cambiar su tamaño. La zona interior o área cliente de la ventana, se configura mediante un *"Panel"* (Página 344)

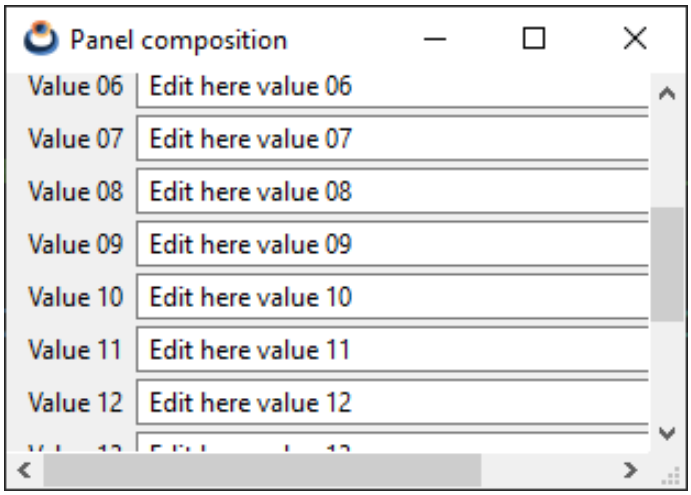


Figura 18.70: Panel con barras de scroll.

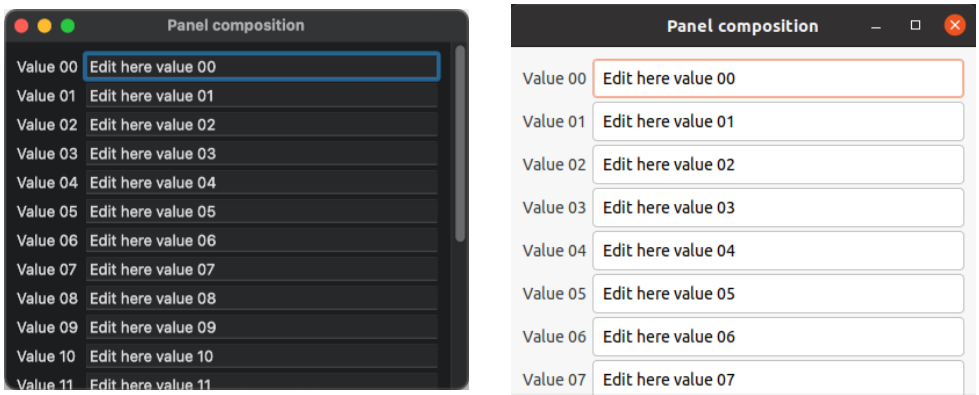


Figura 18.71: Nuestro panel funcionando en macOS y Linux.

principal. En *"¡Hola Mundo!"* (Página 23) tienes un ejemplo sencillo de composición y muestra de una ventana.

- Utiliza `window_create` para crear una ventana.
- Utiliza `window_panel` para asignar el panel principal.
- Utiliza `window_show` para mostrar una ventana.
- Utiliza el flag `ekWINDOW_TITLE` para incluir barra de título.
- Utiliza `window_title` para asignar un título.

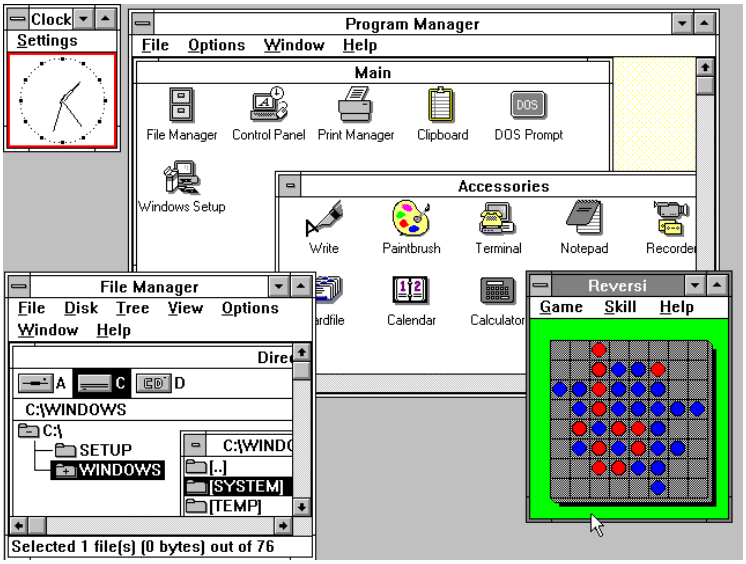


Figura 18.72: El concepto de ventana aparece desde los primeros sistemas de escritorio.

18.19.1. Tamaño de la ventana

En principio, el tamaño de la ventana se calcula automáticamente en función del *"Dimensionado naturalDimensionado natural"* (Página 336) de su panel principal, pero puede ser alterado en cualquier momento.

- Utiliza `window_size` para cambiar el tamaño del panel principal.
- Utiliza el flag `ekWINDOW_MAX` para incluir el botón de maximizar en la barra de título.
- Utiliza el flag `ekWINDOW_MIN` para incluir el botón de minimizar en la barra de título.
- Utiliza el flag `ekWINDOW_RESIZE` para crear una ventana con bordes redimensionables.

El cambio en las dimensiones del área cliente lleva implícita una re-ubicación y redimensionado de los controles interiores. Esto se gestiona automáticamente por los objetos `Layout` en función de como se haya configurado su *"Expansión de celdasExpansión de celdas"* (Página 341), que se propagará de manera recursiva por todos los sublayouts. En *"Die"* (Página 395) tienes un ejemplo del redimensionado de una ventana (Figura 18.73).

18.19.2. Cierre de la ventana

Normalmente una ventana se cierra pulsando el botón `[X]` ubicado a la derecha de la barra de título. Pero, en ocasiones, puede ser útil cerrarla también mediante la teclas

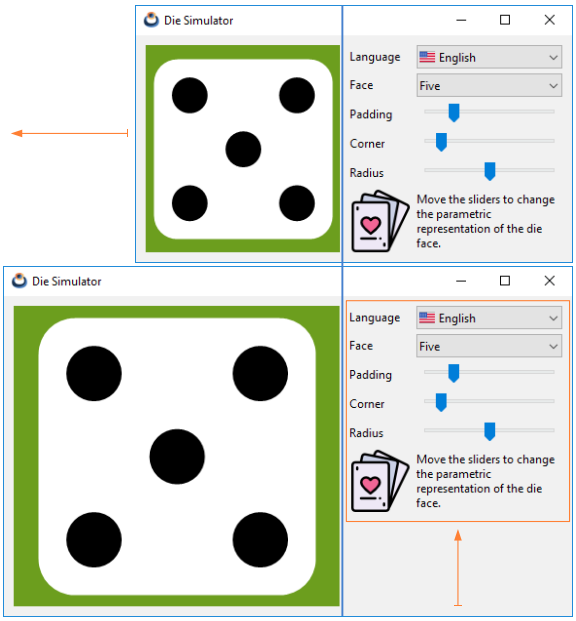


Figura 18.73: Redimensionado de la ventana en la demo **Die**.

`[ENTER]` o `[ESC]`. Cerrar una ventana implica ocultarla, pero no destruirla. Es decir, podemos volver a mostrar una ventana ya cerrada utilizando `window_show`. En el caso de que el cierre esté condicionado a un estado de la aplicación, como guardar un archivo por ejemplo, deberemos asignar un manejador mediante `window_OnClose` y decidir ahí si se cierra o no.

- Utiliza `window_hide` para ocultar una ventana.
- Utiliza `window_destroy` para destruir definitivamente una ventana.
- Utiliza el flag `ekWINDOW_CLOSE` para incluir el botón de cierre en la barra de título.
- Utiliza el flag `ekWINDOW_RETURN` para habilitar el cierre mediante `[ENTER]`.
- Utiliza el flag `ekWINDOW_ESC` para habilitar el cierre mediante `[ESC]`.
- Utiliza el flag `window_OnClose` para evitar el cierre de una ventana (Listado 18.13).

Listado 18.13: Evita el cierre de la ventana.

```
static void i_OnClose(App *app, Event *e)
{
    const EvWinClose *params = event_params(e, EvWinClose);
    if (can_close(app, params->origin) == FALSE)
    {
        bool_t *result = event_result(e, bool_t);
        *result = FALSE;
    }
}
```

```
...
window_OnClose(window, listener(app, i_OnClose, App));
```

Destruir una ventana destruye implícitamente todos sus elementos y controles internos.

18.19.3. Ventanas modales

Son aquellas que, al ser lanzadas, bloquean a la ventana anterior (o padre) hasta que no se produzca el cierre de la misma (Figura 18.74). El ser o no "modal" no es una característica de la ventana en sí, si no del modo de lanzarla. En *"¡Hola Modal Window!¡Hola Modal Window!"* (Página 527) tienes un ejemplo de uso.

- Utiliza `window_modal` para mostrar una ventana en modo modal.
- Utiliza `window_stop_modal` para ocultarla y detener el ciclo modal.

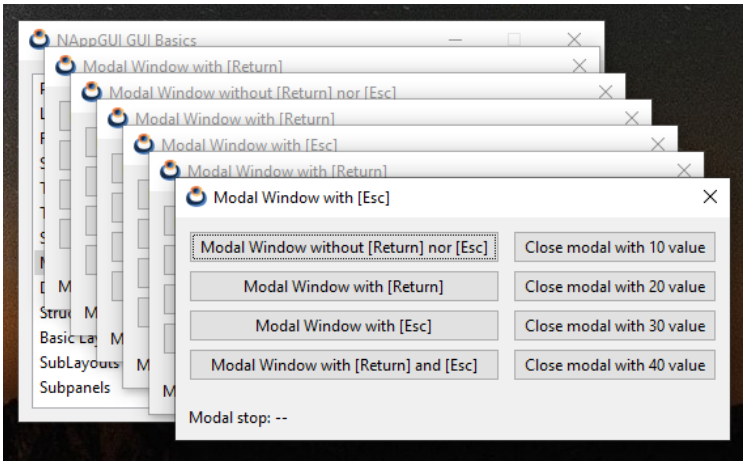


Figura 18.74: Varias ventanas modales.

Tras la llamada a `window_modal`, el programa queda detenido en este punto, a la espera del cierre de la ventana, que podrá realizarse mediante `[X]`, `[ENTER]`, `[ESC]` o llamando a `window_stop_modal` (Listado 18.14). El valor retornado por esta función será:

- `ekGUI_CLOSE_ESC` (1). Si la ventana modal se cerró pulsando `[ESC]`.
- `ekGUI_CLOSE_INTRO` (2). Si la ventana modal se cerró pulsando `[ENTER]`.
- `ekGUI_CLOSE_BUTTON` (3). Si la ventana modal se cerró pulsando `[X]`.
- El valor indicado en `window_stop_modal`.

Listado 18.14: Uso de ventanas modales.

```
static void i_OnAcceptClick(Window *window, Event *e)
{
    window_stop_modal(window, 300);
}

Window *window = i_create_window_with_accept_button();
// The program will stop HERE until window is closed
uint32_t ret = window_modal(window);

if (ret == 1)
{
    // Closed by ESC
}
else if (ret == 2)
{
    // Closed by INTRO
}
else if (ret == 3)
{
    // Closed by [X]
}
else if (ret == 300)
{
    // Closed by window_stop_modal
}

window_destroy(&window);
```

18.19.4. Teclas de acceso directo

Normalmente, el foco del teclado estará fijado en algún control del interior de la ventana como `Edit`, `Button` o `View`. Pero es posible que queramos definir acciones globales asociadas a alguna tecla concreta.

- Utilizar `window_hotkey` para asignar una acción a una tecla.

Las *hotkeys* tendrán prioridad sobre el foco del teclado (Figura 18.75). Es decir, si tenemos una acción vinculada con la tecla `[F9]` está se ejecutará en el momento que se pulse la tecla y el evento `ekGUI_EVENT_KEYDOWN`(F9) no será recibido por el control que disponga del foco.

18.20. GUI Data binding

Por **GUI Data Binding** entendemos el mapeo automático entre las variables del programa y los controles de la interfaz de usuario (Figura 18.76). De esta forma ambos estarán sincronizados sin que el programador tenga que realizar ningún trabajo extra como la captura de eventos, asignación de valores, comprobación de rangos, etc. En *"¡Hola Gui*

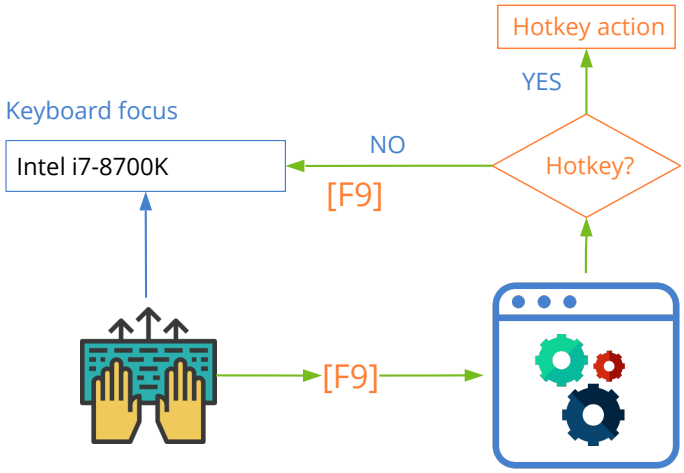


Figura 18.75: Procesamiento de un atajo de teclado.

Binding!¡Hola Gui Binding!" (Página 532) tienes el código fuente completo del ejemplo que mostraremos a continuación.

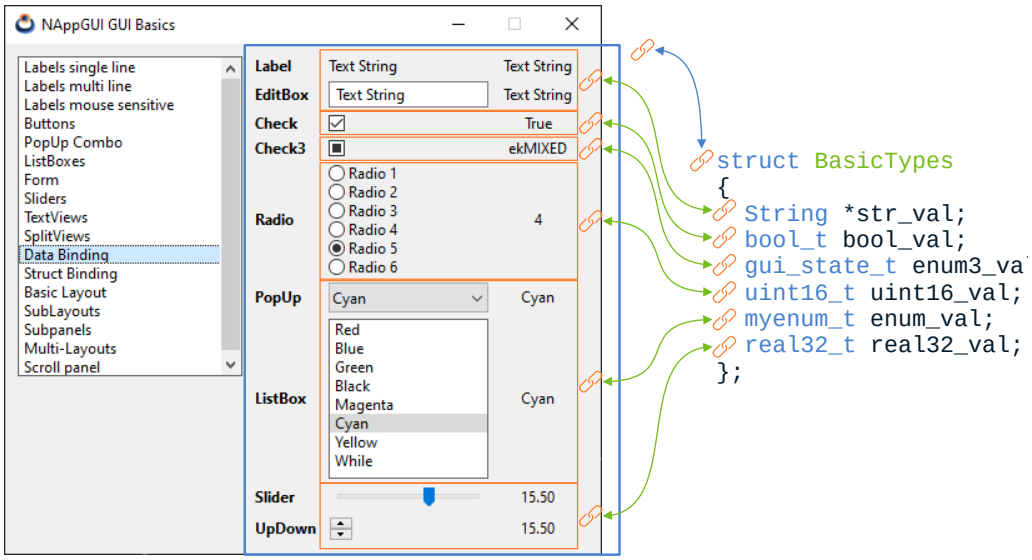


Figura 18.76: Sincronización automática de datos con la interfaz de usuario.

18.20.1. Vinculación de tipos básicos

Partimos de una estructura de datos compuesta de varios campos de tipos básicos (Listado 18.15), donde no hay anidadas otras estructuras u objetos.

Listado 18.15: Sencillo modelo de datos.

```
typedef struct _basictypes_t BasicTypes;

typedef enum _myenum_t
{
    ekRED,
    ekBLUE,
    ekGREEN,
    ekBLACK,
    ekMAGENTA,
    ekCYAN,
    ekYELLOW,
    ekWHITE
} myenum_t;

struct _basictypes_t
{
    bool_t bool_val;
    uint16_t uint16_val;
    real32_t real32_val;
    myenum_t enum_val;
    gui_state_t enum3_val;
    String *str_val;
};
```

Lo primero que debemos hacer es registrar los campos de la estructura con dbind (Listado 18.16):

Listado 18.16: Registro en dbind de los campos de la estructura.

```
dbind_enum(gui_state_t, ekGUI_OFF, "");
dbind_enum(gui_state_t, ekGUI_ON, "");
dbind_enum(gui_state_t, ekGUI_MIXED, "");
dbind_enum(myenum_t, ekRED, "Red");
dbind_enum(myenum_t, ekBLUE, "Blue");
dbind_enum(myenum_t, ekGREEN, "Green");
dbind_enum(myenum_t, ekBLACK, "Black");
dbind_enum(myenum_t, ekMAGENTA, "Magenta");
dbind_enum(myenum_t, ekCYAN, "Cyan");
dbind_enum(myenum_t, ekYELLOW, "Yellow");
dbind_enum(myenum_t, ekWHITE, "While");
dbind(BasicTypes, bool_t, bool_val);
dbind(BasicTypes, uint16_t, uint16_val);
dbind(BasicTypes, real32_t, real32_val);
dbind(BasicTypes, gui_state_t, enum3_val);
dbind(BasicTypes, myenum_t, enum_val);
dbind(BasicTypes, String*, str_val);
dbind_range(BasicTypes, real32_t, real32_val, -50, 50);
dbind_increment(BasicTypes, real32_t, real32_val, 5);
```

> *`DBind` es un registro, dentro de la aplicación, que permite automatizar ciertas operaciones sobre los datos, así como establecer rangos, precisiones o alias. Su uso va más allá de las interfaces gráficas de usuario. Más información en "Data binding" (Página 230).*

Por otro lado, construimos un *"Layout"* (Página 335) que alberga los diferentes controles de la interfaz de usuario (Listado 18.17):

Listado 18.17: Controles de interfaz organizados en un layout (Figura 18.76).

```
static Layout *i_layout(void)
{
    Layout *layout = layout_create(3, 9);
    Label *label = label_create();
    Edit *edit = edit_create();
    Button *check = button_check();
    Button *check3 = button_check3();
    Layout *radios = i_radio_layout();
    PopUp *popup = popup_create();
    ListBox *listbox = listbox_create();
    Slider *slider = slider_create();
    UpDown *updown = updown_create();
    layout_label(layout, label, 1, 0);
    layout_edit(layout, edit, 1, 1);
    layout_button(layout, check, 1, 2);
    layout_button(layout, check3, 1, 3);
    layout_layout(layout, radios, 1, 4);
    layout_popup(layout, popup, 1, 5);
    layout_listbox(layout, listbox, 1, 6);
    layout_slider(layout, slider, 1, 7);
    layout_updown(layout, updown, 1, 8);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_halign(layout, 1, 8, ekLEFT);
    return layout;
}
```

Ahora vincularemos las celdas de nuestro layout con los campos de la estructura (Listado 18.18). Presta atención a que aún **no hemos creado ningún objeto** del tipo `BasicTypes`. Por lo tanto, se trata de un enlace semántico donde no intervienen posiciones de memoria, sino los desplazamientos (offset) de los campos dentro de la estructura de datos.

- Utiliza `cell_dbind` para vincular un campo con una celda individual.
- Utiliza `layout_dbind` para vincular una estructura con un layout.
- Utiliza `layout_cell` para obtener una celda de un Layout.

Listado 18.18: Vinculación de variables con celdas del layout.

```
cell_dbind(layout_cell(layout, 1, 0), BasicTypes, String*, str_val);
cell_dbind(layout_cell(layout, 1, 1), BasicTypes, String*, str_val);
cell_dbind(layout_cell(layout, 1, 2), BasicTypes, bool_t, bool_val);
cell_dbind(layout_cell(layout, 1, 3), BasicTypes, gui_state_t, enum3_val);
cell_dbind(layout_cell(layout, 1, 4), BasicTypes, uint16_t, uint16_val);
cell_dbind(layout_cell(layout, 1, 5), BasicTypes, myenum_t, enum_val);
cell_dbind(layout_cell(layout, 1, 6), BasicTypes, myenum_t, enum_val);
cell_dbind(layout_cell(layout, 1, 7), BasicTypes, real32_t, real32_val);
cell_dbind(layout_cell(layout, 1, 8), BasicTypes, real32_t, real32_val);
layout_dbind(layout, NULL, BasicTypes);
```

Al vincular una estructura de datos con `layout_dbind` debemos tener presente que las celdas de dicho layout **solo pueden asociarse con campos de la misma estructura**. De lo contrario, obtendremos un error en tiempo de ejecución, debido a la incoherencia de datos que se produciría. Dicho de otro modo, no podemos mezclar estructuras dentro de un mismo layout.

No se pueden utilizar variables aisladas en el Data Binding. Todas deben pertenecer a un `struct` ya que, internamente, se establecen las relaciones (Layout –>Estructura) y (Cell –>Campo o Variable).

Ya, por último, asociaremos un objeto de tipo `BasicTypes` con el layout creado anteriormente (Listado 18.19).

- Utiliza `layout_dbind_obj` para vincular un objeto con la interfaz de usuario.

Listado 18.19: Vinculación de un objeto con la interfaz.

```
BasicTypes *data = heap_new(BasicTypes);
data->bool_val = TRUE;
data->uint16_val = 4;
data->real32_val = 15.5f;
data->enum3_val = ekGUI_MIXED;
data->enum_val = ekCYAN;
data->str_val = str_c("Text String");
layout_dbind_obj(layout, data, BasicTypes);
```

- Puedes cambiar el objeto que se está “editando” en cualquier momento, con una nueva llamada a `layout_dbind_obj`.

- Si pasamos `NULL` a `layout_dbind_obj` se deshabilitarán las celdas vinculadas con campos de la estructura.

18.20.2. Límites y rangos

Ten presente que la capacidad expresiva de los controles estará, en general, muy por debajo del rango de valores soportados por los tipos de datos (Listado 18.20). Por ejemplo, si vinculamos un `uint16_t` con un `RadioGroup` este último solo soportará valores entre `0` y `n-1`, donde `n` es el número total de radios. Los controles están preparados para manejar los valores fuera de rango de la forma más coherente posible, pero esto no exime al programador de hacer las cosas bien. En (Tabla 18.5) tienes un resumen de los tipos de datos y rangos soportados por los controles estándar.

Listado 18.20: Valor no representable en el RadioGroup de (Figura 18.76).

```
data->uint16_val = 1678;
cell_dbind(layout_cell(layout, 1, 4), BasicTypes, uint16_t, uint16_val);
```

Control	Data Type
"Label" (Página 309)	String, Number, Enum
"Edit" (Página 313)	String, Number
"Button" (Página 310) (CheckBox)	Boolean
"Button" (Página 310) (CheckBox3)	Enum (3 values), Integer (0,1,2)
"RadioGroupRadioGroup" (Página 311)	Enum, Integer (0,1,2...n-1)
"PopUp" (Página 313)	Enum, Integer (0,1,2...n-1)
"ListBox" (Página 316)	Enum, Integer (0,1,2...n-1)
"Slider" (Página 317)	Number (min..max)
"UpDown" (Página 317)	Enum, Number

Tabla 18.5: Tipos de datos y rangos de los controles GUI.

18.20.3. Estructuras anidadas

Veamos ahora un modelo de datos algo más complicado, que incluye estructuras anidadas además de los tipos básicos (Figura 18.77). En este caso contamos con una estructura denominada `StructTypes` que contiene instancias de otra estructura denominada `Vector` (Listado 18.21). El código fuente completo de este segundo ejemplo lo tienes en *"¡Hola Struct Binding!¡Hola Struct Binding!"* (Página 536).

Listado 18.21: Modelo de datos con estructuras anidadas y registro en dbind.

```
typedef struct _vector_t Vector;
typedef struct _structtypes_t StructTypes;
```

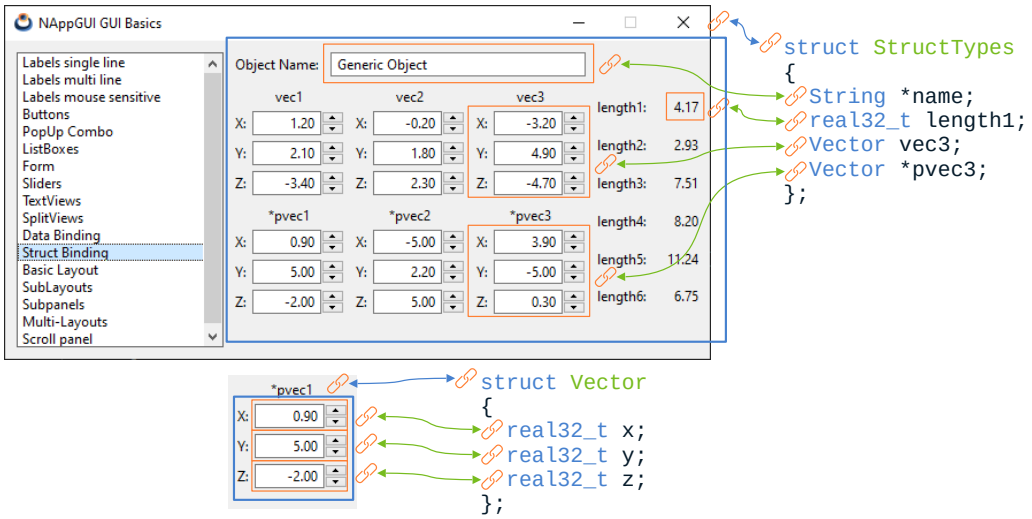


Figura 18.77: Vinculación de datos con sub-estructuras.

```
struct _vector_t
{
    real32_t x;
    real32_t y;
    real32_t z;
};

struct _structtypes_t
{
    String *name;
    Vector vec1;
    Vector vec2;
    Vector vec3;
    Vector *pvec1;
    Vector *pvec2;
    Vector *pvec3;
    real32_t length1;
    real32_t length2;
    real32_t length3;
    real32_t length4;
    real32_t length5;
    real32_t length6;
};

dbind(Vector, real32_t, x);
dbind(Vector, real32_t, y);
dbind(Vector, real32_t, z);
dbind(StructTypes, String*, name);
dbind(StructTypes, Vector, vec1);
dbind(StructTypes, Vector, vec2);
```

```
dbind(StructTypes, Vector, vec3);
dbind(StructTypes, Vector*, pvec1);
dbind(StructTypes, Vector*, pvec2);
dbind(StructTypes, Vector*, pvec3);
dbind(StructTypes, real32_t, length1);
dbind(StructTypes, real32_t, length2);
dbind(StructTypes, real32_t, length3);
dbind(StructTypes, real32_t, length4);
dbind(StructTypes, real32_t, length5);
dbind(StructTypes, real32_t, length6);
dbind_range(Vector, real32_t, x, -5, 5);
dbind_range(Vector, real32_t, y, -5, 5);
dbind_range(Vector, real32_t, z, -5, 5);
dbind_increment(Vector, real32_t, x, .1f);
dbind_increment(Vector, real32_t, y, .1f);
dbind_increment(Vector, real32_t, z, .1f);
```

Empezamos con la misma metodología que empleamos con el primer ejemplo. Creamos un layout y lo vinculamos con la estructura `Vector` (Listado 18.22). Esto no presenta problemas, al estar compuesta exclusivamente por tipos básicos `real32_t`.

Listado 18.22: Layout para editar objetos de tipo `Vector`.

```
static Layout *i_vector_layout(void)
{
    Layout *layout = layout_create(3, 3);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    Edit *edit3 = edit_create();
    UpDown *updown1 = updown_create();
    UpDown *updown2 = updown_create();
    UpDown *updown3 = updown_create();
    label_text(label1, "X:");
    label_text(label2, "Y:");
    label_text(label3, "Z:");
    edit_align(edit1, ekRIGHT);
    edit_align(edit2, ekRIGHT);
    edit_align(edit3, ekRIGHT);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 2);
    layout_edit(layout, edit1, 1, 0);
    layout_edit(layout, edit2, 1, 1);
    layout_edit(layout, edit3, 1, 2);
    layout_updown(layout, updown1, 2, 0);
    layout_updown(layout, updown2, 2, 1);
    layout_updown(layout, updown3, 2, 2);
    cell_dbind(layout_cell(layout, 1, 0), Vector, real32_t, x);
```

```
    cell_dbind(layout_cell(layout, 1, 1), Vector, real32_t, y);
    cell_dbind(layout_cell(layout, 1, 2), Vector, real32_t, z);
    cell_dbind(layout_cell(layout, 2, 0), Vector, real32_t, x);
    cell_dbind(layout_cell(layout, 2, 1), Vector, real32_t, y);
    cell_dbind(layout_cell(layout, 2, 2), Vector, real32_t, z);
    layout_dbind(layout, NULL, Vector);
    return layout;
}
```

La idea ahora es utilizar esta función para crear *"Sub-layoutsSub-layouts"* (Página 340) y asociarlos a celdas de un layout de más alto nivel, que pueda soportar objetos de tipo `StructTypes` (Listado 18.23). Los sub-layout de tipo `Vector` se vinculan con los campos `{ Vector vec1, Vector *pvec1, ... }` utilizando `cell_dbind`, de forma similar a como lo hicimos con los tipos básicos.

Listado 18.23: Layout que soporta objetos de tipo `StructTypes`.

```
static Layout *i_struct_types_layout(void)
{
    Layout *layout1 = i_create_layout();
    Layout *layout2 = i_vector_layout();
    Layout *layout3 = i_vector_layout();
    Layout *layout4 = i_vector_layout();
    Layout *layout5 = i_vector_layout();
    Layout *layout6 = i_vector_layout();
    Layout *layout7 = i_vector_layout();
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 1, 0);
    layout_layout(layout1, layout4, 2, 0);
    layout_layout(layout1, layout5, 0, 1);
    layout_layout(layout1, layout6, 1, 1);
    layout_layout(layout1, layout7, 2, 1);
    layout_label(layout1, label1, 0, 2);
    layout_label(layout1, label2, 1, 2);
    layout_label(layout1, label3, 2, 2);
    cell_dbind(layout_cell(layout1, 0, 0), StructTypes, Vector, vec1);
    cell_dbind(layout_cell(layout1, 1, 0), StructTypes, Vector, vec2);
    cell_dbind(layout_cell(layout1, 2, 0), StructTypes, Vector, vec3);
    cell_dbind(layout_cell(layout1, 0, 1), StructTypes, Vector*, pvec1);
    cell_dbind(layout_cell(layout1, 1, 1), StructTypes, Vector*, pvec2);
    cell_dbind(layout_cell(layout1, 2, 1), StructTypes, Vector*, pvec3);
    cell_dbind(layout_cell(layout1, 0, 2), StructTypes, real32_t, length1);
    cell_dbind(layout_cell(layout1, 1, 2), StructTypes, real32_t, length2);
    cell_dbind(layout_cell(layout1, 2, 2), StructTypes, real32_t, length3);
    layout_dbind(layout1, NULL, StructTypes);
    return layout1;
}
```

Y ya, por último, solo nos queda vincular objetos de tipo `StructTypes` con el layout principal (Listado 18.24). DBind detectará los sub-layouts de tipo `Vector` y asociará de forma automática los sub-objetos (por valor o por puntero) correspondientes. Por tanto, solo será necesaria una llamada a `layout_dbind_obj` (la del objeto principal).

Listado 18.24: Asociar objeto y sub-objetos a un layout.

```
StructTypes *data = heap_new(StructTypes);
Layout *layout = i_struct_types_layout();
data->name = str_c("Generic Object");
data->pvec1 = heap_new(Vector);
data->pvec2 = heap_new(Vector);
data->pvec3 = heap_new(Vector);
data->vec1 = i_vec_init(1.2f, 2.1f, -3.4f);
data->vec2 = i_vec_init(-0.2f, 1.8f, 2.3f);
data->vec3 = i_vec_init(-3.2f, 4.9f, -4.7f);
*data->pvec1 = i_vec_init(0.9f, 7.9f, -2.0f);
*data->pvec2 = i_vec_init(-6.9f, 2.2f, 8.6f);
*data->pvec3 = i_vec_init(3.9f, -5.5f, 0.3f);
data->length1 = i_vec_length(&data->vec1);
data->length2 = i_vec_length(&data->vec2);
data->length3 = i_vec_length(&data->vec3);
data->length4 = i_vec_length(data->pvec1);
data->length5 = i_vec_length(data->pvec2);
data->length6 = i_vec_length(data->pvec3);

layout_dbind_obj(layout, data, StructTypes);
```

En resumen:

- Para cada sub-estructura creamos un sub-layout, vinculando los campos a nivel local.
- Las celdas que contienen estos sub-layouts se vincularán con la estructura principal.
- Asignamos al layout principal el objeto a editar.

18.20.4. Notificaciones y campos calculados

Si aplicamos lo visto en las secciones anteriores, la sincronización entre datos e interfaz se realiza en estas dos situaciones:

- Cuando el programa llama a `layout_dbind_obj`. En ese momento la interfaz reflejará el estado del objeto.
- Cuando el usuario manipula cualquier control, momento en que se actualizará valor del objeto.

No obstante, es posible que el programa deba ser notificado cuando el usuario modifique el objeto, con el fin de realizar ciertas acciones (actualizar dibujos, guardar datos

en ficheros, lanzar algoritmos de cálculo, etc). Esto se resolverá por medio de eventos, como se refleja en (Figura 18.78). Por otro lado, el programa puede alterar los valores de ciertos campos del objeto y deberá notificar los cambios a la interfaz (layout) para que se mantenga actualizada.

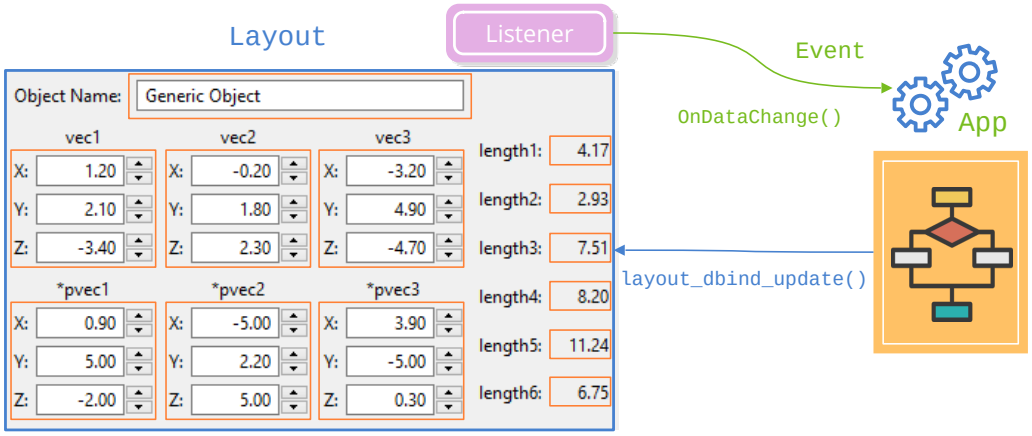


Figura 18.78: Notificación de cambio de valor al programa principal.

- Utiliza `layout_dbind` para incluir un `listener` que notifique los cambios a la aplicación.
- Utiliza `evbind_object` para obtener, dentro del `callback`, el objeto que se está editando.
- Utiliza `event_sender` para obtener, dentro del `callback`, el layout que ha mandado la notificación.
- Utiliza `evbind_modify` para saber, dentro del `callback`, si un campo del objeto ha cambiado o no.
- Utiliza `layout_dbind_update` para notificar al layout que un campo del objeto ha sido modificado por la aplicación.

Todo esto puede verse reflejado en (Listado 18.25). Cada vez que el usuario cambie cualquier valor de `StructTypes` se lanzará una notificación de tipo `ekGUI_EVENT_OBJCHANGE` que comprobará si el campo `vec1` ha cambiado. En caso afirmativo, se recalculará su longitud y se actualizarán los controles GUI asociados con dicha variable.

Listado 18.25: Notificación de modificación de valores del objeto.

```
static void i_OnDataChange(App *app, Event *e)
{
    StructTypes *data = evbind_object(e, StructTypes);
    Layout *layout = event_sender(e, Layout);
    cassert(event_type(e) == ekGUI_EVENT_OBJCHANGE);
```

```
    if (evbind_modify(e, StructTypes, Vector, vec1) == TRUE)
    {
        app_update_drawing(app);
        data->length1 = i_vec_length(&data->vec1);
        layout_dbind_update(layout, StructTypes, real32_t, length1);
    }
}

layout_dbind(layout, listener(app, i_OnDataChange, App), StructTypes);
```

Si, por algún motivo, el valor modificado no es admisible por la aplicación, se puede revertir devolviendo `FALSE` como resultado del evento (Listado 18.26).

Listado 18.26: Anulando los cambios realizados por el usuario.

```
static void i_OnDataChange(App *app, Event *e)
{
    StructTypes *data = evbind_object(e, StructTypes);
    Layout *layout = event_sender(e, Layout);

    if (evbind_modify(e, StructTypes, Vector, vec1) == TRUE)
    {
        real32_t length = i_vec_length(&data->vec1);
        if (length < 5.f)
        {
            app_update_drawing(app);
            data->length1 = length;
            layout_dbind_update(layout, StructTypes, real32_t, length1);
        }
        else
        {
            // This will REVERT the changes in 'vec1' variable
            bool_t *res = event_result(e, bool_t);
            *res = FALSE;
        }
    }
}
```

18.21. Menu

Un **Menu** no es más que un contenedor (o ventana) que integra una serie de opciones, también llamadas Items o **MenuItems** (Figura 18.79). Cada una de ellas consta de un texto corto, opcionalmente un icono y opcionalmente también un atajo de teclado, como por ejemplo el clásico `Ctrl+C/Ctrl+V` para copiar y pegar. Adicionalmente, un item puede albergar un submenu conformando una jerarquía con diferentes niveles de profundidad. En *"Products"* (Página 437) tienes una aplicación de ejemplo que utiliza menús.

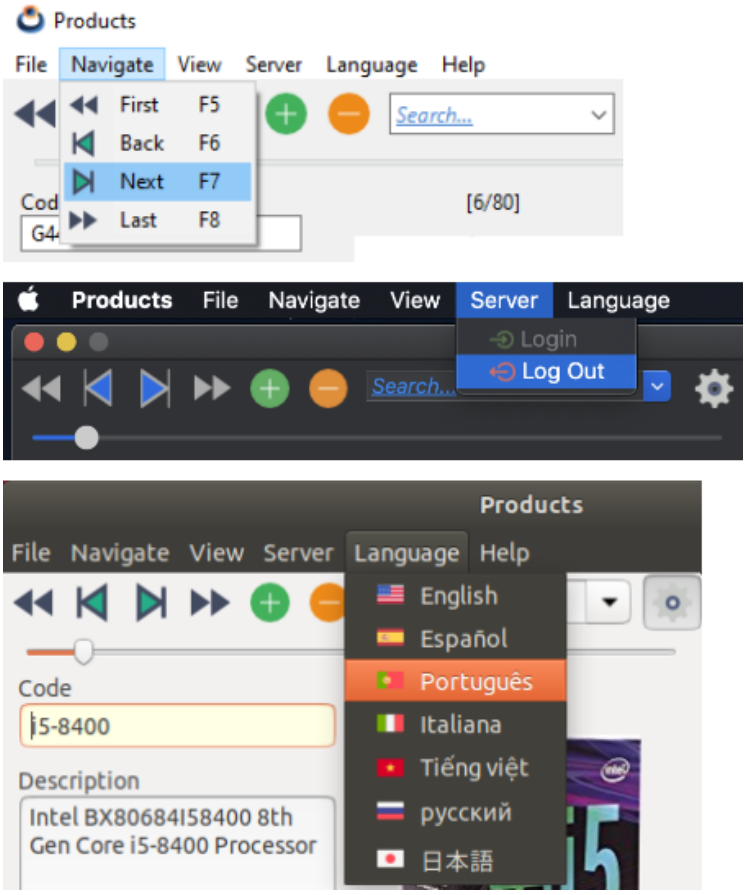


Figura 18.79: Barra de menú en Windows, macOS y Linux.

El concepto de menú, al igual que el de ventana, existe desde el origen de las interfaces gráficas. El primer ordenador en incorporarlas fue el Xerox Alto que apareció en 1973 y su sucesor comercial el Xerox Star. Conceptos aún muy vivos a día de hoy como: Menú, Ventana, Icono, Escritorio, o Ratón ya estaban presentes en estos equipos que sirvieron de inspiración a Steve Jobs en la creación del Apple Lisa (Figura 18.80), precursor del Machintosh e inspirador de Microsoft Windows.

18.22. MenuItem

Representa una opción dentro de un *"Menu"* (Página 366). Tendrán siempre asociada una acción que se ejecutará al activarlos.

- Utiliza `menuitem_create` para crear un item.
- Utiliza `menuitem_text` para asignar un texto.

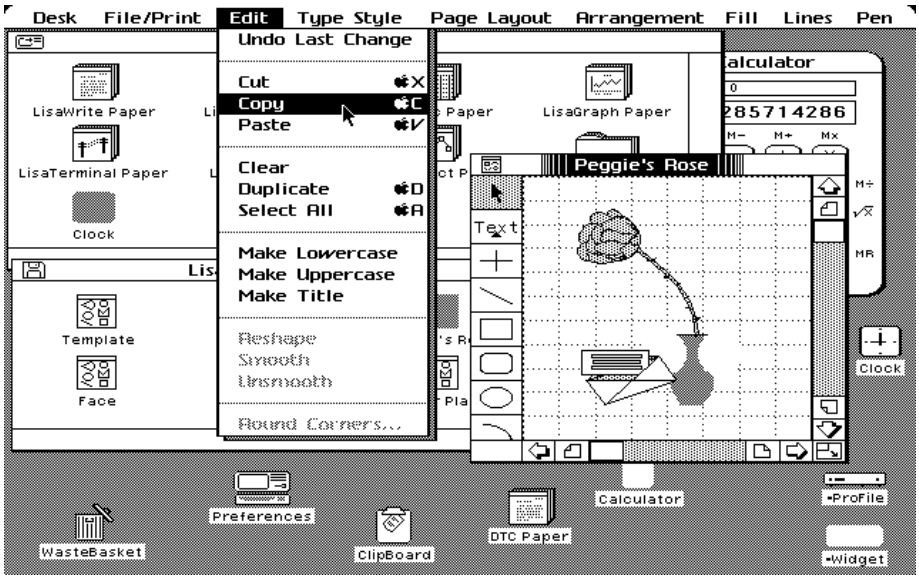


Figura 18.80: El Apple Lisa fue uno de los primeros sistemas en incorporar menús como parte de la interfaz gráfica.

- Utiliza `menuitem_image` para asignar un icono.

18.23. Diálogos comunes

Los diálogos comunes son ventanas predeterminadas proporcionadas por el sistema operativo para realizar tareas cotidianas como: Abrir archivos (Figura 18.81), seleccionar colores (Figura 18.85), tipografías, etc. Su uso es doblemente beneficioso. Por un lado evitamos programarlas como parte de la aplicación y, por otro, aprovechamos el conocimiento previo del usuario ya que seguramente las habrá utilizado en otros programas.

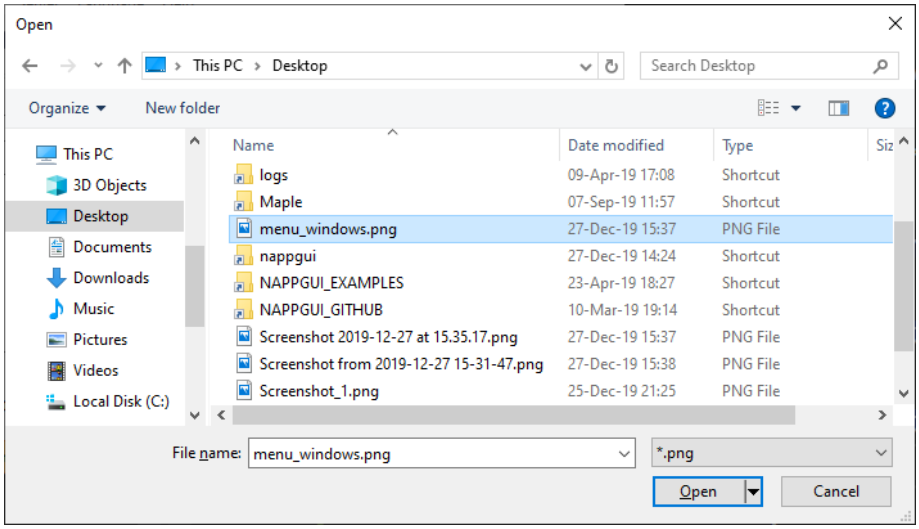


Figura 18.81: Explorador de archivos en Windows.

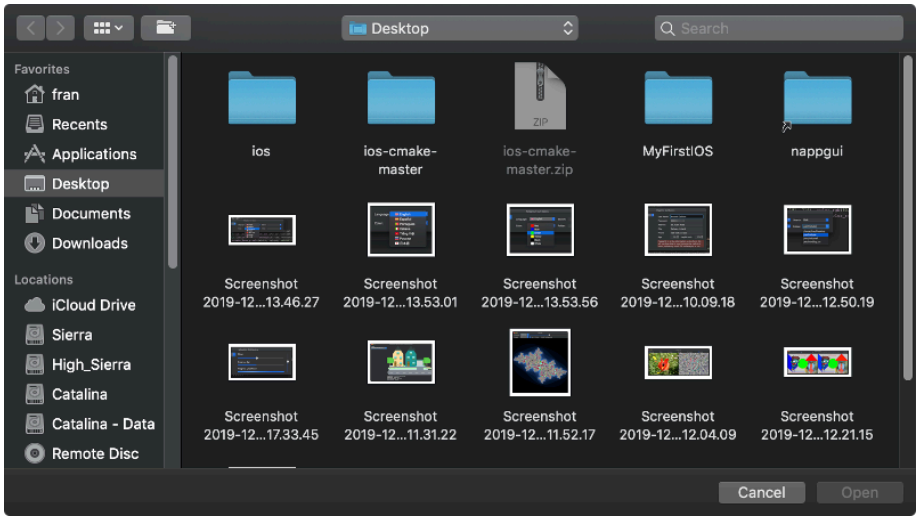


Figura 18.82: Explorador de archivos en macOS.

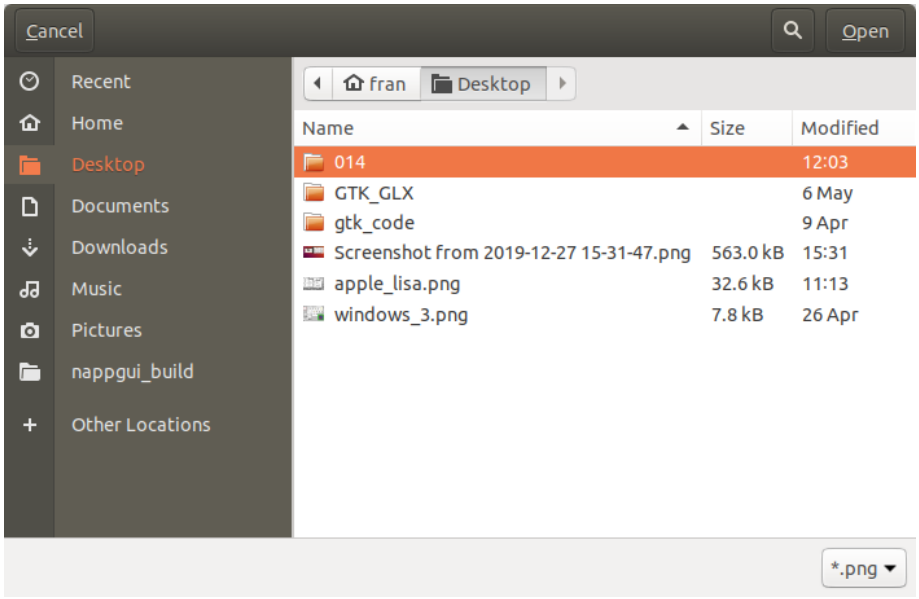


Figura 18.83: Explorador de archivos en Linux.

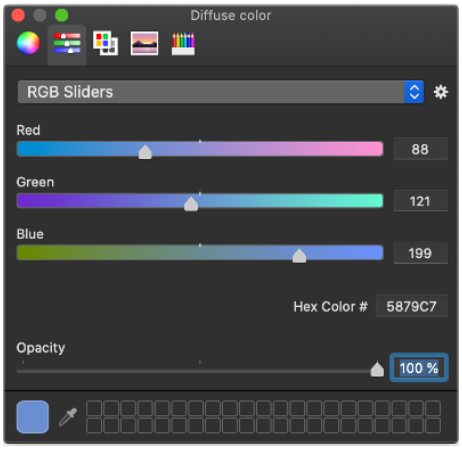


Figura 18.84: Selección de colores en macOS.

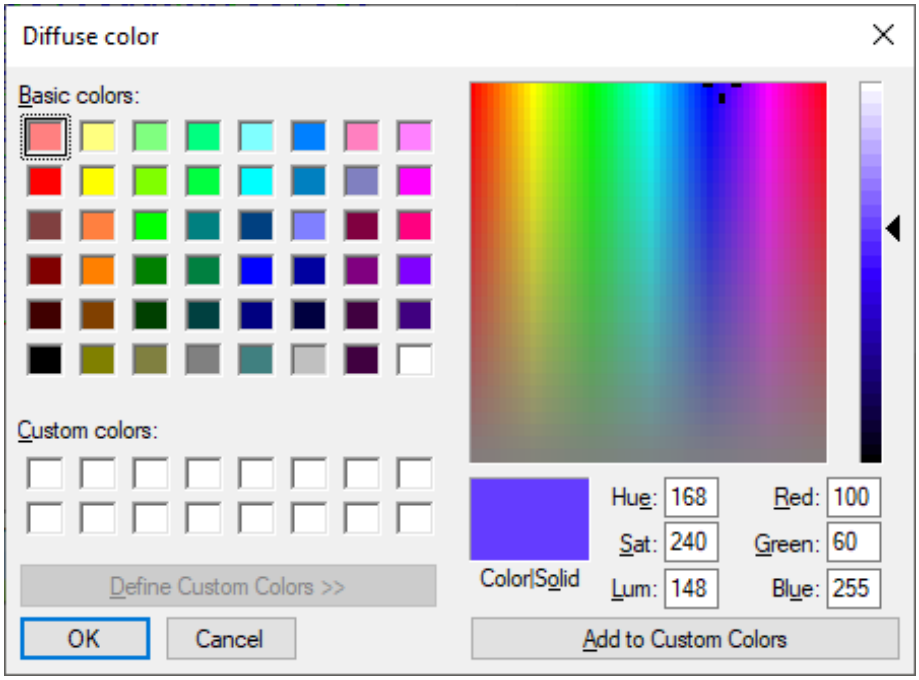


Figura 18.85: Selección de colores en Windows.

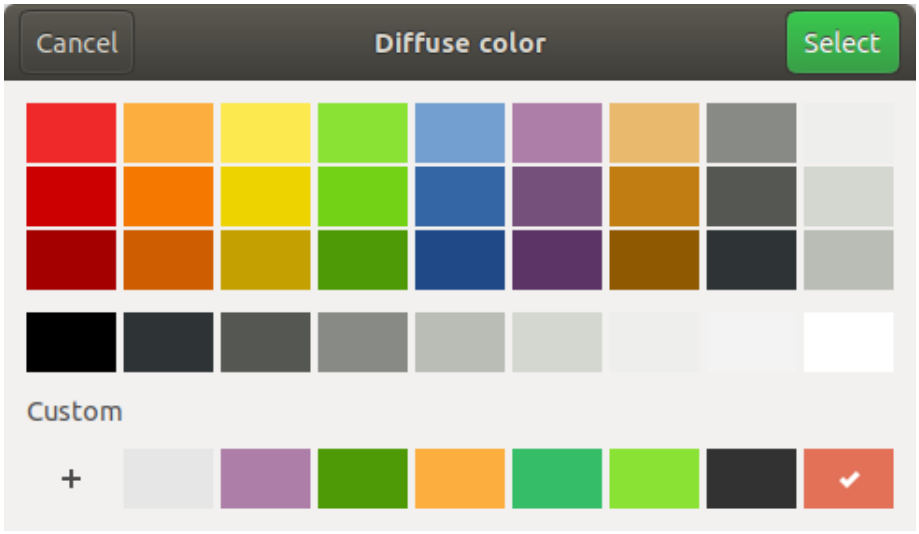


Figura 18.86: Selección de colores en Linux.

Capítulo 19

Librería OSApp

19.1. OSApp

La librería *OSApp* arranca y gestiona el **ciclo de mensajes** de una aplicación de escritorio (Figura 19.1). Si bien la librería **Gui** podría integrarse en aplicaciones existentes mediante un *plugin*, si queremos crear una aplicación desde cero, necesitaremos gestionar los eventos que el sistema operativo envía al programa.

- Utiliza `osmain` para iniciar una aplicación de escritorio.
- Utiliza `osapp_finish` para finalizar una aplicación de escritorio.

19.2. main() y osmain()

La clásica función `main` es el punto de inicio de cualquier programa C/C++ por línea de comandos (Figura 19.2). Su funcionamiento no entraña dificultad alguna y puede resumirse en:

- (1) El sistema operativo carga el programa en memoria y llama a la función `main()` para empezar su ejecución.
- (2) Las sentencias se van ejecutando de manera secuencial y en el orden en el que están escritas. Dicho orden puede alterarse mediante sentencias de control (`for`, `if`, `switch`, etc) o llamadas a función.

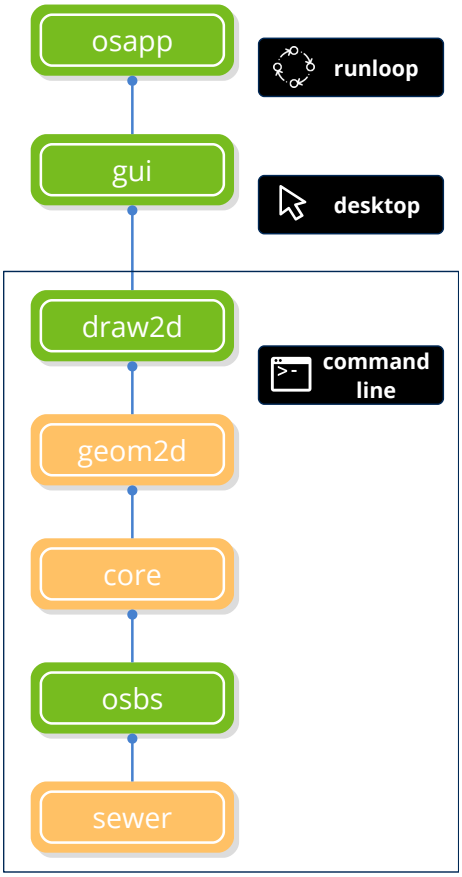


Figura 19.1: Dependencias de *OSApp*. Ver *"NAppGUI API"* (Página 147).

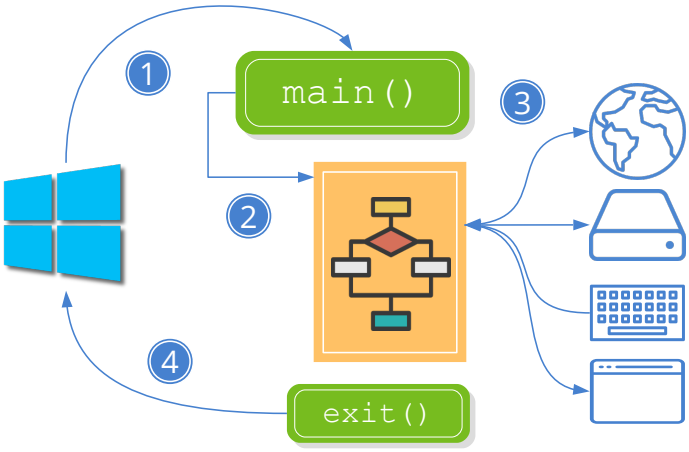


Figura 19.2: Ejecución de una aplicación C de consola.

- ③ Si es necesario realizar entrada/salida, el programa esperará a que termine la comunicación y continuará con la ejecución.

- 4 Cuando se alcance el final de la función `main()` o se ejecute una sentencia `exit()` el programa finalizará y el sistema operativo lo descargará de memoria.

Sin embargo, en aplicaciones de escritorio (dirigidas por eventos), el ciclo de ejecución es un poco más complicado. En esencia, el programa se encuentra continuamente ejecutando un bucle a la espera que el usuario realice alguna acción (Figura 19.3) (Listado 19.1). En *"¡Hola Mundo!"* (Página 23) tienes un ejemplo sencillo:

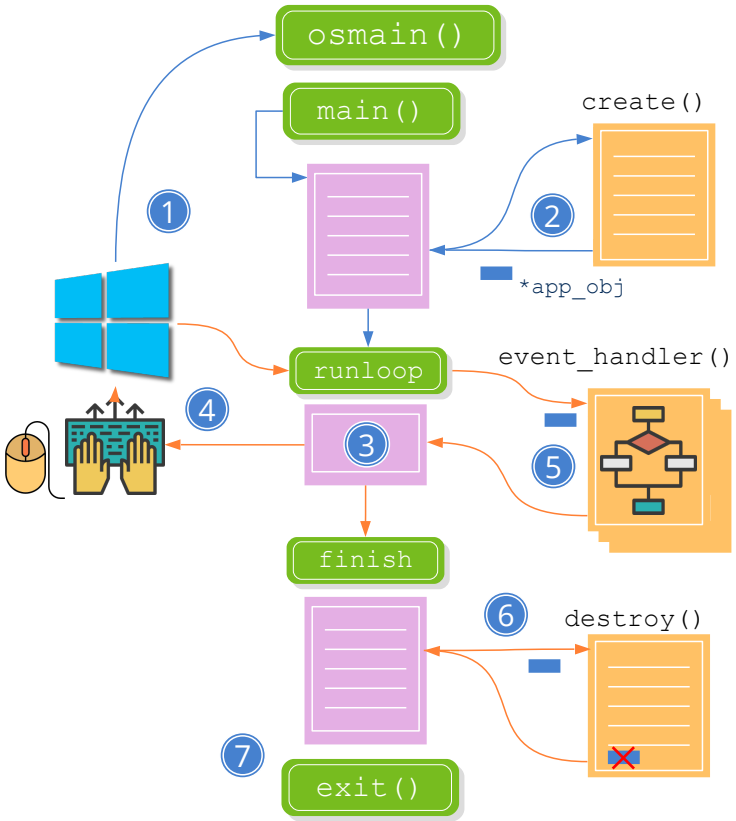


Figura 19.3: Ejecución de una aplicación C de escritorio.

- 1 El sistema operativo carga el programa en memoria y llama a la función `main()`. Ahora está encapsulada dentro de la macro `osmain` que inicia ciertas estructuras necesarias para la captura y gestión de eventos.
- 2 En el algún momento de este proceso inicial, se llamará al constructor de la aplicación (el primer parámetro de `osmain()`) que deberá crear el objeto principal. Dado que el programa está continuamente devolviendo el control al sistema operativo, en este objeto se mantendrá el estado de los datos y de las ventanas.
- 3 Una vez inicializada, la aplicación entrará en un bucle conocido como **ciclo de**

mensajes (Figura 19.4), permaneciendo a la espera de que el usuario realice alguna acción sobre la interfaz del programa.

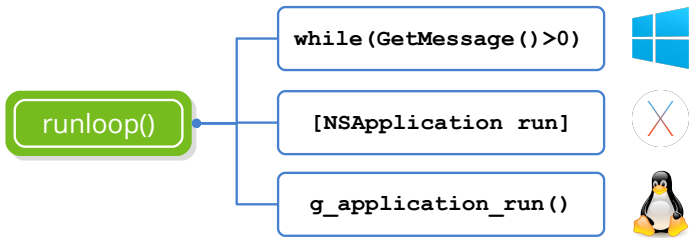


Figura 19.4: Implementación del ciclo de mensajes.

- ④ Cuando esto ocurra, el sistema operativo capturará el evento producido y lo enviará a la aplicación.
- ⑤ Si la aplicación ha definido un manejador para dicho evento, este será invocado y se ejecutará el código de respuesta. Una aplicación puede recibir cientos de mensajes pero solo responderá a los que considere necesarios, ignorando el resto.
- ⑥ Hay un evento especial **de salida** que es generado al llamar a `osapp_finish`. Cuando esto ocurre, `osmain()` empieza a liberar recursos y a preparar una salida limpia. En algún punto se llamará al destructor de la aplicación (segundo parámetro de `osmain()`) para que haga su parte del trabajo, cerrando posibles archivos abiertos y destruyendo el objeto principal.
- ⑦ El sistema operativo descarga la aplicación de la memoria.
- Los bloques de color rosa son dependientes de plataforma y están implementados dentro de NAppGUI.
- Los bloques de color naranja son multiplataforma (totalmente portables) y están implementados dentro de la aplicación.

Listado 19.1: Esqueleto elemental de una aplicación de escritorio.

```
typedef struct _app_t App;
struct _app_t
{
    // Program data
    Window *window;
};

static App* i_create(void)
{
    App *app = heap_new(App);
    // Init program data, GUI and Event handlers
    app->window = ...
    return app;
}
```

```
static void i_destroy(App *app)
{
    // Destroy program data
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

osmain(i_create, i_destroy, "", App);
```

19.3. Aplicaciones síncronas

Cierto tipo de aplicaciones, entre las que se encuentran los videojuegos, reproductores multimedia o simuladores requieren actualizarse a intervalos regulares de tiempo intervenga o no el usuario (Figura 19.5) (Listado 19.2). Para estos casos necesitaremos una variante de `osmain`, que acepte una función de actualización y un intervalo de tiempo. En *"Bricks"* (Página 411) tienes un ejemplo.

- Utiliza `osmain_sync` para iniciar una aplicación síncrona.

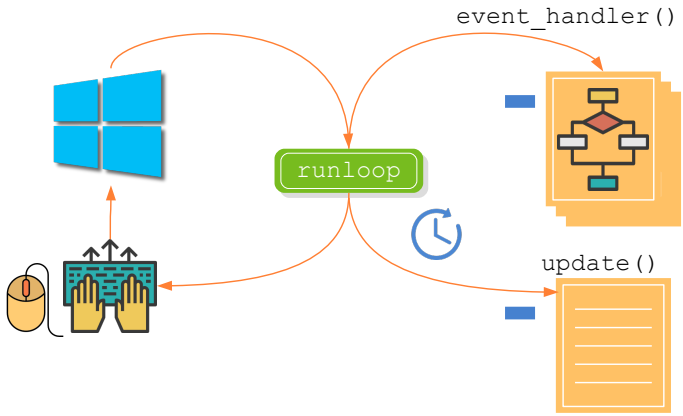


Figura 19.5: Eventos en aplicaciones síncronas.

Listado 19.2: Esqueleto elemental de una aplicación síncrona.

```
typedef struct _app_t App;
struct _app_t
{
    // Program data
    Window *window;
};

static App* i_create(void)
{
    App *app = heap_new(App);
    // Init program data, GUI and Event handlers
    app->window = ...
    return app;
```

```
}

static void i_update(App *app, const real64_t prtime, const real64_t ctime
    ↪ )
{
    // Update program state every 40ms
}

static void i_destroy(App *app)
{
    // Destroy program data
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

osmain_sync(0.04, i_create, i_destroy, i_update, "", App);
```

19.4. Tareas multi-hilo

Tanto las aplicaciones síncronas como asíncronas ejecutan el ciclo de mensajes en un único hilo de la CPU. Esto significa que si, como respuesta a un evento, se debe ejecutar una tarea relativamente lenta la aplicación quedará "congelada" hasta que esta termine (Figura 19.6) **(a)**. Esto producirá un efecto indeseado ya que el programa no responderá durante unos segundos, dando la impresión que se ha bloqueado. La solución es lanzar una tarea en paralelo (Figura 19.6) **(b)** (Listado 19.3) que libere rápidamente el hilo que gestiona el GUI. En *"Login multi-hiloLogin multi-hilo"* (Página 452) tienes un ejemplo del uso de tareas.

- Utiliza `osapp_task` para lanzar una nueva tarea en un hilo paralelo.

Listado 19.3: Nueva tarea en un hilo paralelo.

```
// Runs in new thread
static uint32_t i_task_main(TaskData *data)
{
    // Do the task work here!
}

// Runs in GUI thread
static void i_task_update(TaskData *data)
{
    // Update the GUI here!
}

// Runs in GUI thread
static void i_task_end(TaskData *data, const uint32_t rvalue)
{
    // Finish task code here!
```

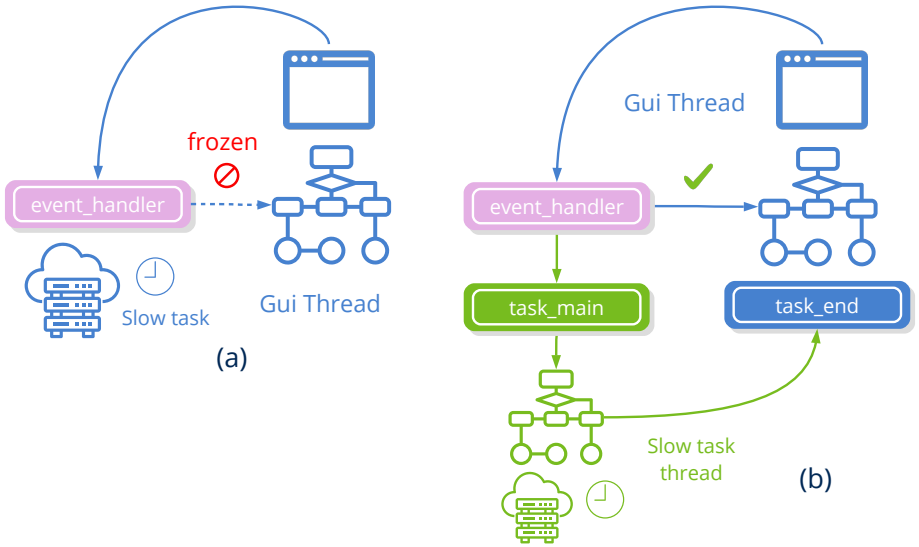


Figura 19.6: (a) Bloqueo de la interfaz por una función lenta. (b) Función lenta en un hilo paralelo.

```
}

osapp_task(tdata, .04, i_task_main, i_task_update, i_task_end, TaskData);
```

El nuevo hilo comenzará su ejecución en `task_main`. Esta función **no debería** acceder a los elementos de la interfaz, tan solo realizar cálculos o tareas de entrada/salida. Si fuera necesario actualizar el GUI mientras dure la tarea (incrementando una barra de progreso o similar) deberá realizarse en `task_update`, indicando en `updtime` el intervalo de actualización. El nuevo hilo terminará cuando se retorne de `task_main`, momento que se llamará a `task_end` en el hilo principal. Evidentemente, si ambos hilos acceden a variables compartidas, deberán protegerse mediante un `Mutex`.

Capítulo 20

Librería INet

20.1. INet

La librería **INet** implementa protocolos generales de Internet. Si bien los *"Sockets"* (Página 181) nos permiten abrir un canal de comunicación entre dos máquinas remotas, es necesario definir un formato para los mensajes que intercambiarán ambos interlocutores, con el fin de que la comunicación se lleve a cabo de manera satisfactoria. Cualquier sistema operativo moderno proporciona APIs para utilizar los servicios de Internet más populares, como HTTP. INet accede a esta funcionalidad bajo una interfaz común unificada y simplificada (Figura 20.1).

20.2. HTTP

Es habitual que una aplicación necesite información más allá de la almacenada en el propio computador. La forma más sencilla y habitual de compartir información es almacenarla en un Servidor Web y publicar una URL que provea el contenido deseado (Figura 20.2). Este esquema cliente/servidor utiliza el protocolo HTTP/HTTPS, que fue originalmente

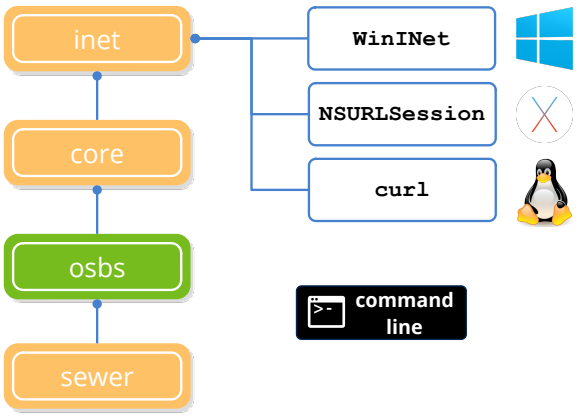


Figura 20.1: Dependencias de *INet*. Ver *"NAppGUI API"* (Página 147).

diseñado para transmitir documentos HTML entre servidores Web y navegadores. Debido a la gran repercusión que ha tenido a lo largo de los años, se ha ido ampliando su uso para el intercambio de información estructurada entre cualquier aplicación que "entienda" HTTP. La respuesta del servidor será, por lo general, un bloque de texto formateado en JSON o XML.

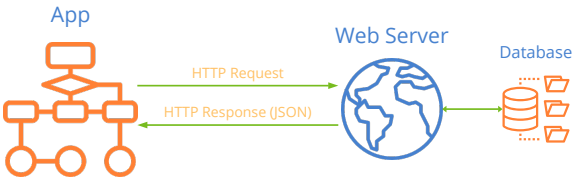


Figura 20.2: Petición de un recurso remoto mediante HTTP.

- Utiliza `http_dget` para descargar un recurso a partir de su *"URL"* (Página 391) (Listado 20.1).
- Utiliza `http_create` para crear una sesión HTTP.
- Utiliza `http_secure` para crear una sesión HTTPS (encriptado).

Listado 20.1: Descarga directa del contenido de una URL.

```
Stream *webpage = http_dget("https://nappgui.com/en/start/win_mac_linux.
    ↪ html", NULL, NULL);
Stream *imgdata = http_dget("http://test.nappgui.com/image_formats/
    ↪ sea_02_rgb.png", NULL, NULL);
Image *image = image_read(imgdata);

if (webpage != NULL)
{
    ...
    stm_close(&webpage);
}
```

Por otro lado, si vamos a realizar sucesivas llamadas a un mismo servidor o si necesitamos mayor control sobre las cabeceras HTTP, deberemos crear una sesión (Listado 20.2).

Listado 20.2: Sesión HTTP.

```
Stream *webpage = NULL;

Http *http = http_secure("nappgui.com", UINT16_MAX);
if (http_get(http, "/en/start/win_mac_linux.html", NULL, 0, NULL) == TRUE)
{
    if (http_response_status(http) == 200)
    {
        webpage = stm_memory(1024);
        if (http_response_body(http, webpage, NULL) == FALSE)
            stm_close(&webpage);
    }
}

http_destroy(&http);

if (webpage != NULL)
{
    ...
    stm_close(&webpage);
}
```

20.3. JSON

JSON *JavaScript Object Notation*, es un formato de datos en modo texto que permite representar de manera sencilla tipos básicos, objetos y arrays. Aunque su uso se ha popularizado en el ámbito Web puede utilizarse también para otros fines, como por ejemplo, archivos de configuración o intercambio local. Su sintaxis es fácil de entender para los humanos y sencilla de procesar para las máquinas. En (Listado 20.3) reproducimos un pequeño fragmento de la respuesta JSON de un servicio Web:

Listado 20.3: Fragmento JSON devuelto por un Web Service.

```
{
    "code":0,
    "size":80,
    "data":[
    {
        "id":0,
        "code":"i7-8700K",
        "description":"Intel BX80684I78700K 8th Gen Core i7-8700K Processor",
        type":0,
        "price":374.89,
        "image":"cpu_00.jpg",
        "image64":"\/9j\/4AAQSkZJRgABAQ...."
```

```
    },
    {
        "id":1,
        "code":"G3930",
        "description":"Intel BX80677G3930 7th Gen Celeron Desktop Processors",
        "type":0,
        "price":51.95,
        "image":"cpu_01.jpg",
        "image64":"\/9j\/4AAQSkZJRgABAQAAAQABAAD..."
    },
    ...
    ]
}
```

En su estructura podemos encontrar estos tipos de datos:

- **Booleanos:** Representados por las constantes `true` o `false`.
- **Números:** Utiliza la notación exponencial de C para valores en coma flotante: `23`, `.76`, `-0.54` o `5.6e12` son ejemplos válidos de valores numéricos. JSON no distingue entre enteros, negativos o reales.
- **Cadenas:** Cualquier texto entre comillas se considera una cadena. Admite cualquier carácter Unicode en *"UTF-8UTF-8"* (Página 160) o mediante la secuencia de escape `\uXXXX` para indicar el codepoint.
- **Arrays:** Listas de valores delimitados por cochetes `[ ... ]` y separados por comas (Listado 20.4).

Listado 20.4: Array JSON

```
[
    "Red", "Green", "Blue", "Yellow"
]
```

- **Objetos:** Están delimitados por llaves y compuestos por varios campos separados por comas. Cada campo lo forman un identificador (cadena) seguido del carácter dos puntos y un valor que puede ser cualquier tipo simple, objeto u array (Listado 20.5).

Listado 20.5: Objeto JSON

```
{
    "field1" : true,
    "field2" : 24.67,
    "field3" : "Hello Pibe",
    "field4" : [1, 2, 4, 8.4],
    "field5" : { "x" : 34.32, "y" : -6.19 }
}
```

- **null:** Indica la ausencia de valor.

- **Binarios:** JSON no soporta datos binarios por lo que objetos opacos (imágenes, por ejemplo) deben ser codificados en texto y transmitidos como un valor de tipo cadena. El formato más extendido y soportado globalmente es el *"Base64"* (Página 392) donde cada carácter representa 6 bits de información.

El parser JSON de NAppGUI transforma automáticamente los objetos `Image` a Base64 y viceversa, lo que permite incrustar imágenes como campos de datos.

20.3.1. Análisis de JSON y conversión a datos en C

NAppGUI permite el análisis automático de información Json.

- Utiliza `json_read` para traducir un Json a C.
- Utiliza `json_destroy` para destruir un objeto previamente leído.

Mostraremos a continuación diferentes ejemplos con tipos básicos, arrays y objetos. En *"Lectura/Escritura de Json"* (Página 653) tienes el código completo. El primer paso es crear un `Stream` con el contenido del Json (Listado 20.6):

Listado 20.6: Crear un Stream con datos Json.

```
/* Json data from web service */
Stream *stm = http_dget("http://serv.nappgui.com/dproducts.php", NULL, NULL);

/* Json data from disk file */
Stream *stm = hfile_stream("/home/fran/appdata/products.json", NULL);

/* Json data from memory block */
const char_t *data = "[12, 34, 67, 45]";
Stream *stm = stm_from_block((const byte_t*)data, str_len_c(data));
```

El Stream deberá destruirse con `stm_close` al finalizar el análisis.

Después utilizaremos `json_read` indicando el tipo de dato esperado del Json.

Listado 20.7: Json boolean.

```
json: true

bool_t *json = json_read(stm, NULL, bool_t);
bstd_printf("Json boolean: %d\n", *json);
json_destroy(&json, bool_t);
```

Listado 20.8: Json number.

```
json: 6654
```

```
uint16_t *json = json_read(stm, NULL, uint16_t);
bstd_printf("Json unsigned int: %d\n", *json);
json_destroy(&json, uint16_t);
```

Listado 20.9: Json string.

```
json: "Hello World"

String *json = json_read(stm, NULL, String);
bstd_printf("Json string: %s\n", tc(json));
json_destroy(&json, String);
```

Listado 20.10: Json string/b64 image (jpg, png, bmp).

```
json: "/9j/4QB4RXhpZgAASUkqAAgAAA..."

Image *json = json_read(stm, NULL, Image);
uint32_t width = image_width(json);
uint32_t height = image_height(json);
bstd_printf("Json image: width: %d height: %d\n", width, height);
json_destroy(&json, Image);
```

Listado 20.11: Json integer array

```
json: [ -321, 12, -8943, 228, -220, 347 ]

ArrSt(int16_t) *json = json_read(stm, NULL, ArrSt(int16_t));
bstd_printf("Json array: ");
arrst_foreach(id, json, int16_t)
    bstd_printf("%d ", *id);
arrst_end()
bstd_printf("\n");
json_destroy(&json, ArrSt(int16_t));
```

Listado 20.12: Json string array

```
json: [ "Red", "Green", "Blue", "Yellow", "Orange" ]

ArrPt(String) *json = json_read(stm, NULL, ArrPt(String));
bstd_printf("Json array: ");
arrpt_foreach(str, json, String)
    bstd_printf("%s ", tc(str));
arrpt_end()
bstd_printf("\n");
json_destroy(&json, ArrPt(String));
```

Para el análisis de objetos es necesario que registremos con *"Data binding"* (Página 230) la estructura de los mismos, de tal forma que coincidan los tipos y nombres de los campos del objeto Json con los `struct` de C. Dado este Json:

Listado 20.13: Json object

```
{
    "size" : 3,
    "data" : [
        {
            "description" : "Intel i7-7700K",
            "price" : 329.99
        },
        {
            "description" : "Ryzen-5-1600",
            "price" : 194.99
        },
        {
            "description" : "GTX-1060",
            "price" : 449.99
        }
    ]
}
```

Definimos estos `struct` y los registramos:

Listado 20.14: Estructuras que albergarán los datos del objeto Json.

```
typedef struct _product_t Product;
typedef struct _products_t Products;

struct _product_t
{
    String *description;
    real32_t price;
};

struct _products_t
{
    uint32_t size;
    ArrSt(Product) *data;
};

DeclSt(Product);

dbind(Product, String*, description);
dbind(Product, real32_t, price);
dbind(Products, uint32_t, size);
dbind(Products, ArrSt(Product)*, data);
```

De esta forma ya podemos llamar a `json_read`:

Listado 20.15: Lectura del objeto Json.

```
Products *json = json_read(stm, NULL, Products);
bstd_printf("Json object: Size %d\n", json->size);
```

```
arrst_foreach(elem, json->data, Product)
    bstd_printf("Product: %s Price %.2f\n", tc(elem->description), elem->price)
        ↪ ;
arrst_end()
bstd_printf("\n");
json_destroy(&json, Products);
```

***json_read()** ignora (salta) aquellos campos de objetos Json que no estén registrados con **dbind**. En ningún caso generarán cachés ni memoria dinámica.*

20.3.2. Mapeo entre Json y C

`json_read` reconoce los tipos básicos de NAppGUI, así como `String`, `Image`, `ArrSt` y `ArrPt`. **No funcionará con otro tipo de datos** como `int` o `float`. Tampoco reconocerá las estructuras STL `vector`, `map`, etc. En (Tabla 20.1) mostramos la equivalencia entre los campos de un Json y los tipos de C que necesitamos para mapearlo correctamente.

Json	C	
boolean	`bool_t`	true, false
number	`int8_t`, `int16_t`, `int32_t`, `int64_t`	-6785, 45, 0
number	`uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`	1, 36734, 255, 0, 14
number	`real32_t`, `real64_t`	67.554, -3.456, 1.5e7
string	`String`	“Intel Celeron”, “Red”
string	`Image`	“/9j/4QB4RXhpZgAASUkqAA
array	`ArrSt`(uint16_t)	[12, 111, 865]
array	`ArrSt`(real32_t)	[-34.89, 0.0001, 567.45, 1
array	`ArrPt`(String)	[“red”, “green”, “blue”
array	`ArrPt`(Image)	[“/9j/4QB4RXh...”, “/9j/4QB4
object	`struct Product` (*“Data binding”* (Página 230))	{ “description” : “i7-8700K”, “ ”pri
array	`ArrSt`(Product)	[{ “description” : “i7-8700K”, “ ”price
array	`ArrPt`(Product)	[{ “description” : “i7-8700K”, “ ”price

Tabla 20.1: Equivalencia entre tipos Json y NAppGUI.

20.3.3. Convertir desde C a JSON

- Utiliza `json_write` para escribir datos/objetos desde C a Json.

Basándonos de nuevo en (Tabla 20.1), vamos a realizar el proceso inverso y generar datos Json a partir de tipos y objetos en C. Lo primero es crear un stream de escritura para albergar el resultado (Listado 20.16):

Listado 20.16: Crear un Stream de escritura.

```
/* Write stream in memory */
Stream *stm = stm_memory(2048);

/* Write stream in disk */
Stream *stm = stm_to_file("/home/fran/appdata/products.json", NULL);
```

El Stream deberá destruirse con `stm_close` cuando ya no sea necesario.

Después utilizaremos `json_write` indicando el tipo de dato esperado del Json.

Listado 20.17: Escribir booleano a Json.

```
bool_t data_bool = TRUE;
stm_writef(stm, "Json from bool_t: ");
json_write(stm, &data_bool, NULL, bool_t);

// Json from bool_t: true
```

Listado 20.18: Escribir entero a Json.

```
uint16_t data_uint = 6654;
stm_writef(stm, "Json from uint16_t: ");
json_write(stm, &data_uint, NULL, uint16_t);

// Json from uint16_t: 6654
```

Listado 20.19: Escribir String a Json.

```
String *data_str = str_c("Hello World");
stm_writef(stm, "Json from String: ");
json_write(stm, data_str, NULL, String);
str_destroy(&data_str);

// Json from String: "Hello World"
```

Listado 20.20: Escribir Image a Json.

```
Image *data_image = load_image();
stm_writef(stm, "Json from Image: ");
json_write(stm, data_image, NULL, Image);
```

```
image_destroy(&data_image);

// Json from Image: "iVBORw0KGgoAAAANSUhEUgAAAAIA..."
```

Listado 20.21: Escribir ArrSt(int16_t) a Json.

```
ArrSt(int16_t) *array = arrst_create(int16_t);
arrst_append(array, -321, int16_t);
arrst_append(array, 12, int16_t);
arrst_append(array, -8943, int16_t);
arrst_append(array, 228, int16_t);
arrst_append(array, -220, int16_t);
arrst_append(array, 347, int16_t);
stm_writef(stm, "Json from int array: ");
json_write(stm, array, NULL, ArrSt(int16_t));
arrst_destroy(&array, NULL, int16_t);

// Json from int array: [ -321, 12, -8943, 228, -220, 347 ]
```

Listado 20.22: Escribir ArrPt(String) a Json.

```
ArrPt(String) *array = arrpt_create(String);
arrpt_append(array, str_c("Red"), String);
arrpt_append(array, str_c("Green"), String);
arrpt_append(array, str_c("Blue"), String);
arrpt_append(array, str_c("Yellow"), String);
arrpt_append(array, str_c("Orange"), String);
stm_writef(stm, "Json from string array: ");
json_write(stm, array, NULL, ArrPt(String));
arrpt_destroy(&array, str_destroy, String);

// Json from string array: [ "Red", "Green", "Blue", "Yellow", "Orange" ]
```

Listado 20.23: Escribir objeto Products a Json.

```
Products *products = heap_new(Products);
products->size = 3;
products->data = arrst_create(Product);

{
    Product *product = arrst_new(products->data, Product);
    product->description = str_c("Intel i7-7700K");
    product->price = 329.99f;
}

{
    Product *product = arrst_new(products->data, Product);
    product->description = str_c("Ryzen-5-1600");
    product->price = 194.99f;
}
```

```
{
    Product *product = arrst_new(products->data, Product);
    product->description = str_c("GTX-1060");
    product->price = 449.99f;
}

stm_writef(stm, "Json from object: ");
json_write(stm, products, NULL, Products);
dbind_destroy(&products, Products);

// Json from object: {"size" : 3, "data" : [ {"description" : "Intel i7-7700K",
    ↪  "price" : 329.989990 }, {"description" : "Ryzen-5-1600", "price" :
    ↪ 194.990005 }, {"description" : "GTX-1060", "price" : 449.989990 } ] }
```

20.4. URL

URL es el acrónimo de *Uniform Resource Locator* que identifica un recurso único en Internet. El uso más común lo encontramos al realizar peticiones a un servidor Web. Por ejemplo `https://www.google.com` es una URL ampliamente reconocida y utilizada. Siendo algo más concretos, podemos decir que es una cadena de caracteres con formato específico compuesta por una serie de campos que permiten localizar sin ambigüedad un recurso único global (Listado 20.24) (Figura 20.3).

Listado 20.24: Análisis de una cadena URL.

```
Url *url = url_parse("https://frang@www.nappgui.com/services/demo/userlist.php?
    ↪ id=peter&city=Alicante");
const char_t *scheme = url_scheme(url); // https
const char_t *host = url_host(url);     // www.nappgui.com
const char_t *path = url_path(url);     // /services/demo/userlist.php
const char_t *query = url_query(url);   // id=peter&city=Alicante
```

- **Scheme:** Protocolo de comunicación utilizado. **http**, **https**, **ftp**, **smtp**, **mailto**, etc.
- **Authority:** Cadena de acceso al servidor compuesta por varios campos, donde tan solo el nombre del host es requerido. El resto son opcionales.
  - **Host:** Nombre del servidor o dirección IP.
  - **User:** Nombre del usuario. Opcional, solo si el servicio lo requiere.
  - **Password:** Contraseña. Opcional, solo si el servicio lo requiere.
  - **Port:** Puerto de acceso. Cada protocolo tiene un puerto por defecto, que será el utilizado si no se especifica ninguno. 80 = http, 413 = https.
- **Resource:** Ruta dentro del servidor donde se encuentra el recurso que buscamos. El *pathname* es el único requerido.

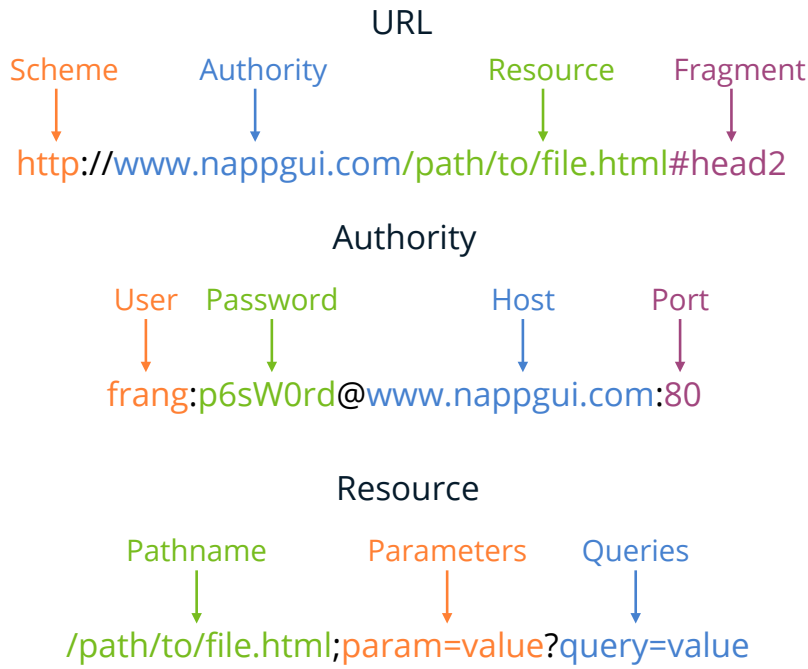


Figura 20.3: Los diferentes campos que componen una URL.

- **Pathname:** Directorio y nombre del archivo o recurso.
- **Parameters:** Lista de argumentos `name=value` que puede necesitar el servicio. Normalmente no utilizados. Si existen varios valores, se separan con el carácter `'&'`.
- **Queries:** Lista de argumentos `name=value` que puede necesitar el servicio. Estos son los normalmente utilizados por los servicios Web. Es decir, en la URL hay que utilizar el separador `'?'` en lugar de `';'` después del *pathname*. Si existen varios valores, se separan con el carácter `'&'`.

- **Fragment:** Es un ancla a una parte concreta del documento que pedimos al servidor. Normalmente utilizado para acceder a un punto concreto de una página HTML.

20.5. Base64

Parte 3

Aplicaciones de ejemplo

Capítulo 21

Die

El buen código debería ser simple, claro y fácil de entender. El buen código debería ser compacto: solo lo necesario para hacer el trabajo y nada más, pero no críptico, hasta el punto de que no se pueda entender. El buen código debería ser genérico, para que pueda resolver una amplia clase de problemas de manera uniforme. Podría describirse como elegante, muestra de buen gusto y refinamiento.

Brian Kernighan

Como el camino se demuestra andando, vamos a dedicar unos capítulos a profundizar en el uso de NAppGUI de la mano de aplicaciones reales. Nuestro objetivo es presentar programas de cierto nivel, a medio camino entre los sencillos "ejemplos de libro" y las aplicaciones comerciales. En esta primera demo tenemos un programa que nos permite dibujar la silueta de un dado (Figura 21.1) y que nos servirá como excusa para introducir conceptos de dibujo paramétrico, composición de *layouts* y uso de recursos. El **código fuente** está en la caperta `/src/demo/die` de la distribución del SDK. En *"Crear nueva aplicación"* (Página 101) y *"Recursos"* (Página 131) vimos como crear el proyecto desde cero.

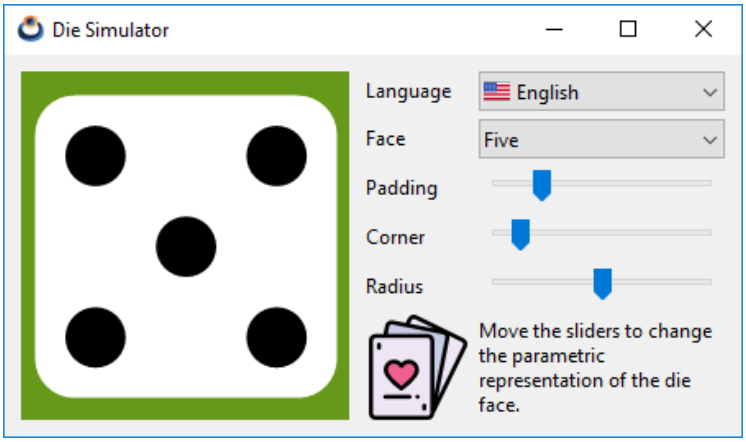


Figura 21.1: Aplicación *Die Simulator*, versión Windows. Inspirado en DieView *(Cocoa Programming for OSX, Hillegass et al.)*

Figura 21.2: Versión macOS.

21.1. Uso de sublayouts

Comenzamos trabajando en la interfaz de usuario, que hemos dividido en dos áreas: una vista personalizada (`View`) donde dibujaremos la representación del dado en 2D, y una zona de controles donde podremos interactuar con dicho dibujo. Como ya vimos en *"¡Hola Mundo!"* (Página 23) utilizaremos objetos `Layout` para ubicar los controles dentro de la ventana principal. No obstante observamos que esta disposición de elementos no encaja bien en una única tabla, por lo tanto, usaremos dos celdas en horizontal como contenedor principal y un *grid* de dos columnas y seis filas para los controles (Listado 21.1) (Listado 21.1). Este segundo layout se ubicará en la celda derecha del primer contenedor y diremos que es un **sublayout** del layout principal.

Figura 21.3: Versión Linux/GTK+.

Listado 21.1: Composición mediante sublayouts.

```
Layout *layout = layout_create(2, 1);
Layout *layout1 = layout_create(2, 6);
layout_view(layout, view, 0, 0);
layout_label(layout1, label1, 0, 0);
layout_label(layout1, label2, 0, 1);
layout_label(layout1, label3, 0, 2);
layout_label(layout1, label4, 0, 3);
layout_label(layout1, label5, 0, 4);
layout_view(layout1, vimg, 0, 5);
layout_popup(layout1, popup1, 1, 0);
layout_popup(layout1, popup2, 1, 1);
layout_slider(layout1, slider1, 1, 2);
layout_slider(layout1, slider2, 1, 3);
layout_slider(layout1, slider3, 1, 4);
layout_label(layout1, label6, 1, 5);
layout_layout(layout, layout1, 1, 0);
```

De igual forma que hicimos en *"Formato del LayoutFormato del Layout"* (Página 29) hemos establecido ciertos márgenes y un ancho fijo para la columna de los controles.

Listado 21.2: Formato del layout

```
view_size(view, s2df(200.f, 200.f));
layout_margin(layout, 10.f);
layout_hsize(layout1, 1, 150.f);
layout_hmargin(layout, 0, 10.f);
layout_hmargin(layout1, 0, 5.f);
layout_vmargin(layout1, 0, 5.f);
layout_vmargin(layout1, 1, 5.f);
layout_vmargin(layout1, 2, 5.f);
```

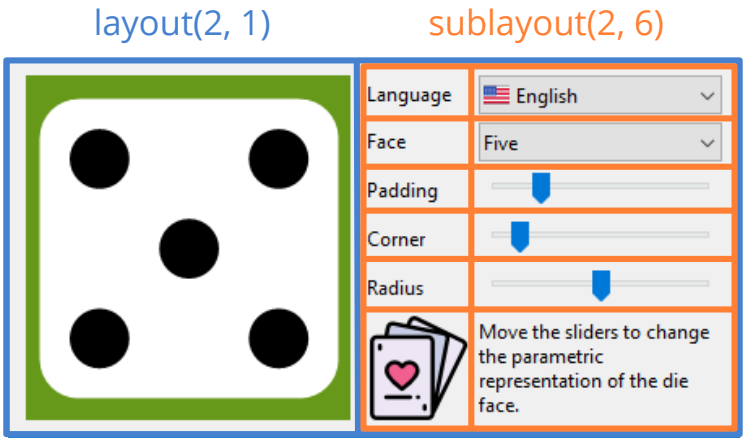


Figura 21.4: El uso de sublayouts añade flexibilidad a la hora de diseñar el *gui*.

```
layout_vmargin(layout1, 3, 5.f);
layout_vmargin(layout1, 4, 5.f);
```

21.2. Uso de vistas personalizadas

`View` son controles que nos permitirán diseñar nuestros propios *widgets*. Al contrario que ocurre con otro tipo de componentes, como *"Slider"* (Página 317) o *"Button"* (Página 310), aquí tendremos total libertad para dibujar cualquier cosa. Podremos interactuar con el control capturando sus eventos (ratón, teclado, etc) e implementando los manejadores adecuados. Estas vistas se integran en el layout como cualquier otro componente (Listado 21.3).

Listado 21.3: Creación de una vista personalizada.

```
View *view = view_create();
view_size(view, s2df(200.f, 200.f));
layout_view(layout, view, 0, 0);
```

No podemos dibujar dentro de un `View` cuando queramos. Tendremos que realizar una solicitud al sistema operativo mediante el método `view_update` (Listado 21.4), ya que el área de dibujo puede afectar a ventanas superpuestas y esto debe gestionarse de forma centralizada. Cuando el control esté listo para refrescarse, el sistema enviará un evento `EvDraw` que debemos capturar a través de `view_OnDraw`.

Listado 21.4: Código básico de refresco del View.

```
static void i_OnPadding(App *app, Event *e)
{
```

```
    const EvSlider *params = event_params(e, EvSlider);
    app->padding = params->pos;
    view_update(app->view);
}

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    die_draw(params->context, params->width, params->height, app);
}

slider_OnMoved(slider1, listener(app, i_OnPadding, App));
view_OnDraw(view, listener(app, i_OnDraw, App));
```

Cada vez que el usuario mueve un slider (parámetro padding, por ejemplo) el sistema operativo captura la acción e informa a la aplicación a través del método `i_OnPadding` (Figura 21.5). Como la acción implica un cambio en el dibujo, este método llama a `view_update` para informar de nuevo al sistema que la vista debe actualizarse. Cuando este lo considera apropiado, manda el evento `EvDraw`, que es capturado por `i_OnDraw` donde se regenera el dibujo con los nuevos parámetros.

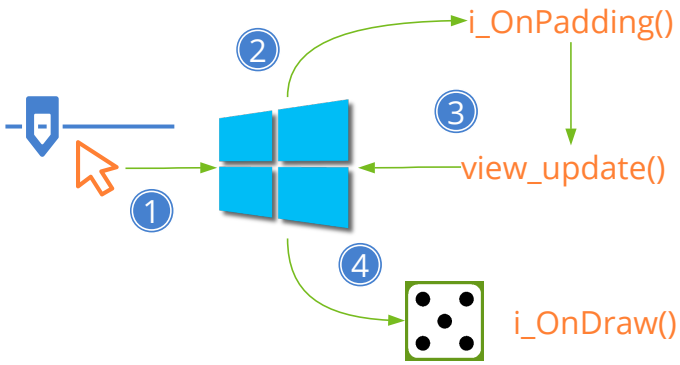


Figura 21.5: Comprendiendo el flujo de eventos en dibujos interactivos.

21.3. Dibujo paramétrico

Bajo este concepto se describe la capacidad para generar imágenes vectoriales a partir de unos pocos valores numéricos conocidos como parámetros (Figura 21.6). Se usa bastante en el diseño asistido por computadora (CAD), ya permite realizar ajustes de forma sencilla en planos o modelos sin tener que editar, una por una, un sinfín de primitivas.

En nuestra aplicación, la representación del dado puede cambiar en tiempo de ejecución a medida que el usuario manipula los sliders o dimensiona la ventana, por lo que calculamos la posición y el tamaño de sus primitivas utilizando fórmulas paramétricas. Una vez resueltas, creamos el dibujo con tres simples comandos del API *"Primitivas de dibujo"* (Página 271).

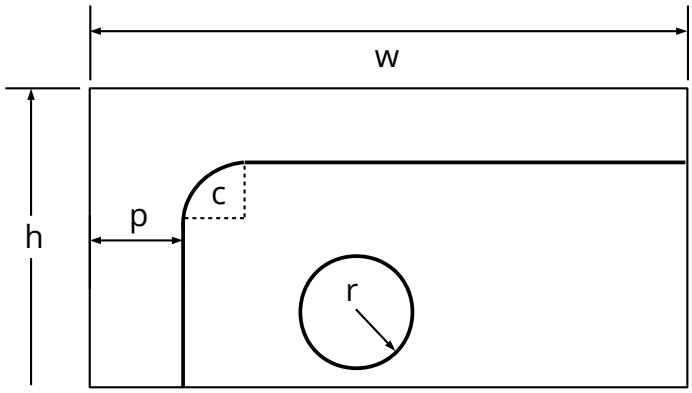


Figura 21.6: Principios del dibujo paramétrico, aplicados en Die.

- `draw_clear`. Borra toda el área de dibujo utilizando un color sólido.
- `draw_rndrect`. Dibuja un rectángulo con bordes redondeados.
- `draw_circle`. Dibuja un círculo.

Listado 21.5: demo/casino/ddraw.c

```
/* Die drawing */

#include "ddraw.h"
#include <draw2d/draw2dall.h>

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static const real32_t i_MAX_PADDING = 0.2f;
const real32_t kDEF_PADDING = .15f;
const real32_t kDEF_CORNER = .15f;
const real32_t kDEF_RADIUS = .35f;

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

void die_draw(DCtx *ctx, const real32_t x, const real32_t y, const
    ↪ real32_t width, const real32_t height, const real32_t padding,
    ↪ const real32_t corner, const real32_t radius, const uint32_t face)
{
    color_t white = color_rgb(255, 255, 255);
    color_t black = color_rgb(0, 0, 0);
    real32_t dsize, dx, dy;
    real32_t rc, rr;
    real32_t p1, p2, p3;

    dsize = width < height ? width : height;
```

```
    dsize -= bmath_floorf(2.f * dsize * padding * i_MAX_PADDING);
    dx = x + .5f * (width - dsize);
    dy = y + .5f * (height - dsize);
    rc = dsize * (.1f + .3f * corner);
    rr = dsize * (.05f + .1f * radius);
    p1 = 0.5f * dsize;
    p2 = 0.2f * dsize;
    p3 = 0.8f * dsize;

    draw_fill_color(ctx, white);
    draw_rndrect(ctx, ekFILL, dx, dy, dsize, dsize, rc);
    draw_fill_color(ctx, black);

    if (face == 1 || face == 3 || face == 5)
        draw_circle(ctx, ekFILL, dx + p1, dy + p1, rr);

    if (face != 1)
    {
        draw_circle(ctx, ekFILL, dx + p3, dy + p2, rr);
        draw_circle(ctx, ekFILL, dx + p2, dy + p3, rr);
    }

    if (face == 4 || face == 5 || face == 6)
    {
        draw_circle(ctx, ekFILL, dx + p2, dy + p2, rr);
        draw_circle(ctx, ekFILL, dx + p3, dy + p3, rr);
    }

    if (face == 6)
    {
        draw_circle(ctx, ekFILL, dx + p2, dy + p1, rr);
        draw_circle(ctx, ekFILL, dx + p3, dy + p1, rr);
    }
}
```

Los comandos de dibujo se plasman sobre un lienzo (canvas), también conocido como contexto `DCtx`. Este objeto llega a `i_OnDraw` como parámetro del evento `EvDraw`. En este caso, el lienzo lo proporciona el propio control `View`, pero también es posible crear contextos para dibujar directamente en memoria.

21.4. Redimensionado

En esta aplicación la ventana puede cambiar de tamaño estirando el cursor sobre sus bordes, algo habitual en programas de escritorio. Vamos a ver algunos aspectos básicos sobre esta característica no presente en *"¡Hola Mundo!"* (Página 23), que presentaba una ventana estática. Lo primero es habilitar la opción dentro del constructor de la ventana.

```
window_create(ekWINDOW_STDRES, &panel);
```

Cuando una ventana cambia de tamaño, los controles interiores deben hacerlo proporcionalmente así como cambiar su ubicación dentro del panel. Esta gestión se lleva a cabo dentro de cada objeto `Layout`. Cuando se inicia la ventana, el tamaño por defecto de cada layout se calcula aplicando el **dimensionamiento natural**, que es el resultante del tamaño inicial de los controles más los márgenes, como ya vimos en *"Formato del Layout-Formato del Layout"* (Página 29). Cuando estiramos o contraemos la ventana, la diferencia de píxeles entre el dimensionamiento natural y el real se distribuye entre las columnas del layout (Figura 21.7). Lo mismo ocurre con la diferencia vertical, que se distribuye entre sus filas. Si una celda contiene un sublayout, este incremento se distribuirá recursivamente por sus propias columnas y filas.

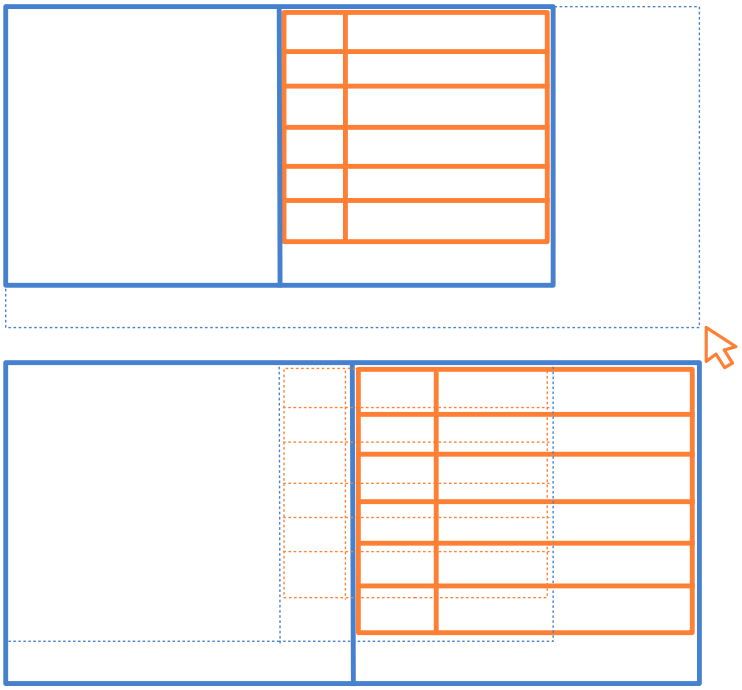


Figura 21.7: Al redimensionar, el exceso de píxeles se distribuye proporcionalmente por las filas y columnas del `Layout`.

Pero en este caso particular, queremos que todo el incremento vaya al área de dibujo (columna 0). Dicho de otro modo, buscamos que la columna de los controles permanezca con ancho fijo y no crezca (Figura 21.8). Para ello debemos cambiar la proporción del redimensionado:

```
layout_hexpand(layout, 0);
```

Con esta función el 100 % del excedente horizontal irá a la columna 0. Por defecto, teníamos una proporción del (50 %, 50 %) ya que son dos columnas (33 % para tres, 25 %

para cuatro, etc). Con esto ya tendríamos resuelto el redimensionado para la dimensión X de la ventana, pero ¿qué ocurre con la vertical?. En el layout principal, solo tenemos una fila que, al expandirse, cambiará la altura de la vista personalizada. Pero esta expansión también afectará a la celda de la derecha, donde los controles también crecerán verticalmente debido al aumento recursivo de píxeles en las filas del sublayout. Para resolverlo, forzamos la alineación vertical `ekTOP` en la celda derecha del layout.

```
layout_valign(layout, 1, 0, ekTOP);
```

en lugar del `ekJUSTIFY`, que es la alineación predeterminada para sublayouts. De esta manera, el contenido de la celda (el sublayout completo) no se expandirá verticalmente, si no que se ajustará al borde superior dejando todo el espacio libre en la parte inferior de la celda. Obviamente, si usamos `ekCENTER` o `ekBOTTOM`, el sublayout se centrará o ajustará al borde inferior.

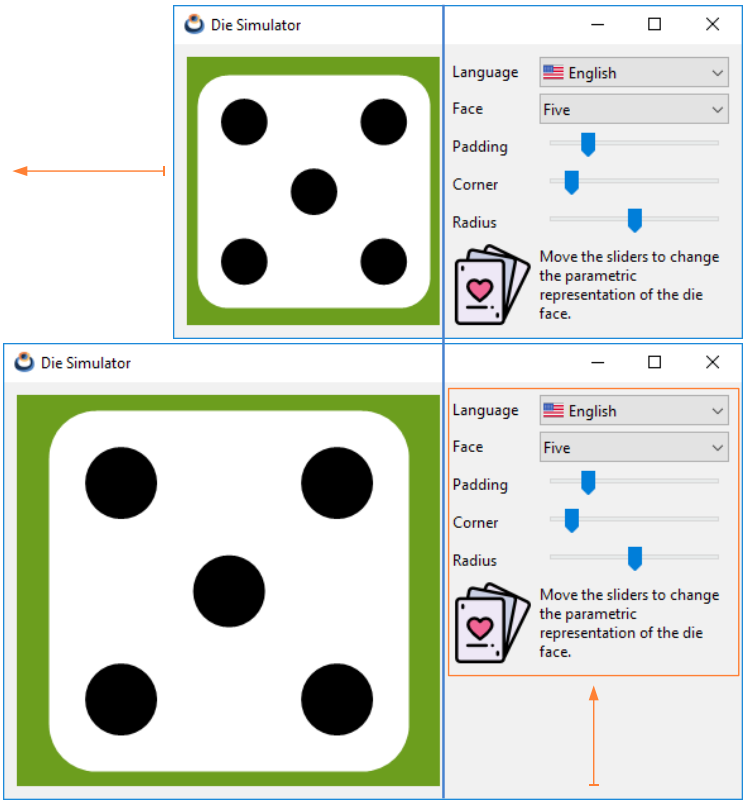


Figura 21.8: Jugando con la proporción horizontal y la alineación vertical, solo el área de dibujo se verá afectada por los cambios de tamaño.

21.5. Uso de recursos

Tanto los texto como los iconos que hemos utilizado en *Die* se han externalizado en el paquete de recursos `all`. Gracias a ello, podemos realizar una traducción automática de la interfaz entre los idiomas Inglés y Español. Puedes consultar *"Recursos"* (Página 131) para obtener información detallada de como se han asignado los textos e imágenes en la interfaz del programa.

Listado 21.6: demo/die/res/res_die/strings.msg

```
/* Die strings */
TEXT_FACE       Face
TEXT_PADDING    Padding
TEXT_CORNER     Corner
TEXT_RADIUS     Radius
TEXT_ONE        One
TEXT_TWO        Two
TEXT_THREE      Three
TEXT_FOUR       Four
TEXT_FIVE       Five
TEXT_SIX        Six
TEXT_TITLE      Die Simulator
TEXT_INFO       Move the sliders to change the parametric representation of the
    ↪  die face.
TEXT_LANG       Language
TEXT_ENGLISH    English
TEXT_SPANISH    Spanish
```

Listado 21.7: demo/die/res/res_die/es_es/strings.msg

```
/* Die strings */
TEXT_FACE       Cara
TEXT_PADDING    Margen
TEXT_CORNER     Borde
TEXT_RADIUS     Radio
TEXT_ONE        Uno
TEXT_TWO        Dos
TEXT_THREE      Tres
TEXT_FOUR       Cuatro
TEXT_FIVE       Cinco
TEXT_SIX        Seis
TEXT_TITLE      Simulador de dado
TEXT_INFO       Mueve los sliders para cambiar la representación paramétrica de
    ↪  la cara del dado.
TEXT_LANG       Idioma
TEXT_ENGLISH    Inglés
TEXT_SPANISH    Español
```

21.6. Die y Dice

Esta aplicación se ha utilizado como hilo conductor del capítulo *"Crear nueva aplicación"* (Página 101) y siguientes del tutorial de NAppGUI. El ejemplo completo consta de dos aplicaciones (**Die** y **Dice**), así como de la librería **casino** que agrupa las rutinas comunes para ambos programas (Figura 21.9). Los tres proyectos completos los tienes listos para compilar y probar en la carpeta `src/demo` de la distribución del SDK.

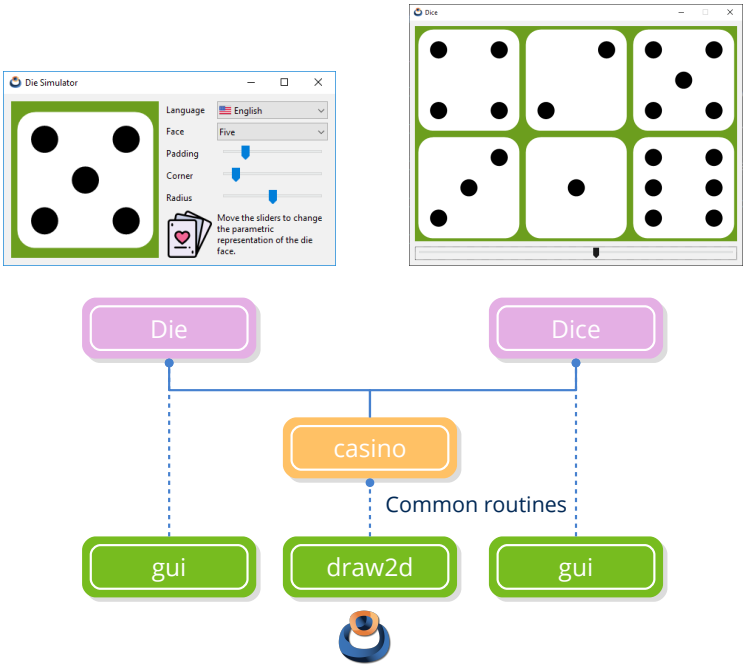


Figura 21.9: Las rutinas comunes para ambas aplicaciones se comparten mediante la librería **casino**.

21.7. El programa Die completo

Listado 21.8: demo/die/die.hxx

```
/* Die Types */

#ifndef __DIE_HXX__
#define __DIE_HXX__

#include <gui/gui.hxx>

typedef struct _app_t App;

struct _app_t
{
```

```
    real32_t padding;
    real32_t corner;
    real32_t radius;
    uint32_t face;
    View *view;
    Window *window;
};

#endif
```

Listado 21.9: demo/die/main.c

```
/* Die application */

#include "dgui.h"
#include <nappgui.h>

/*---------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    app->padding = 0.2f;
    app->corner = 0.1f;
    app->radius = 0.5f;
    app->face = 5;
    app->window = dgui_window(app);
    window_origin(app->window, v2df(200.f, 200.f));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/
```

```
#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Listado 21.10: demo/die/dgui.c

```
/* Die Gui */

#include "dgui.h"
#include "ddraw.h"
#include "res_die.h"
#include <gui/guiall.h>

/*---------------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    color_t green = color_rgb(102, 153, 26);
    const EvDraw *params = event_params(e, EvDraw);
    draw_clear(params->ctx, green);
    die_draw(params->ctx, 0, 0, params->width, params->height, app->padding,
        ↪ app->corner, app->radius, app->face);
}

/*---------------------------------------------------------------------------*/

static void i_OnFace(App *app, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    app->face = params->index + 1;
    view_update(app->view);
}

/*---------------------------------------------------------------------------*/

static void i_OnPadding(App *app, Event *e)
{
    const EvSlider *params = event_params(e, EvSlider);
    app->padding = params->pos;
    view_update(app->view);
}

/*---------------------------------------------------------------------------*/

static void i_OnCorner(App *app, Event *e)
{
    const EvSlider *params = event_params(e, EvSlider);
    app->corner = params->pos;
    view_update(app->view);
}
```

```
/*-----------------------------------------------------------------------------*/

static void i_OnRadius(App *app, Event *e)
{
    const EvSlider *params = event_params(e, EvSlider);
    app->radius = params->pos;
    view_update(app->view);
}

/*-----------------------------------------------------------------------------*/

static void i_OnLang(App *app, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    const char_t *lang = params->index == 0 ? "en_us" : "es_es";
    gui_language(lang);
    unref(app);
}

/*-----------------------------------------------------------------------------*/

 static Panel *i_panel(App *app)
 {
    Panel *panel = panel_create();
    Layout *layout = layout_create(2, 1);
    Layout *layout1 = layout_create(2, 6);
    View *view = view_create();
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_multiline();
    PopUp *popup1 = popup_create();
    PopUp *popup2 = popup_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    Slider *slider3 = slider_create();
    ImageView *img = imageview_create();
    app->view = view;
    view_size(view, s2df(200, 200));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    label_text(label1, TEXT_LANG);
    label_text(label2, TEXT_FACE);
    label_text(label3, TEXT_PADDING);
    label_text(label4, TEXT_CORNER);
    label_text(label5, TEXT_RADIUS);
    label_text(label6, TEXT_INFO);
    popup_add_elem(popup1, TEXT_ENGLISH, resid_image(USA_PNG));
    popup_add_elem(popup1, TEXT_SPANISH, resid_image(SPAIN_PNG));
    popup_OnSelect(popup1, listener(app, i_OnLang, App));
```

```
    popup_add_elem(popup2, TEXT_ONE, NULL);
    popup_add_elem(popup2, TEXT_TWO, NULL);
    popup_add_elem(popup2, TEXT_THREE, NULL);
    popup_add_elem(popup2, TEXT_FOUR, NULL);
    popup_add_elem(popup2, TEXT_FIVE, NULL);
    popup_add_elem(popup2, TEXT_SIX, NULL);
    popup_OnSelect(popup2, listener(app, i_OnFace, App));
    popup_selected(popup2, app->face - 1);
    slider_value(slider1, app->padding);
    slider_value(slider2, app->corner);
    slider_value(slider3, app->radius);
    slider_OnMoved(slider1, listener(app, i_OnPadding, App));
    slider_OnMoved(slider2, listener(app, i_OnCorner, App));
    slider_OnMoved(slider3, listener(app, i_OnRadius, App));
    imageview_image(img, (const Image*)CARDS_PNG);
    layout_view(layout, view, 0, 0);
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 0, 1);
    layout_label(layout1, label3, 0, 2);
    layout_label(layout1, label4, 0, 3);
    layout_label(layout1, label5, 0, 4);
    layout_imageview(layout1, img, 0, 5);
    layout_popup(layout1, popup1, 1, 0);
    layout_popup(layout1, popup2, 1, 1);
    layout_slider(layout1, slider1, 1, 2);
    layout_slider(layout1, slider2, 1, 3);
    layout_slider(layout1, slider3, 1, 4);
    layout_label(layout1, label6, 1, 5);
    layout_layout(layout, layout1, 1, 0);
    layout_margin(layout, 10);
    layout_hsize(layout1, 1, 150);
    layout_hmargin(layout, 0, 10);
    layout_hmargin(layout1, 0, 5);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 1, 5);
    layout_vmargin(layout1, 2, 5);
    layout_vmargin(layout1, 3, 5);
    layout_vmargin(layout1, 4, 5);
    layout_hexpand(layout, 0);
    layout_valign(layout, 1, 0, ekTOP);
    panel_layout(panel, layout);
    return panel;
}

/*-----------------------------------------------------------------------------*/

Window *dgui_window(App *app)
{
    gui_respack(res_die_respack);
    gui_language("");
```

```
    {
        Panel *panel = i_panel(app);
        Window *window = window_create(ekWINDOW_STDRES);
        window_panel(window, panel);
        window_title(window, TEXT_TITLE);
        return window;
    }
}
```

Listado 21.11: demo/die/dgui.h

```
/* Die Gui */

#include "die.hxx"

__EXTERN_C

Window *dgui_window(App *app);

__END_C
```

Capítulo 22

Bricks

Briks es una imitación muy simplista del videojuego **Atari Breakout**, que nos permitirá realizar una introducción al mundo de las *"Aplicaciones síncronasAplicaciones síncronas"* (Página 377). Cualquier aplicación en tiempo real debe estar constantemente actualizándose intervenga o no el usuario. El **código fuente** está en la caperta `/src/demo/bricks` de la distribución del SDK.

- Utiliza `osmain_sync` para iniciar una aplicación síncrona, indicando un intervalo y función *callback* de actualización. NAppGUI lanzará periódicamente eventos de tiempo que actualizarán el programa.

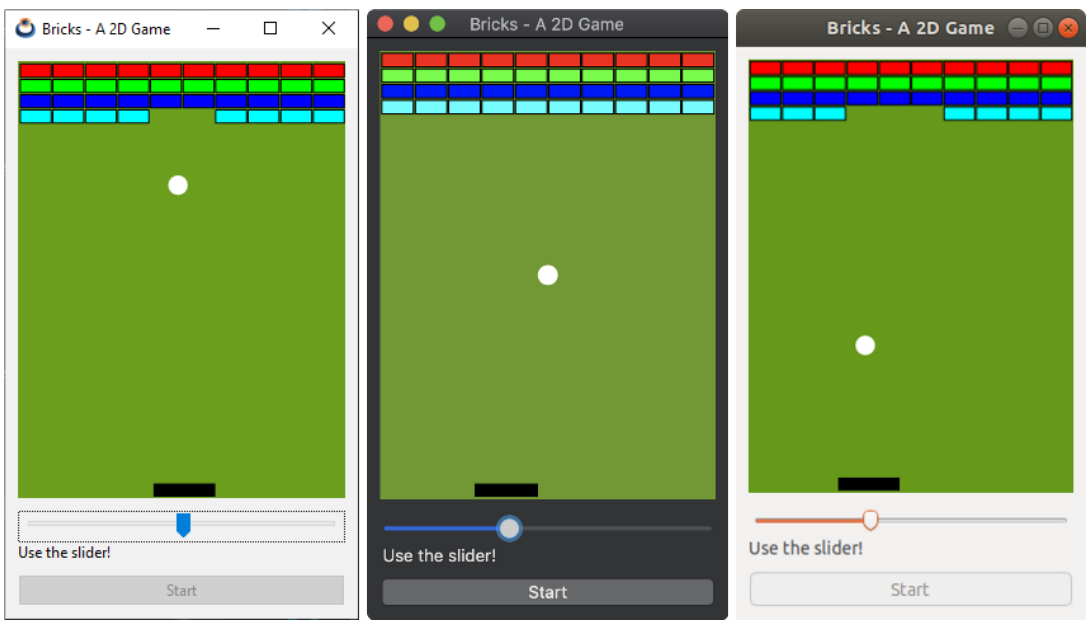


Figura 22.1: Videojuego bricks en Windows, macOS y Linux.

Esta aplicación está dirigida por dos eventos (Figura 22.2). Por un lado el movimiento del slider, que puede producirse en cualquier momento (evento asíncrono), y actualizará la posición del jugador. Por otro un evento síncrono producido por `osmain_sync` cada 40 milisegundos y será notificado a través de `i_update()` para que se actualice el estado del juego y la vista gráfica.

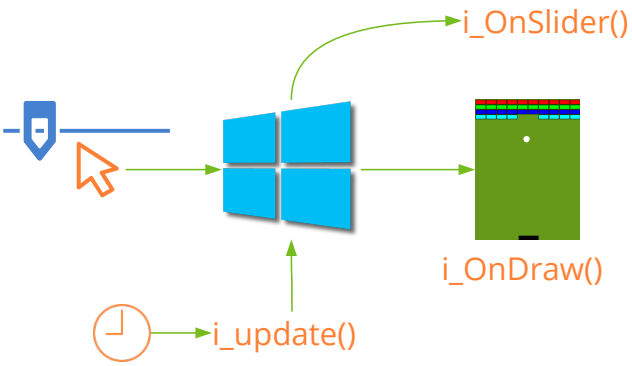


Figura 22.2: Eventos síncronos y asíncronos.

Listado 22.1: demo/bricks/bricks.c

```
/* Simplistic Breakout-like game */

#include <nappgui.h>

#define NUM_BRICKS 40

typedef struct _brick_t Brick;
typedef struct _app_t App;

struct _brick_t
{
    real32_t x;
    real32_t y;
    uint8_t color;
    bool_t is_visible;
};

struct _app_t
{
    bool_t is_running;
    Brick bricks[NUM_BRICKS];
    color_t color[4];
    real32_t brick_width;
    real32_t player_pos;
    real32_t ball_x;
    real32_t ball_y;
    V2Df ball_dir;
    real32_t ball_speed;
    Cell *button;
    Slider *slider;
```

```
    View *view;
    Window *window;
};

/*--------------------------------------------------------------------------*/

static const real32_t i_BALL_RADIUS = .03f;
static const real32_t i_BRICK_HEIGHT = .03f;
static const real32_t i_BRICK_SEPARATION = .005f;
static const uint32_t i_BRICKS_PER_ROW = 10;
static const uint32_t i_NUM_ROWS = 4;

/*--------------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    uint32_t i = 0;

    draw_clear(params->ctx, color_rgb(102, 153, 26));
    draw_line_color(params->ctx, kCOLOR_BLACK);

    for (i = 0; i < NUM_BRICKS; ++i)
    {
        if (app->bricks[i].is_visible == TRUE)
        {
            real32_t x = app->bricks[i].x * params->width;
            real32_t y = app->bricks[i].y * params->height;
            real32_t width = app->brick_width * params->width;
            real32_t height = i_BRICK_HEIGHT * params->height;
            draw_fill_color(params->ctx, app->color[app->bricks[i].color]);
            draw_rect(params->ctx, ekFILLSK, x, y, width, height);
        }
    }

    {
        real32_t x = (app->player_pos - app->brick_width) * params->width;
        real32_t y = (1 - i_BRICK_HEIGHT - i_BRICK_SEPARATION) * params->height
            ↪ ;
        real32_t width = 2 * app->brick_width * params->width;
        real32_t height = i_BRICK_HEIGHT * params->height;
        draw_fill_color(params->ctx, kCOLOR_BLACK);
        draw_rect(params->ctx, ekFILL, x, y, width, height);
    }

    {
        real32_t x = app->ball_x * params->width;
        real32_t y = app->ball_y * params->height;
        real32_t rad = i_BALL_RADIUS * params->width;
        draw_fill_color(params->ctx, kCOLOR_WHITE);
        draw_circle(params->ctx, ekFILL, x, y, rad);
```

```
    }
}

/*-------------------------------------------------------------------------*/

static void i_OnSlider(App *app, Event *e)
{
    const EvSlider *params = event_params(e, EvSlider);
    app->player_pos = params->pos;
}

/*-------------------------------------------------------------------------*/

static void i_OnStart(App *app, Event *e)
{
    unref(e);
    app->is_running = TRUE;
    cell_enabled(app->button, FALSE);
}

/*-------------------------------------------------------------------------*/

 static Panel *i_panel(App *app)
 {
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 4);
    View *view = view_create();
    Slider *slider = slider_create();
    Label *label = label_create();
    Button *button = button_push();
    view_size(view, s2df(258, 344));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    slider_OnMoved(slider, listener(app, i_OnSlider, App));
    label_text(label, "Use the slider!");
    button_text(button, "Start");
    button_OnClick(button, listener(app, i_OnStart, App));
    layout_view(layout, view, 0, 0);
    layout_slider(layout, slider, 0, 1);
    layout_label(layout, label, 0, 2);
    layout_button(layout, button, 0, 3);
    layout_vexpand(layout, 0);
    layout_vmargin(layout, 0, 10);
    layout_vmargin(layout, 2, 10);
    layout_margin(layout, 10);
    panel_layout(panel, layout);
    app->view = view;
    app->slider = slider;
    app->button = layout_cell(layout, 0, 3);
    return panel;
}
```

```
/*-----------------------------------------------------------------------------*/

static void i_init_game(App *app)
{
    real32_t hoffset;
    Brick *brick = NULL;
    uint32_t j, i;

    app->color[0] = color_rgb(255, 0, 0);
    app->color[1] = color_rgb(0, 255, 0);
    app->color[2] = color_rgb(0, 0, 255);
    app->color[3] = color_rgb(0, 255, 255);

    hoffset = i_BRICK_SEPARATION;
    brick = app->bricks;

    app->is_running = FALSE;
    app->brick_width = (1 - ((real32_t)i_BRICKS_PER_ROW + 1) *
        ↪ i_BRICK_SEPARATION) / (real32_t)i_BRICKS_PER_ROW;

    for (j = 0; j < i_NUM_ROWS; ++j)
    {
        real32_t woffset = i_BRICK_SEPARATION;

        for (i = 0; i < i_BRICKS_PER_ROW; ++i)
        {
            brick->x = woffset;
            brick->y = hoffset;
            brick->is_visible = TRUE;
            brick->color = (uint8_t)j;
            woffset += app->brick_width + i_BRICK_SEPARATION;
            brick++;
        }

        hoffset += i_BRICK_HEIGHT + i_BRICK_SEPARATION;
    }

    app->player_pos = slider_get_value(app->slider);
    app->ball_x = .5f;
    app->ball_y = .5f;
    app->ball_dir.x = .3f;
    app->ball_dir.y = -.1f;
    app->ball_speed = .6f;
    v2d_normf(&app->ball_dir);
}

/*-----------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
```

```
    unref(app);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STDRES);
    window_panel(app->window, panel);
    window_origin(app->window, v2df(200, 200));
    window_title(app->window, "Bricks - A 2D Game");
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    i_init_game(app);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/

static bool_t i_collision(Brick *brick, real32_t brick_width, real32_t ball_x,
    ↪ real32_t ball_y)
{
    if (ball_x + i_BALL_RADIUS < brick->x)
        return FALSE;
    if (ball_x - i_BALL_RADIUS > brick->x + brick_width)
        return FALSE;
    if (ball_y + i_BALL_RADIUS < brick->y)
        return FALSE;
    if (ball_y - i_BALL_RADIUS > brick->y + i_BRICK_HEIGHT)
        return FALSE;
    return TRUE;
}

/*---------------------------------------------------------------------------*/

static void i_update(App *app, const real64_t prtime, const real64_t ctime)
{
    if (app->is_running == TRUE)
    {
        real32_t step = (real32_t)(ctime - prtime);
```

```
    bool_t collide;
    uint32_t i;

    /* Update ball position */
    app->ball_x += step * app->ball_speed * app->ball_dir.x;
    app->ball_y += step * app->ball_speed * app->ball_dir.y;

    /* Collision with limits */
    if (app->ball_x + i_BALL_RADIUS >= 1.f && app->ball_dir.x >= 0.f)
        app->ball_dir.x = - app->ball_dir.x;

    if (app->ball_x - i_BALL_RADIUS <= 0.f && app->ball_dir.x <= 0.f)
        app->ball_dir.x = - app->ball_dir.x;

    if (app->ball_y - i_BALL_RADIUS <= 0.f && app->ball_dir.y <= 0.f)
        app->ball_dir.y = - app->ball_dir.y;

    /* Collision with bricks */
    collide = FALSE;
    for (i = 0; i < NUM_BRICKS; ++i)
    {
        if (app->bricks[i].is_visible == TRUE)
        {
            if (i_collision(&app->bricks[i], app->brick_width, app->ball_x,
                ↪  app->ball_y) == TRUE)
            {
                app->bricks[i].is_visible = FALSE;
                if (collide == FALSE)
                {
                    real32_t brick_x = app->bricks[i].x + .5f * app->
                        ↪ brick_width;
                    app->ball_dir.x = 5.f * (app->ball_x - brick_x);
                    app->ball_dir.y = - app->ball_dir.y;
                    v2d_normf(&app->ball_dir);
                    collide = TRUE;
                }
            }
        }
    }

    /* Collision with player */
    {
        Brick player;
        player.x = app->player_pos - app->brick_width;
        player.y = 1.f - i_BRICK_HEIGHT - i_BRICK_SEPARATION;
        if (i_collision(&player, 2.f * app->brick_width, app->ball_x, app->
            ↪ ball_y) == TRUE)
        {
            app->ball_dir.x = 5.f * (app->ball_x - app->player_pos);
            app->ball_dir.y = - app->ball_dir.y;
            v2d_normf(&app->ball_dir);
```

```
            }
        }

        /* Game Over */
        if (app->ball_y + i_BALL_RADIUS >= 1.f)
        {
            i_init_game(app);
            cell_enabled(app->button, TRUE);
        }
    }

    view_update(app->view);
}

/*--------------------------------------------------------------------------*/

#include "osmain.h"
osmain_sync(.04, i_create, i_destroy, i_update, "", App)
```

Capítulo 23

Fractals

En esta aplicación creamos una imagen de forma procedimental calculando el color de cada píxel mediante algoritmos fractales . Algunos de los resultados más fascinantes producidos por un sistema dinámico se dan cuando iteramos una función de variable compleja en lugar de real. Este es el caso de los **conjuntos de Julia**. El **código fuente** está en la caperta `/src/demo/fractals` de la distribución del SDK.

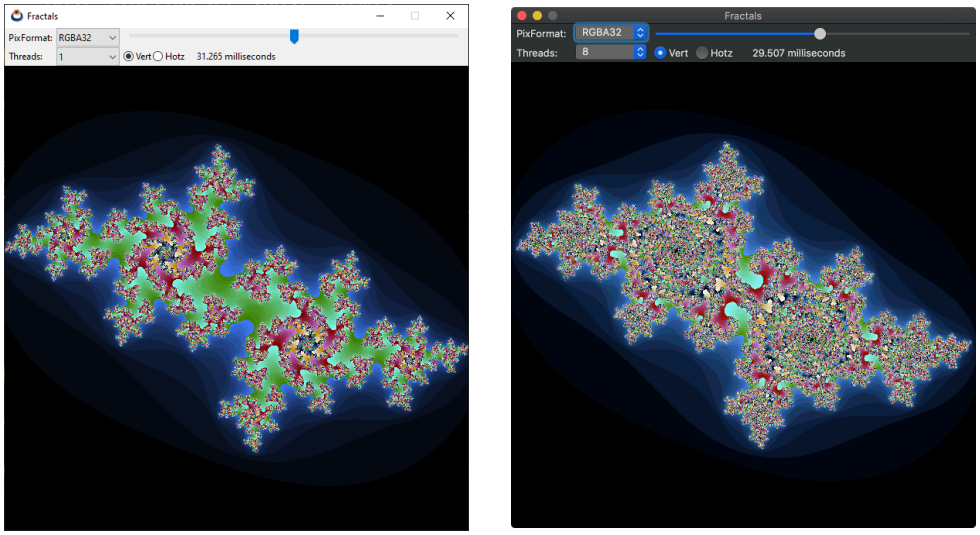


Figura 23.1: Aplicación **Fractals** versión Windows y macOS.

Debido a la gran carga computacional de este algoritmo hemos dividido el cálculo en varios hilos (Figura 23.3). Este problema es fácilmente paralelizable simplemente fraccionando la imagen, gracias a que cada píxel se obtiene de forma independiente.

Listado 23.1: demo/fractals/fractals.c

```
/* Multi-threaded fractals */
```

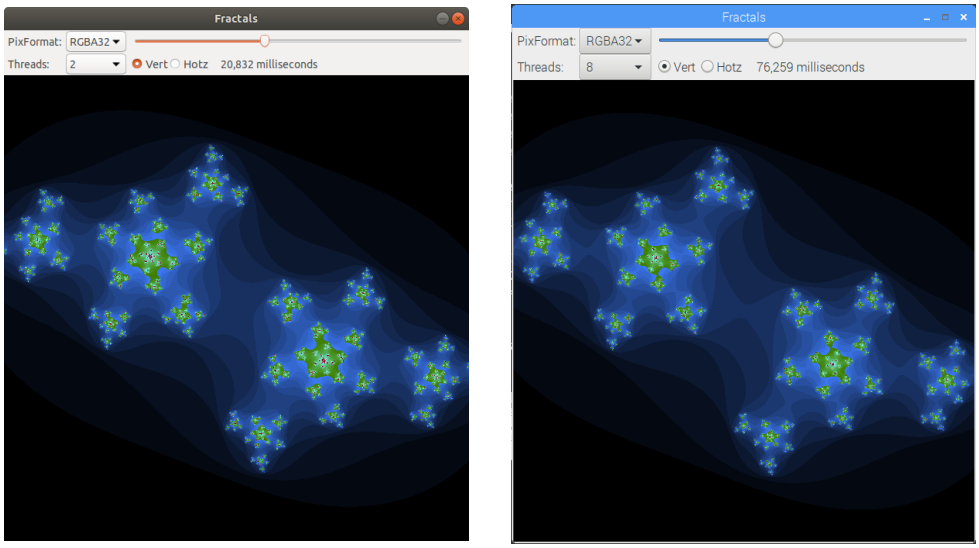


Figura 23.2: Versión Ubuntu y Raspbian.

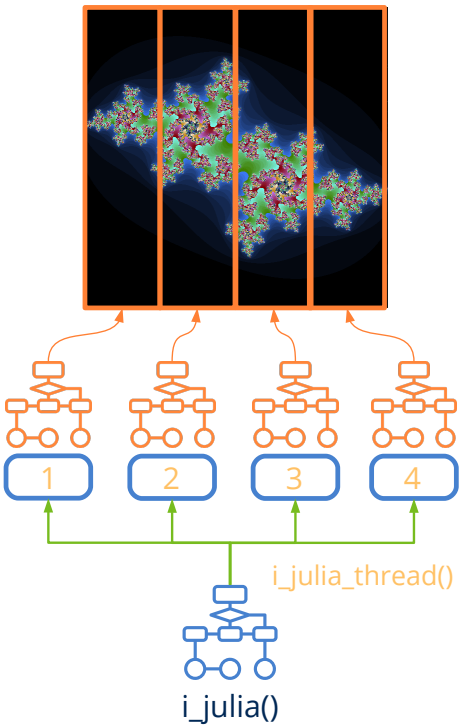


Figura 23.3: Colaboración de varios hilos de ejecución.

```
#include <nappgui.h>
```

```
typedef struct _app_t App;
typedef struct _thdata_t ThData;

struct _app_t
{
    Window *window;
    ImageView *view;
    Label *time_label;
    Clock *clock;
    uint32_t threads;
    bool_t vertical;
    real64_t fct;
};

struct _thdata_t
{
    real64_t fct;
    real64_t kreal;
    real64_t kimag;
    Pixbuf *pixbuf;
    uint32_t i;
    uint32_t j;
    uint32_t width;
    uint32_t height;
};

static const real64_t i_FCT = 2.85;
static const uint32_t i_ITERATIONS = 512;
static const uint32_t i_WIDTH = 601;
static const uint32_t i_HEIGHT = 601;

/*--------------------------------------------------------------------------*/

static uint32_t i_inset(real64_t zreal, real64_t zimag, real64_t creal,
    ↪ real64_t cimag)
{
    uint32_t i;
    for(i = 0; i < i_ITERATIONS; ++i)
    {
        real64_t ztmp, zdist;
        ztmp = zreal * zreal - zimag * zimag;
        zimag = zreal * zimag + zreal * zimag;
        zreal = ztmp;
        zreal = zreal + creal;
        zimag = zimag + cimag;
        zdist = zimag * zimag  + zreal * zreal;
        if (zdist > 3)
            return i;
    }

    return 0;
```

```
}

/*--------------------------------------------------------------------------*/

static uint32_t i_julia_thread(ThData *data)
{
    real64_t fct = data->fct;
    uint32_t imgwidth = pixbuf_width(data->pixbuf);
    real64_t freal = fct / imgwidth;
    real64_t fimag = fct / pixbuf_height(data->pixbuf);
    real64_t kreal = data->kreal;
    real64_t kimag = data->kimag;
    uint32_t val;
    real64_t creal, cimag;
    register uint32_t stj = data->j;
    register uint32_t edj = data->j + data->height;
    register uint32_t sti = data->i;
    register uint32_t edi = data->i + data->width;
    register uint32_t i, j;

    for(j = stj; j < edj; ++j)
    {
        cimag = fimag * j - (fct / 2);

        for(i = sti; i < edi; ++i)
        {
            creal = freal * i - (fct / 2);
            val = i_inset(creal, cimag, kreal, kimag);
            if (val > 0)
            {
                uint8_t n_val = (uint8_t)(val % 255);
                if ( val < ( i_ITERATIONS >> 1 ) )
                    val = color_rgb((uint8_t)(n_val << 2), (uint8_t)(n_val <<
                        ↪ 3), (uint8_t)(n_val << 4));
                else
                    val = color_rgb((uint8_t)(n_val << 4), (uint8_t)(n_val <<
                        ↪ 2), (uint8_t)(n_val << 5));
            }
            else
            {
                val = kCOLOR_BLACK;
            }

            pixbuf_set(data->pixbuf, i, j, val);
        }
    }

    return 5;
}

/*--------------------------------------------------------------------------*/
```

```
static void i_julia(const uint32_t nthreads, const bool_t vertical, const
    ↪ real64_t fct, const real64_t kreal, const real64_t kimag, Pixbuf *pixbuf
    ↪ )
{
    ThData data[8];
    uint32_t width = pixbuf_width(pixbuf);
    uint32_t height = pixbuf_height(pixbuf);
    data[0].fct = fct;
    data[0].kreal = kreal;
    data[0].kimag = kimag;
    data[0].pixbuf = pixbuf;

    if (nthreads == 1)
    {
        data[0].i = 0;
        data[0].j = 0;
        data[0].width = width;
        data[0].height = height;
        i_julia_thread(&data[0]);
    }
    else
    {
        Thread *thread[8];

        register uint32_t i;
        if (vertical == TRUE)
        {
            uint32_t twidth = width / nthreads;
            for (i = 0; i < nthreads; ++i)
            {
                data[i] = data[0];
                data[i].i = i * twidth;
                data[i].j = 0;
                data[i].width = twidth;
                data[i].height = height;
            }

            data[nthreads-1].width += (width - (twidth * nthreads));
        }
        else
        {
            uint32_t theight = height / nthreads;
            for (i = 0; i < nthreads; ++i)
            {
                data[i] = data[0];
                data[i].i = 0;
                data[i].j = i * theight;
                data[i].width = width;
                data[i].height = theight;
            }
```

```
            data[nthreads-1].height += (height - (theight * nthreads));
        }

        for (i = 0; i < nthreads; ++i)
            thread[i] = bthread_create(i_julia_thread, &data[i], ThData);

        for (i = 0; i < nthreads; ++i)
        {
            uint32_t thid = bthread_wait(thread[i]);
            cassert_unref(thid == 5, thid);
            bthread_close(&thread[i]);
        }
    }
}

/*---------------------------------------------------------------------------*/

static void i_image(App *app)
{
    Pixbuf *pixbuf = pixbuf_create(i_WIDTH, i_HEIGHT, ekRGBA32);
    real64_t rfactor = app->fct / i_WIDTH;
    real64_t ifactor = app->fct / i_HEIGHT;
    real64_t kreal = rfactor * 307 - 2;
    real64_t kimag = ifactor * 184 - 1.4;
    Image *image = NULL;
    real64_t timems;
    String *str;
    clock_reset(app->clock);
    i_julia(app->threads, app->vertical, app->fct, kreal, kimag, pixbuf);
    timems = 1000. * clock_elapsed(app->clock);
    str = str_printf("%.3f milliseconds", timems);
    label_text(app->time_label, tc(str));
    str_destroy(&str);
    image = image_from_pixbuf(pixbuf, NULL);
    imageview_image(app->view, image);
    image_destroy(&image);
    pixbuf_destroy(&pixbuf);
}

/*---------------------------------------------------------------------------*/

static void i_OnSlider(App *app, Event *e)
{
    const EvSlider *p = event_params(e, EvSlider);
    real64_t st = i_FCT - 1;
    real64_t ed = i_FCT + 1;
    app->fct = ((ed - st) * p->pos) + st;
    i_image(app);
}
```

```
/*---------------------------------------------------------------------------*/

static void i_OnThreads(App *app, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    switch(p->index) {
    case 0: app->threads = 1; break;
    case 1: app->threads = 2; break;
    case 2: app->threads = 3; break;
    case 3: app->threads = 4; break;
    case 4: app->threads = 8; break; }
    i_image(app);
}

/*---------------------------------------------------------------------------*/

static void i_OnVertical(App *app, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    app->vertical = p->index == 0 ? TRUE : FALSE;
    i_image(app);
}

/*---------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(1, 3);
    Layout *layout2 = layout_create(5, 1);
    Label *label1 = label_create();
    Label *label2 = label_create();
    PopUp *popup = popup_create();
    Slider *slider = slider_create();
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    ImageView *view = imageview_create();
    label_text(label1, "Threads:");
    popup_add_elem(popup, "1", NULL);
    popup_add_elem(popup, "2", NULL);
    popup_add_elem(popup, "3", NULL);
    popup_add_elem(popup, "4", NULL);
    popup_add_elem(popup, "8", NULL);
    popup_selected(popup, 0);
    popup_OnSelect(popup, listener(app, i_OnThreads, App));
    slider_value(slider, .5f);
    slider_OnMoved(slider, listener(app, i_OnSlider, App));
    button_text(button1, "Vert");
    button_text(button2, "Hotz");
    button_state(button1, ekGUI_ON);
    button_OnClick(button1, listener(app, i_OnVertical, App));
```

```
    imageview_size(view, s2di(i_WIDTH, i_HEIGHT));
    layout_slider(layout1, slider, 0, 0);
    layout_label(layout2, label1, 0, 0);
    layout_popup(layout2, popup, 1, 0);
    layout_button(layout2, button1, 2, 0);
    layout_button(layout2, button2, 3, 0);
    layout_label(layout2, label2, 4, 0);
    layout_halign(layout2, 4, 0, ekJUSTIFY);
    layout_hexpand(layout2, 4);
    layout_layout(layout1, layout2, 0, 1);
    layout_imageview(layout1, view, 0, 2);
    layout_vmargin(layout1, 1, 5);
    layout_margin2(layout2, 0, 5);
    layout_hmargin(layout2, 0, 5);
    layout_hmargin(layout2, 1, 10);
    layout_hmargin(layout2, 2, 5);
    layout_hmargin(layout2, 3, 15);
    panel_layout(panel, layout1);
    app->fct = i_FCT;
    app->threads = 1;
    app->vertical = TRUE;
    app->view = view;
    app->time_label = label2;
    return panel;
}

/*---------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STD);
    app->clock = clock_create(0);
    i_image(app);
    window_panel(app->window, panel);
    window_title(app->window, "Fractals");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}
```

```
/*----------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    clock_destroy(&(*app)->clock);
    heap_delete(app, App);
}

/*----------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 24

Bode

En este proyecto abordamos la construcción de una interfaz de usuario interactiva para los **Diagramas de Bode**, herramienta muy utilizada en Ingeniería del Control (Figura 24.1). El módulo de cálculo ha sido escrito en lenguaje C por Javier Gil Chica[1], profesor titular del Departamento de Física de la Universidad de Alicante. El código fuente al completo está disponible disponible en la carpeta `/src/demo/bode` de la distribución del SDK.

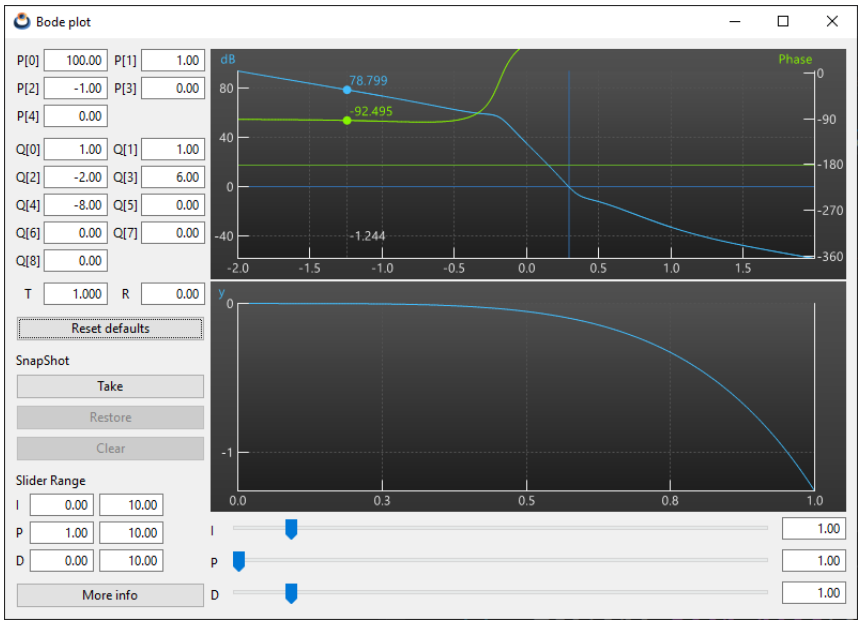


Figura 24.1: Versión Windows.

La ventana principal se ha dividido verticalmente en dos partes, mediante un `layout`

[1] `mailto:francisco.gil@ua.es`

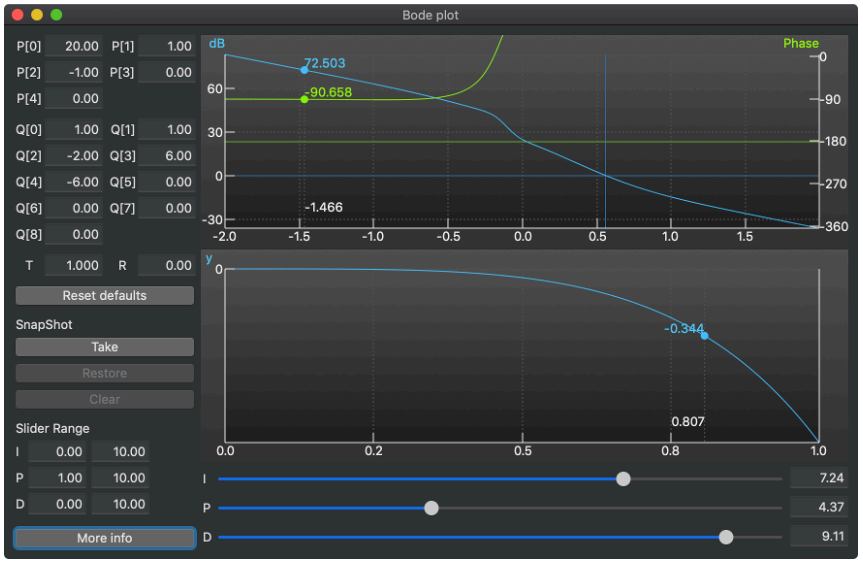


Figura 24.2: Versión macOS.

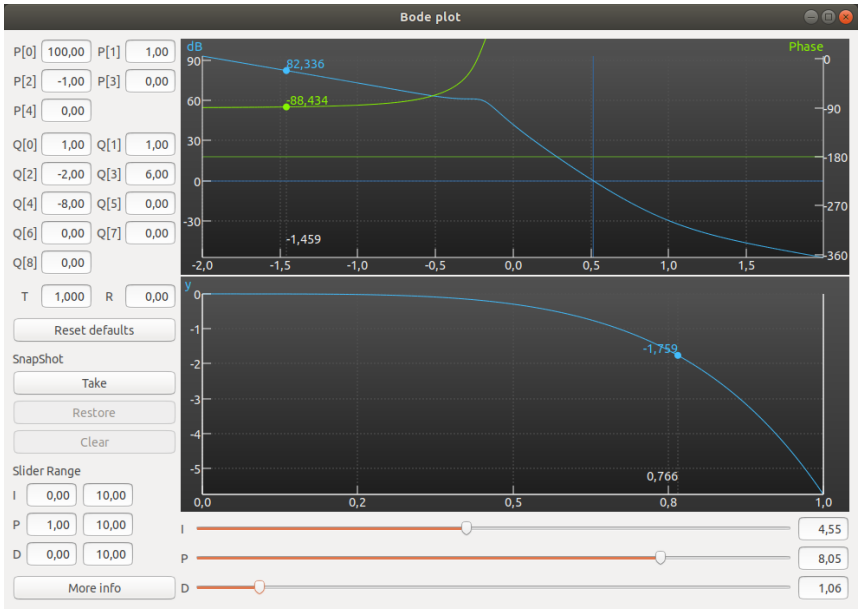


Figura 24.3: Versión Linux Ubuntu.

`(2,1)` (Figura 24.4). En la parte izquierda tenemos los parámetros `P, Q, T, R` y algunos botones. Se han utilizado los sublayouts `i_coeffs(4,9)` y `i_ranges(3,3)` para agrupar controles. En la zona de la derecha se ubican dos controles de dibujo `View` para las gráficas y otro sublayout `i_sliders(3,3)` con los parámetros `I, P, D`.

El re-dimensionado horizontal se realiza íntegramente sobre la celda derecha (gráficas y sliders), manteniendo la zona de parámetros un tamaño horizontal constante. Durante el re-dimensionado vertical crecerán las gráficas con una proporción del 50 % cada una. Para la parte izquierda, se ha reservado una celda vacía, que se expandirá horizontalmente, alineando el botón `[More Info]` al borde inferior de la ventana.

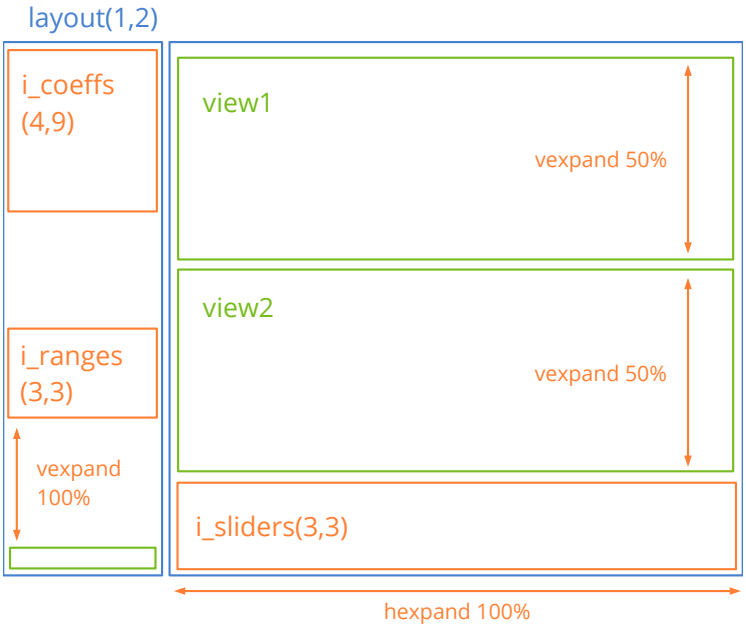


Figura 24.4: Distribución de la interfaz de usuario de Bode.

Listado 24.1: demo/bode/bdview.c

```
/* Bode View */

#include "bdview.h"
#include "bdctrl.h"
#include <gui/guiall.h>

static const real32_t kEDIT_WIDTH = 60;

/*---------------------------------------------------------------------------*/

static Cell *i_coeff(Layout *layout, const char_t *text, const uint32_t col,
    ↪ const uint32_t row)
{
    Label *label = label_create();
    Edit *edit = edit_create();
    label_text(label, text);
    edit_align(edit, ekRIGHT);
    layout_halign(layout, col * 2, row, ekCENTER);
```

```
    layout_label(layout, label, col * 2, row);
    layout_edit(layout, edit, col * 2 + 1, row);
    return layout_cell(layout, col * 2 + 1, row);
}

/*---------------------------------------------------------------------------*/

static Layout *i_coeffs(void)
{
    Layout *layout = layout_create(4, 9);
    cell_dbind(i_coeff(layout, "P[0]", 0, 0), Params, real32_t, P[0]);
    cell_dbind(i_coeff(layout, "P[1]", 1, 0), Params, real32_t, P[1]);
    cell_dbind(i_coeff(layout, "P[2]", 0, 1), Params, real32_t, P[2]);
    cell_dbind(i_coeff(layout, "P[3]", 1, 1), Params, real32_t, P[3]);
    cell_dbind(i_coeff(layout, "P[4]", 0, 2), Params, real32_t, P[4]);
    cell_dbind(i_coeff(layout, "Q[0]", 0, 3), Params, real32_t, Q[0]);
    cell_dbind(i_coeff(layout, "Q[1]", 1, 3), Params, real32_t, Q[1]);
    cell_dbind(i_coeff(layout, "Q[2]", 0, 4), Params, real32_t, Q[2]);
    cell_dbind(i_coeff(layout, "Q[3]", 1, 4), Params, real32_t, Q[3]);
    cell_dbind(i_coeff(layout, "Q[4]", 0, 5), Params, real32_t, Q[4]);
    cell_dbind(i_coeff(layout, "Q[5]", 1, 5), Params, real32_t, Q[5]);
    cell_dbind(i_coeff(layout, "Q[6]", 0, 6), Params, real32_t, Q[6]);
    cell_dbind(i_coeff(layout, "Q[7]", 1, 6), Params, real32_t, Q[7]);
    cell_dbind(i_coeff(layout, "Q[8]", 0, 7), Params, real32_t, Q[8]);
    cell_dbind(i_coeff(layout, "T", 0, 8), Params, real32_t, T);
    cell_dbind(i_coeff(layout, "R", 1, 8), Params, real32_t, R);
    layout_hsize(layout, 1, kEDIT_WIDTH);
    layout_hsize(layout, 3, kEDIT_WIDTH);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_vmargin(layout, 2, 10);
    layout_vmargin(layout, 3, 5);
    layout_vmargin(layout, 4, 5);
    layout_vmargin(layout, 5, 5);
    layout_vmargin(layout, 6, 5);
    layout_vmargin(layout, 7, 10);
    layout_hmargin(layout, 1, 5);
    layout_hmargin(layout, 0, 3);
    layout_hmargin(layout, 2, 3);
    return layout;
}

/*---------------------------------------------------------------------------*/

static void i_range(Layout *layout, const char_t *text, const uint32_t i)
{
    Label *label = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    label_text(label, text);
    edit_align(edit1, ekRIGHT);
```

```
    edit_align(edit2, ekRIGHT);
    layout_label(layout, label, 0, i);
    layout_edit(layout, edit1, 1, i);
    layout_edit(layout, edit2, 2, i);
}

/*--------------------------------------------------------------------------*/

static Layout *i_ranges(void)
{
    Layout *layout = layout_create(3, 3);
    i_range(layout, "I", 0);
    i_range(layout, "P", 1);
    i_range(layout, "D", 2);
    layout_hsize(layout, 1, kEDIT_WIDTH);
    layout_hsize(layout, 2, kEDIT_WIDTH);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_hmargin(layout, 0, 5);
    layout_hmargin(layout, 1, 5);
    cell_dbind(layout_cell(layout, 1, 0), Params, real32_t, KRg[0]);
    cell_dbind(layout_cell(layout, 2, 0), Params, real32_t, KRg[1]);
    cell_dbind(layout_cell(layout, 1, 1), Params, real32_t, KRg[2]);
    cell_dbind(layout_cell(layout, 2, 1), Params, real32_t, KRg[3]);
    cell_dbind(layout_cell(layout, 1, 2), Params, real32_t, KRg[4]);
    cell_dbind(layout_cell(layout, 2, 2), Params, real32_t, KRg[5]);
    return layout;
}

/*--------------------------------------------------------------------------*/

static Layout *i_left(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 10);
    Layout *layout1 = i_coeffs();
    Button *button = button_push();
    Label *label = label_create();
    Button *button2 = button_push();
    Button *button3 = button_push();
    Button *button4 = button_push();
    Label *label2 = label_create();
    Layout *layout2 = i_ranges();
    Button *button5 = button_push();
    button_text(button, "Reset defaults");
    button_text(button2, "Take");
    button_text(button3, "Restore");
    button_text(button4, "Clear");
    button_text(button5, "More info");
    label_text(label, "SnapShot");
    label_text(label2, "Slider Range");
    layout_layout(layout, layout1, 0, 0);
```

```
    layout_button(layout, button, 0, 1);
    layout_label(layout, label, 0, 2);
    layout_button(layout, button2, 0, 3);
    layout_button(layout, button3, 0, 4);
    layout_button(layout, button4, 0, 5);
    layout_label(layout, label2, 0, 6);
    layout_layout(layout, layout2, 0, 7);
    layout_button(layout, button5, 0, 9);
    layout_halign(layout, 0, 7, ekLEFT);
    layout_vmargin(layout, 0, 10);
    layout_vmargin(layout, 1, 10);
    layout_vmargin(layout, 2, 5);
    layout_vmargin(layout, 3, 5);
    layout_vmargin(layout, 4, 5);
    layout_vmargin(layout, 5, 10);
    layout_vmargin(layout, 6, 5);
    layout_vmargin(layout, 7, 10);
    layout_vexpand(layout, 8);
    ctrl_reset(ctrl, button);
    ctrl_take(ctrl, layout_cell(layout, 0, 3));
    ctrl_restore(ctrl, layout_cell(layout, 0, 4));
    ctrl_clear(ctrl, layout_cell(layout, 0, 5));
    ctrl_info(ctrl, button5);
    return layout;
}

/*----------------------------------------------------------------------------*/

static void i_slider_K(Layout *layout, const char_t *title, const uint32_t row)
{
    Label *label = label_create();
    Slider* slider = slider_create();
    Edit* edit = edit_create();
    label_text(label, title);
    edit_align(edit, ekRIGHT);
    layout_label(layout, label, 0, row);
    layout_slider(layout, slider, 1, row);
    layout_edit(layout, edit, 2, row);
}

/*----------------------------------------------------------------------------*/

static Layout *i_sliders(Ctrl *ctrl)
{
    Layout *layout = layout_create(3, 3);
    i_slider_K(layout, "I", 0);
    i_slider_K(layout, "P", 1);
    i_slider_K(layout, "D", 2);
    layout_hsize(layout, 2, kEDIT_WIDTH);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
```

```
    layout_hmargin(layout, 0, 5);
    layout_hmargin(layout, 1, 5);
    layout_hexpand(layout, 1);
    cell_dbind(layout_cell(layout, 1, 0), Params, real32_t, K[0]);
    cell_dbind(layout_cell(layout, 2, 0), Params, real32_t, K[0]);
    cell_dbind(layout_cell(layout, 1, 1), Params, real32_t, K[1]);
    cell_dbind(layout_cell(layout, 2, 1), Params, real32_t, K[1]);
    cell_dbind(layout_cell(layout, 1, 2), Params, real32_t, K[2]);
    cell_dbind(layout_cell(layout, 2, 2), Params, real32_t, K[2]);
    ctrl_slider1(ctrl, layout_cell(layout, 1, 0));
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout* i_right(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 3);
    Layout* layout1 = i_sliders(ctrl);
    View* view1 = view_create();
    View* view2 = view_create();
    layout_view(layout, view1, 0, 0);
    layout_view(layout, view2, 0, 1);
    layout_layout(layout, layout1, 0, 2);
    layout_vmargin(layout, 0, 2);
    layout_vmargin(layout, 1, 5);
    layout_vexpand2(layout, 0, 1, .5f);
    ctrl_view1(ctrl, view1);
    ctrl_view2(ctrl, view2);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Panel *i_panel(Ctrl *ctrl)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(2, 1);
    Layout *layout1 = i_left(ctrl);
    Layout* layout2 = i_right(ctrl);
    layout_layout(layout, layout1, 0, 0);
    layout_layout(layout, layout2, 1, 0);
    layout_hmargin(layout, 0, 5);
    layout_hexpand(layout, 1);
    layout_margin(layout, 10);
    panel_layout(panel, layout);
    layout_dbind(layout1, NULL, Params);
    layout_dbind(layout2, NULL, Params);
    cell_dbind(layout_cell(layout, 0, 0), Model, Params, cparams);
    cell_dbind(layout_cell(layout, 1, 0), Model, Params, cparams);
    layout_dbind(layout, listener(ctrl, ctrl_OnModelChange, Ctrl), Model);
```

```
    ctrl_layout(ctrl, layout);
    return panel;
}

/*---------------------------------------------------------------------------*/

Window* bdview_create(Ctrl *ctrl)
{
    Panel *panel = i_panel(ctrl);
    Window *window = window_create(ekWINDOW_STDRES);
    window_panel(window, panel);
    window_title(window, "Bode plot");
    return window;
}
```

Capítulo 25

Products

En este proyecto afrontaremos la construcción de una aplicación que permite navegar por una base de datos de productos obtenida desde un servidor Web (Figura 25.1). Este patrón cliente-servidor es ampliamente utilizado a día de hoy, por lo que dispondremos de una base estable para crear cualquier aplicación basada en este modelo. El **código fuente** está en la caperta `/src/demo/products` de la distribución del SDK.

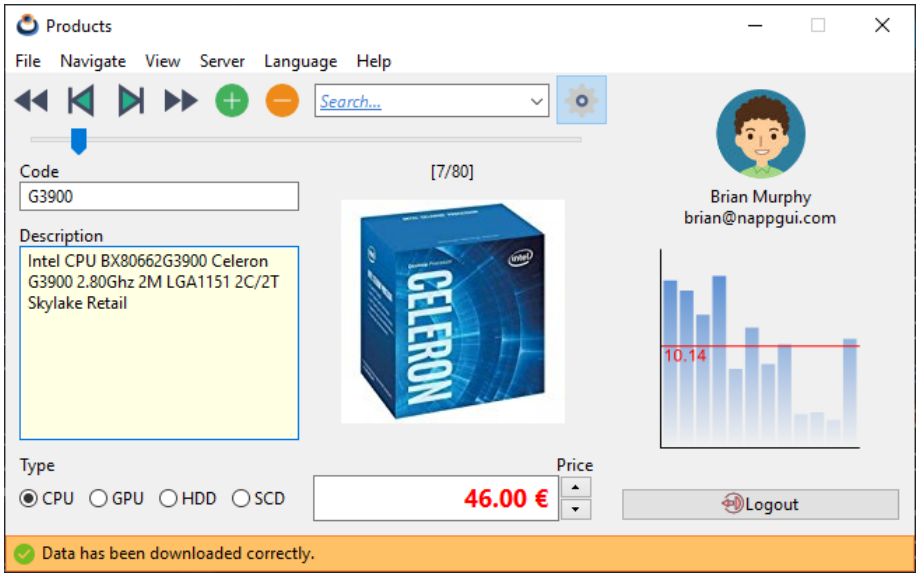


Figura 25.1: Aplicación *Products*, versión Windows.

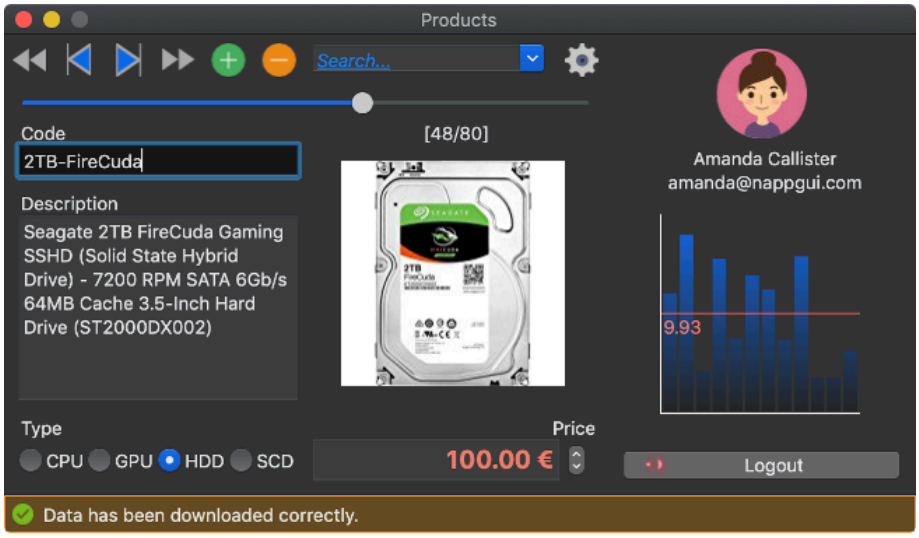


Figura 25.2: Versión macOS.

25.1. Especificaciones

- La base de datos es remota y accederemos a ella a través de servicios Web que encapsularán los datos en JSON. Para obtener los productos utilizaremos este servicio[1]

[1] `http://serv.nappgui.com/dproducts.php`

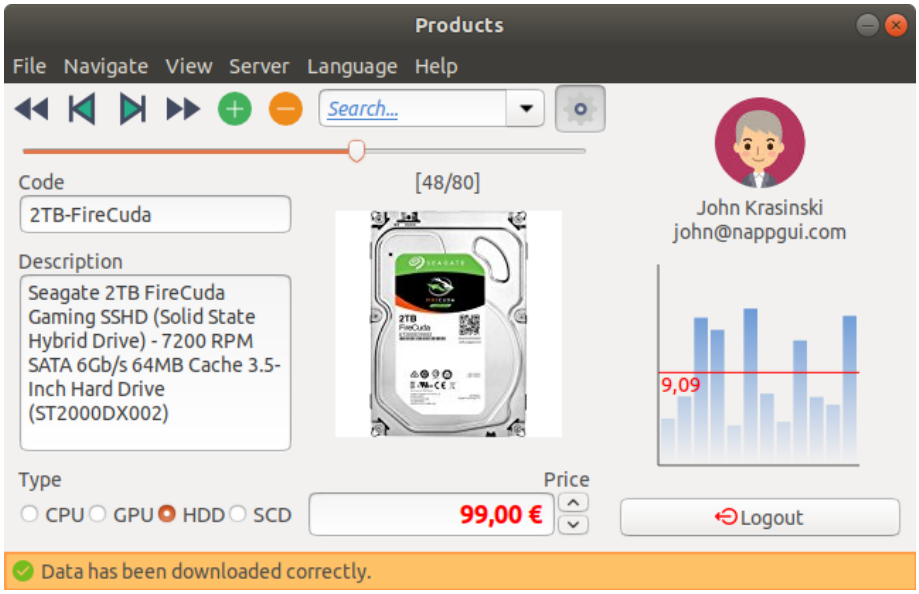


Figura 25.3: Versión Linux/GTK+.

y para registrar a un usuario este otro[2]. Tenemos cuatro **usuarios** dados de alta en nuestra base de datos: *amanda*, *brenda*, *brian* y *john* todos con el **password** 1234.

- La base de datos remota es de solo lectura. No tenemos servicios web para editarla.
- En el momento que un usuario se registre, se descargarán automáticamente todos los artículos.
- Se mostrará un pequeño gráfico con las estadísticas de ventas de cada producto.
- Se puede editar la base de datos de forma local, así como añadir o eliminar registros.
- Se puede exportar la base de datos local a disco, así como importarla.
- Tendremos los controles típicos de navegación: Primero, último, siguiente, anterior.
- Podremos establecer un filtro por descripción. Solo se mostrarán aquellos productos cuya descripción coincida parcialmente con el filtro.
- La interfaz estará en siete idiomas: Inglés, Español, Portugués, Italiano, Vietnamita, Ruso y Japonés. Podremos cambiar de idioma sin necesidad de cerrar la aplicación.
- La aplicación debe correr en Windows, macOS y Linux.

[2] `http://serv.nappgui.com/duser.php?user=amanda&pass=1234`

25.2. Modelo-Vista-Controlador

Dado que este programa tiene un nivel de complejidad medio, lo fragmentaremos en tres partes utilizando el conocido patrón modelo-vista-controlador **MVC** (Figura 25.4).

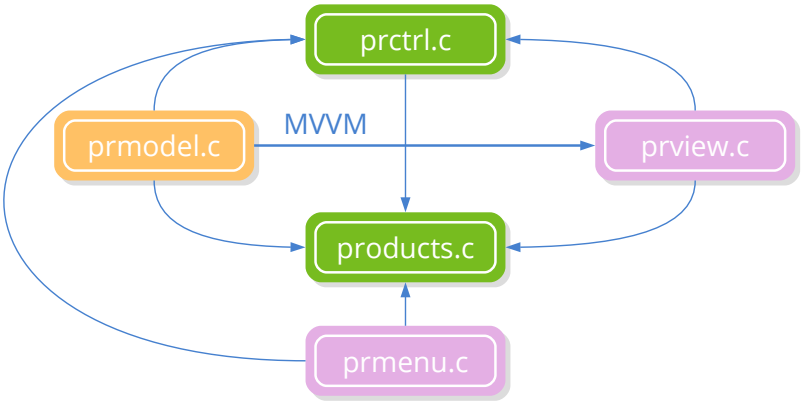


Figura 25.4: Módulos MVC que componen la aplicación.

- **Modelo:** Se ocupará de los datos propiamente dichos, la conexión con el servidor y la lectura/escritura en disco. Se implementará en `prmodel.c`.
- **Vista:** Aquí implementaremos la capa de presentación de datos, compuesta por la ventana principal (en `prview.c`) y la barra de menú (en `prmenu.c`).
- **Controlador:** Se ocupará de la lógica del programa `prctrl.c`. Responderá a los eventos del usuario y mantendrá la coherencia entre el modelo y la vista. Debido a la cantidad de trabajo extra que supone sincronizar cada campo de la estructura con los controles de interfaz, haremos uso del patrón *Model-View-ViewModel* **MVVM** donde los datos del modelo se sincronizarán automáticamente con la interfaz y los canales E/S.
- **Main:** Módulo principal `products.c`. Contiene la función `osmain` y carga los tres actores anteriores.

25.3. Modelo

El modelo de datos de esta aplicación es bastante sencillo (Listado 25.1), ya que únicamente requiere manipular un array de estructuras tipo `Product`.

Listado 25.1: Estructuras que forman el modelo de datos.

```
typedef struct _model_t Model;
typedef struct _product_t Product;

typedef enum _type_t
```

```
{
    ekCPU,
    ekGPU,
    ekHDD,
    ekSCD
} type_t;

struct _product_t
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
};

struct _model_t
{
    ArrSt(uint32_t) *filter;
    ArrPt(Product) *products;
};
```

Como paso previo, registraremos las estructuras del modelo lo que nos permitirá automatizar tareas de E/S sin tener que programarlas explícitamente gracias al *"Data binding"* (Página 230) (Listado 25.2).

Listado 25.2: Registro de los `struct` del modelo de datos.

```
dbind_enum(type_t, ekCPU);
dbind_enum(type_t, ekGPU);
dbind_enum(type_t, ekHDD);
dbind_enum(type_t, ekSCD);
dbind(Product, type_t, type);
dbind(Product, String*, code);
dbind(Product, String*, description);
dbind(Product, Image*, image64);
dbind(Product, real32_t, price);
```

25.3.1. JSON WebServices

La lectura de los artículos desde el servidor Web la haremos en dos pasos. Por un lado descargaremos un `Stream` con el JSON mediante HTTP y, posteriormente, lo "parsearemos" a un objeto C (Listado 25.3).

Listado 25.3: Descarga de datos y procesamiento del JSON.

```
wserv_t model_webserv(Model *model)
{
    Stream *stm = http_dget("serv.nappgui.com",80,"/dproducts.php",NULL);
```

```
    if (stm != NULL)
    {
        PJson *json = json_read(stm, NULL, PJson);
        stm_close(&stm);
        ...
```

El JSON de este servicio web[3] consta de una cabecera y una lista de productos (Listado 25.4), por lo que debemos registrar una nueva estructura con el fin de que `json_read` pueda crear correctamente el objeto (Listado 25.5). Observar que emparejamiento JSON-C se lleva a cabo por el nombre del campo, por lo que estos deben ser idénticos (Figura 25.5).

Listado 25.4: Formato del servicio Web.

```
{
    "code":0,
    "size":80,
    "data":[
    {"id":0,
    "code":"i7-8700K",
    "description":"Intel BX80684I78700K 8th Gen Core i7-8700K Processor",
    type":0,
    "price":374.88999999999998635757947340607643127441406 25,
    "image":"cpu_00.jpg",
    "image64":"\/9j\/4AAQSkZJRgABAQ....
    },
    ...
}
```

Listado 25.5: Registrando de cabecera del JSON.

```
typedef struct _pjson_t PJson;
struct _pjson_t
{
    int32_t code;
    uint32_t size;
    ArrPt(Product) *data;
};

dbind(PJson, int32_t, code);
dbind(PJson, uint32_t, size);
dbind(PJson, ArrPt(Product)*, data);
```

25.3.2. Escribir/Leer en disco

La serialización (Listado 25.6) y de-serialización (Listado 25.7) de objetos mediante streams binarios también puede realizarse de forma automática por el mero hecho de

[3] http://serv.nappgui.com/dproducts.php

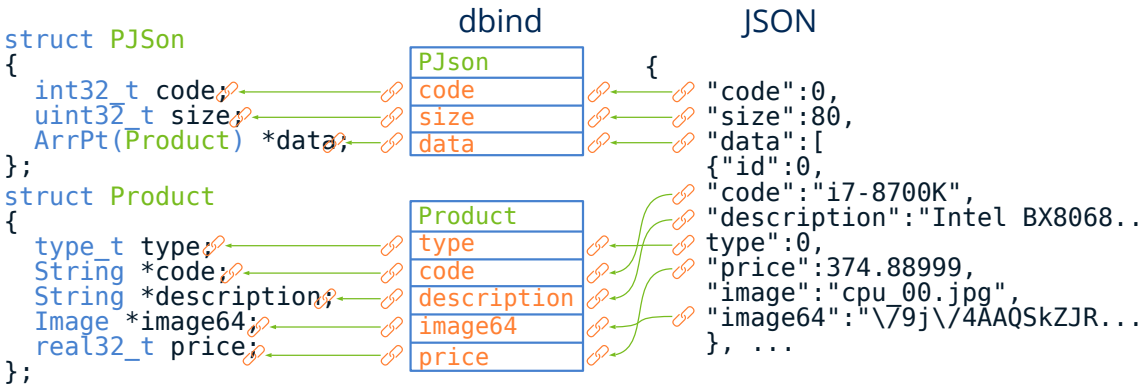


Figura 25.5: `json_read` accede al registro `dbind` para crear un objeto C a partir de un stream JSON.

haber registrado los tipos de datos (Figura 25.6). No es necesario que programemos explícitamente métodos de lectura y escritura de clases.

Listado 25.6: Exportación de la base de datos a disco.

```
bool_t model_export(Model *model, const char_t *pathname, ferror_t *err)
{
    Stream *stm = stm_to_file(pathname, err);
    if (stm != NULL)
    {
        dbind_write(stm, model->products, ArrPt(Product));
        stm_close(&stm);
        return TRUE;
    }

    return FALSE;
}
```

Listado 25.7: Importación de la base de datos desde disco.

```
bool_t model_import(Model *model, const char_t *pathname, ferror_t *err)
{
    Stream *stm = stm_from_file(pathname, err);
    if (stm != NULL)
    {
        ArrPt(Product) *products = dbind_read(stm, ArrPt(Product));
        stm_close(&stm);

        if (products != NULL)
        {
            dbind_destroy(&model->products, ArrPt(Product));
            model->products = products;
            return TRUE;
        }
    }
```

```
    return FALSE;
}
```

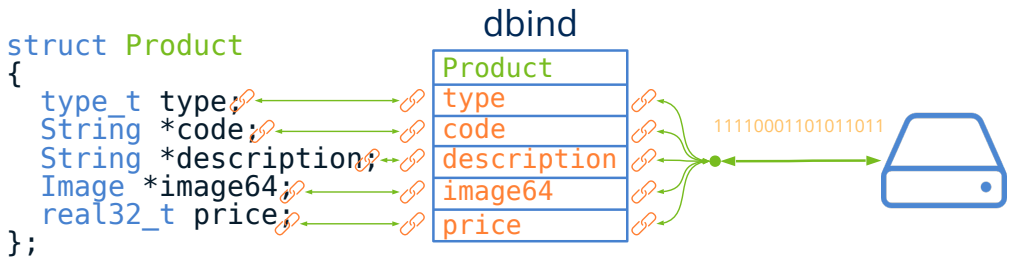


Figura 25.6: (De)serialización de objetos binarios mediante `dbind`.

25.3.3. Añadir/Eliminar registros

Y por último veremos como añadir o eliminar registros a la base de datos utilizando los constructores y destructores que proporciona por defecto `dbind`. En (Listado 25.8) creamos un nuevo artículo y en (Listado 25.9) destruimos otro existente a partir de su índice.

Listado 25.8: Constructor por defecto.

```
void model_add(Model *model)
{
    Product *product = dbind_create(Product);
    arrpt_append(model->products, product, Product);
}
```

Listado 25.9: Destructor.

```
static void i_destroy(Product **product)
{
    dbind_destroy(product, Product);
}

void model_delete(Model *model, const uint32_t index)
{
    arrpt_delete(model->products, index, i_destroy, Product);
}
```

25.4. Vista

Hemos fragmentado el diseño de la ventana principal en varios bloques, implementado cada uno en su propio *sublayout*. En *"Uso de sublayoutsUso de sublayouts"* (Página 396)

y *"Sub-layoutsSub-layouts"* (Página 340) tienes ejemplos al respecto. Comenzamos por un layout de una columna y dos filas (Listado 25.10) (Figura 25.7). En la celda superior ubicaremos un sublayout con otras dos celdas en horizontal: Una para el formulario y otra para el panel de login. La celda inferior la utilizaremos para el *status bar*.

Listado 25.10: Composición del layout principal.

```
static Layout *i_layout(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 2);
    Layout *layout0 = layout_create(2, 1);
    Layout *layout1 = i_form(ctrl);
    Layout *layout2 = i_status_bar(ctrl);
    Panel *panel1 = i_login_panel(ctrl);
    layout_layout(layout0, layout1, 0, 0);
    layout_panel(layout0, panel1, 1, 0);
    layout_layout(layout, layout0, 0, 0);
    layout_layout(layout, layout2, 0, 1);
    return layout;
}
```

A su vez, el layout que integra el formulario, implementado en `i_form()`, está compuesto por tres celdas en vertical (Figura 25.8): Una para la barra de herramientas `i_toolbar()`, otra para el slider de selección y otra para los datos de artículos `i_product()`. Esta última celda es un sublayout de dos columnas y tres filas. En la fila central ubicamos los label `Type` y `Price` y, en las otras dos, cuatro sublayout creados por las funciones `i_code_desc()`, `i_n_img()`, `i_type()` e `i_price()`.

Si observamos el código de `i_product()`, reproducido parcialmente en (Listado 25.11), hemos realizado un *"Formato del LayoutFormato del Layout"* (Página 29), asignando un ancho y alto mínimo para las celdas superiores. También indicamos que la expansión vertical se realice sobre la fila 0, evitando que se expandan las filas 1 y 2, correspondientes a los *label*, los *radiobutton* y el precio.

Listado 25.11: Formato del layout `i_product()`.

```
static Layout *i_product()
{
    Layout *layout = layout_create(2, 3);
    ...
    layout_hsize(layout, 0, 200.f);
    layout_hsize(layout, 1, 200.f);
    layout_vsize(layout, 0, 200.f);
    layout_vexpand(layout, 0);
    ...
}
```

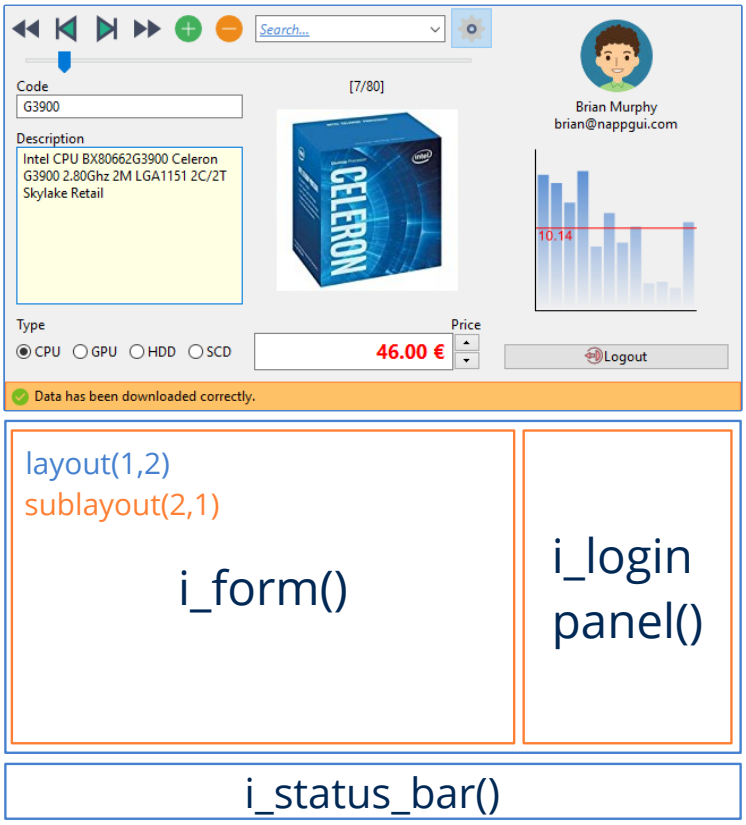


Figura 25.7: Layout principal de la ventana.

25.4.1. Panel de Login

Para el *login* del usuario hemos utilizado un panel con dos layouts diferentes: Uno para registro y otro para mostrar los datos de usuario una vez registrado (Listado 25.12) (Figura 25.9). De esta forma, el controlador podrá alternar entre ellos fácilmente llamando a `panel_visible_layout`. Esta función se encargará de visualizar/ocultar controles y recalcular el tamaño de la ventana, ya que puede haber sufrido variaciones por el cambio de disposición.

Listado 25.12: Creación de un panel multi-layout

```
static Panel *i_login_panel(Ctrl *ctrl)
{
    Panel *panel = panel_create();
    Layout *layout0 = i_login(ctrl);
    Layout *layout1 = i_logout(ctrl);
    panel_layout(panel, layout0);
    panel_layout(panel, layout1);
    return panel;
```

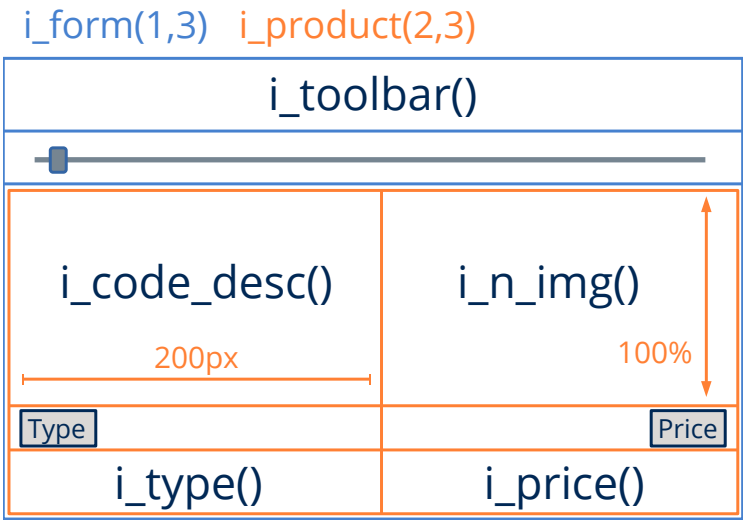


Figura 25.8: Layout que implementa el formulario.

```
}
```

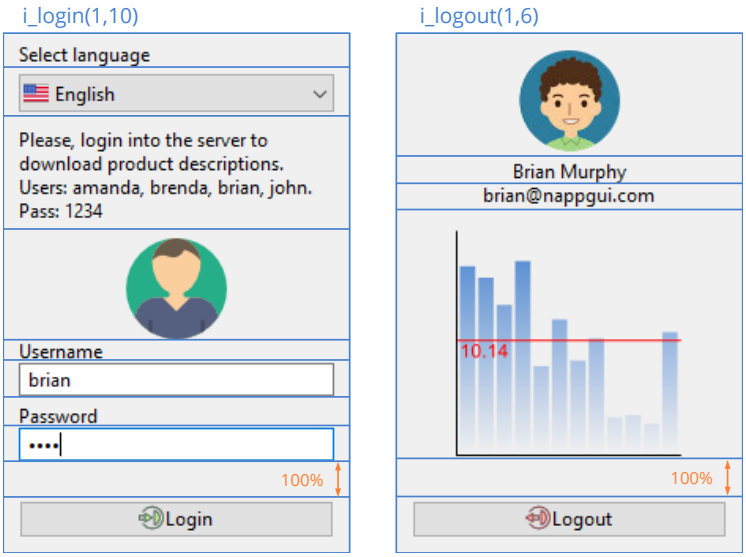


Figura 25.9: Panel de login con dos layouts.

25.4.2. Ocultar columnas

También es posible ocultar el panel de login mediante el menú o el botón correspondiente (Figura 25.10). Esto es sencillo de realizar dentro del controlador, actuando sobre

la columna que contiene dicho panel.

```
layout_show_col(ctrl->layout, 1, state == ekGUI_ON ? TRUE : FALSE);
```

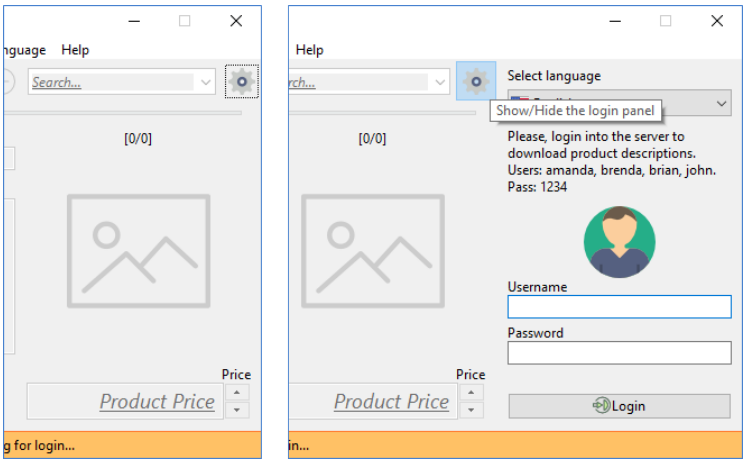


Figura 25.10: Mostrar/Ocultar el panel de login.

25.4.3. Gráficos de barras

Uno de los requisitos es que la interfaz incluya un pequeño gráfico de barras que vaya mostrando las estadísticas de venta de cada producto (Figura 25.11). El código que genera dicho gráfico lo tenemos en (Listado 25.13). En *"Uso de vistas personalizadasUso de vistas personalizadas"* (Página 398), *"Dibujo paramétricoDibujo paramétrico"* (Página 399) y *"Contextos 2D"* (Página 263) tienes más información sobre gráficos interactivos.

Listado 25.13: Dibujo paramétrico de un gráfico de barras.

```
static void i_OnStats(Ctrl *ctrl, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    uint32_t i, n = sizeof(ctrl->stats) / sizeof(real32_t);
    real32_t p = 10.f, x = p, y0 = params->height - p;
    real32_t w = (params->width - p * 2) / n;
    real32_t h = params->height - p * 2;
    real32_t avg = 0, pavg;
    char_t tavg[16];
    color_t c[2];
    real32_t stop[2] = {0, 1};
    c[0] = kHOLDER;
    c[1] = kCOLOR_VIEW;
    draw_fill_linear(params->ctx, c,stop, 2, 0, p, 0, params->height - p + 1);

    for (i = 0; i < n; ++i)
    {
```

```
        real32_t hr = h * (ctrl->stats[i] / i_MAX_STATS);
        real32_t y = p + h - hr;
        draw_rect(params->ctx, ekFILL, x, y, w - 2, hr);
        avg += ctrl->stats[i];
        x += w;
    }

    avg /= n;
    pavg = h * (avg / i_MAX_STATS);
    pavg = p + h - pavg;
    bstd_sprintf(tavg, sizeof(tavg), "%.2f", avg);
    draw_fill_color(params->ctx, kTXTRED);
    draw_line_color(params->ctx, kTXTRED);
    draw_line(params->ctx, p - 2, pavg, params->width - p, pavg);
    draw_line_color(params->ctx, kCOLOR_LABEL);
    draw_line(params->ctx, p - 2, y0 + 2, params->width - p, y0 + 2);
    draw_line(params->ctx, p - 2, y0 + 2, p - 2, p);
    draw_text(params->ctx, ekFILL, tavg, p, pavg);
}
```

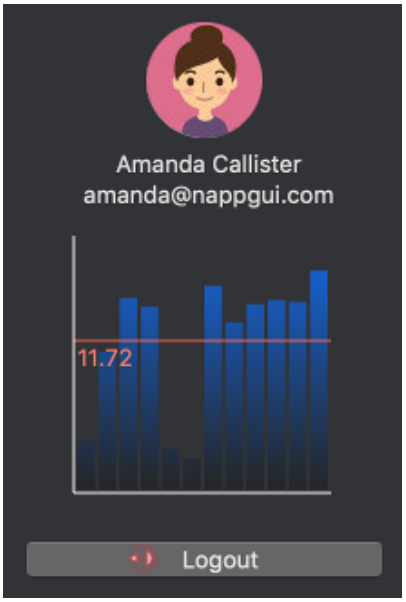


Figura 25.11: Gráficos dinámicos en el panel de login.

25.4.4. Traducciones

La interfaz se ha traducido a siete idiomas, con el Inglés como predeterminado (Figura 25.12). Para cambiar el lenguaje, llamamos a `gui_language` dentro del manejador del evento del `PopUp` (Listado 25.14). En *"Recursos"* (Página 131) tienes una guía paso a paso para localizar y traducir aplicaciones.

Listado 25.14: Código que cambia el idioma del programa.

```
static void i_OnLang(Ctrl *ctrl, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    static const char_t *LANGS[] = { "en_US", "es_ES", "pt_PT", "it_IT", "vi_VN
        ↪ ", "ru_RU", "ja_JP" };
    gui_language(LANGS[params->index]);
}
```

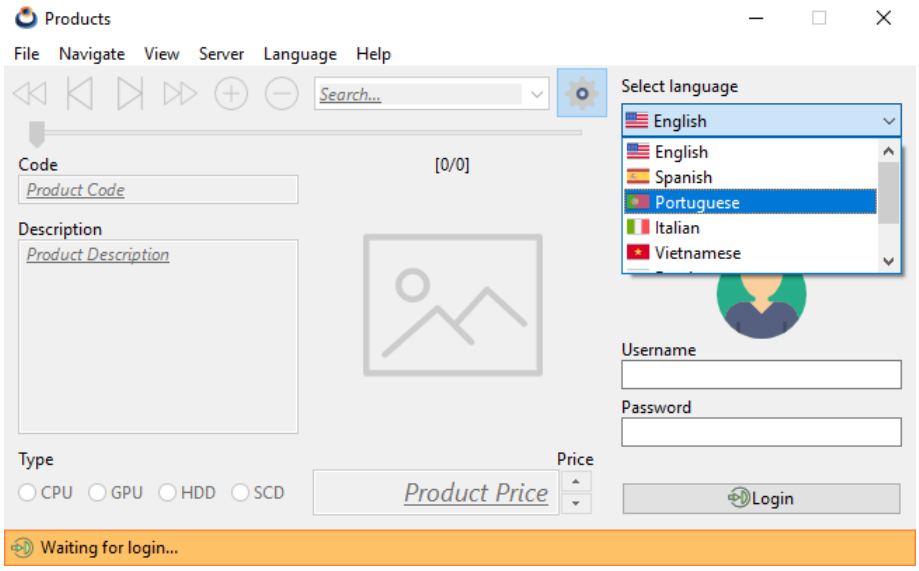


Figura 25.12: Traducciones automáticas.

25.4.5. Temas *Dark Mode*

NAppGUI utiliza controles de interfaz nativos, hecho que que provoca que las ventanas se integren perfectamente con el tema de escritorio activo en cada máquina. No obstante, si utilizamos iconos o colores personalizados es posible estos no siempre sean coherentes al portar a otros sistemas.

- En *"Gui"* (Página 303) se definen una serie de colores "de sistema", por ejemplo `gui_label_color`, cuyo valor RGB se resolverá en tiempo de ejecución en función de la plataforma destino. Utilizando estas funciones, tendremos la certeza que nuestras aplicaciones siempre lucirán bien y presentarán un esquema de colores coherentes. En *"Tabla de colores"* (Página 647) tienes una demo que muestra estos colores.
- Utiliza `gui_alt_color` para definir colores con dos versiones: Una para temas claros y otra para oscuros. NAppGUI se encargará de resolver el RGB cada vez que sea necesario (Listado 25.15).

Listado 25.15: Colores personalizados utilizados en `Products`.

```
kHOLDER = gui_alt_color(color_bgr(0x4681Cf), color_bgr(0x1569E6));
kEDITBG = gui_alt_color(color_bgr(0xFFFFe4), color_bgr(0x101010));
kSTATBG = gui_alt_color(color_bgr(0xFFC165), color_bgr(0x523d1d));
kSTATSK = gui_alt_color(color_bgr(0xFF8034), color_bgr(0xFF8034));
kTXTRED = gui_alt_color(color_bgr(0xFF0000), color_bgr(0xEB665A));
```

- Para las imágenes, deberemos incluir dos versiones en los recursos del programa y seleccionar una u otra en función del valor `gui_dark_mode` (Listado 25.16).

Listado 25.16: Selección de iconos para *Light* o *Dark Themes*.

```
void ctrl_theme_images(Ctrl *ctrl)
{
    bool_t dark = color_dark_mode();
    button_image(cell_button(ctrl->first_cell), dark ? FIRSTD_PNG :
        ↪ FIRST_PNG);
    button_image(cell_button(ctrl->back_cell), dark ? BACKD_PNG : BACK_PNG
        ↪ );
    button_image(cell_button(ctrl->next_cell), dark ? NEXTD_PNG : NEXT_PNG
        ↪ );
    button_image(cell_button(ctrl->last_cell), dark ? LASTD_PNG : LAST_PNG
        ↪ );
    button_image(cell_button(ctrl->add_cell), ADD_PNG);
    button_image(cell_button(ctrl->minus_cell), MINUS_PNG);
    button_image(cell_button(ctrl->setting_cell), SETTINGS_PNG);
    button_image(cell_button(ctrl->login_cell), LOGIN16_PNG);
    button_image(cell_button(ctrl->logout_cell), dark ? LOGOUT16D_PNG :
        ↪ LOGOUT16_PNG);
    menuitem_image(ctrl->import_item, OPEN_PNG);
    menuitem_image(ctrl->export_item, dark ? SAVED_PNG : SAVE_PNG);
    menuitem_image(ctrl->first_item, dark ? FIRST16D_PNG : FIRST16_PNG);
    menuitem_image(ctrl->back_item, dark ? BACK16D_PNG : BACK16_PNG);
    menuitem_image(ctrl->next_item, dark ? NEXT16D_PNG : NEXT16_PNG);
    menuitem_image(ctrl->last_item, dark ? LAST16D_PNG : LAST16_PNG);
    menuitem_image(ctrl->login_item, LOGIN16_PNG);
    menuitem_image(ctrl->logout_item, dark ? LOGOUT16D_PNG : LOGOUT16_PNG)
        ↪ ;
}
```

- Utiliza `gui_OnThemeChanged` para actualizar los iconos personalizados en tiempo de ejecución (Listado 25.17) (Figura 25.13).

Listado 25.17: Actualización de iconos en tiempo de ejecución.

```
static void i_OnThemeChanged(App *app, Event *e)
{
    ctrl_theme_images(app->ctrl);
    unref(e);
}
```

```
gui_OnThemeChanged(listener(app, i_OnThemeChanged, App));
```

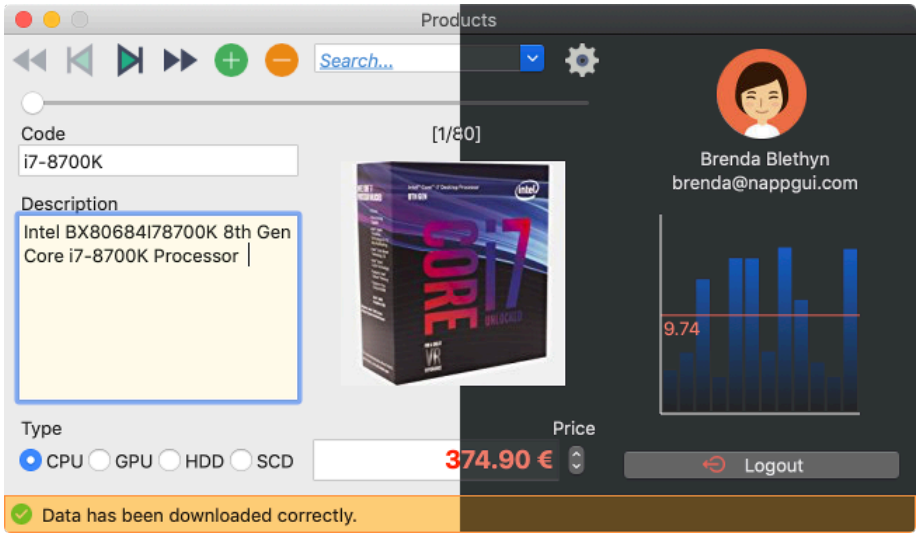


Figura 25.13: Cambio del tema de escritorio.

25.5. Controlador

El controlador es el encargado de mantener la coherencia entre el Modelo y la Vista, así como de implementar la **lógica de negocio**. Concretamente este programa no hace prácticamente nada con los datos, al margen de descargarlos del servidor y mostrarlos, lo que presenta una buena oportunidad para practicar.

25.5.1. Login multi-hilo

Cuando el usuario pulsa el botón `[Login]` el programa llama a dos servicios Web. Uno para registrar al usuario y otro para descargar los datos. Este proceso dura alrededor de un segundo, lo cual es una eternidad desde el punto de vista de un proceso. Durante este tiempo se llega a apreciar que el programa se queda "congelado" a la espera de que se resuelvan las llamadas del servidor. Esto ocurre porque se está ejecutando una tarea "lenta" en el mismo hilo que gestiona el ciclo de mensajes (Figura 25.14)**(a)**.

Para evitar este desagradable efecto, que puede agravarse si la petición durase más tiempo, vamos a utilizar *"Tareas multi-hilo Tareas multi-hilo"* (Página 378) mediante `osapp_task` (Listado 25.18) (Figura 25.14)**(b)**. Esto crea un nuevo hilo de ejecución que comienza en `i_login_begin`. En el momento que los datos se han descargado, el gestor de tareas de NAppGUI llamará a `i_login_end` (ya en el hilo principal) y el programa seguirá con su ejecución mono-hilo.

Listado 25.18: Proceso de login multi-hilo.

```
static void i_OnLogin(Ctrl *ctrl, Event *e)
{
    ctrl->status = ekIN_LOGIN;
    i_status(ctrl);
    osapp_task(ctrl, 0., i_login_begin, NULL, i_login_end, Ctrl);
    unref(e);
}
```

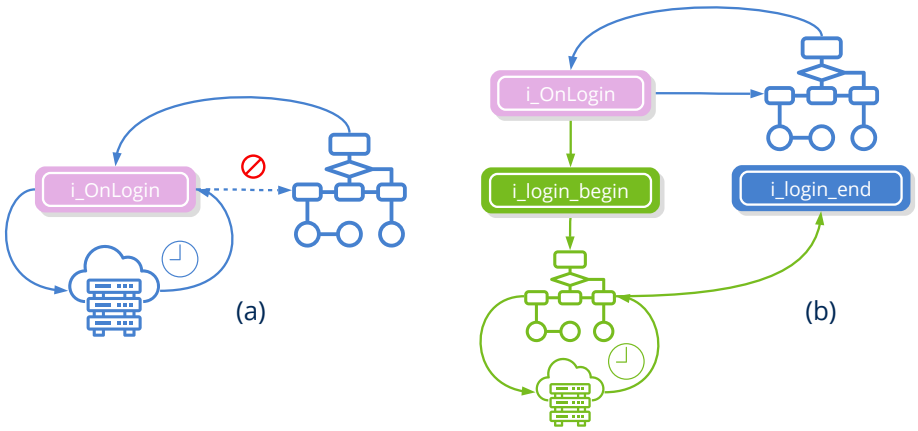


Figura 25.14: Ejecución de una tarea "lenta". Mono-hilo (a), Multi-hilo (b). Con un solo hilo la interfaz se quedará "congelada".

25.5.2. Sincronizar Modelo y Vista

Mantener sincronizados el Modelo de datos y la Vista también es tarea del controlador. A medida que el usuario interactúa con la interfaz, este deberá capturar los eventos, filtrar datos y actualizar los objetos del modelo. De igual forma, cada vez que cambie el modelo tendrá que refrescar la interfaz. Esta sincronización bidireccional se puede automatizar utilizando el registro **dbind**, ahorrando mucho código extra de programación (Figura 25.15).

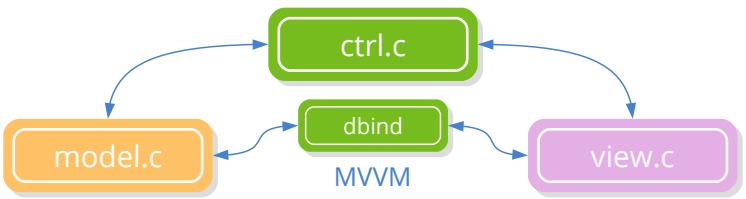


Figura 25.15: DBind libera al controlador de la recurrente tarea de sincronizar los objetos con la interfaz.

La implementación de este patrón **MVVM** *Model-View-ViewModel* es bastante sencilla

y la tenemos resumida en (Listado 25.19) (Figura 25.16).

- Utiliza `cell_dbind` para vincular una celda del layout con un campo del modelo.
- Utiliza `layout_dbind` para vincular el layout que contiene las celdas anteriores con el `struct` que contiene los campos.
- Utiliza `layout_dbind_obj` para asignar un objeto al layout. A partir de aquí las actualizaciones Modelo-Vista se realizarán de forma automática.

Listado 25.19: Vinculación de `struct` con `layout`.

```
// In View
Cell *cell0 = layout_cell(layout, 0, 1);
...
cell_dbind(cell0, Product, String*, code);
cell_dbind(cell1, Product, String*, description);
cell_dbind(cell2, Product, type_t, type);
cell_dbind(cell3, Product, Image*, image64);
cell_dbind(cell4, Product, real32_t, price);
layout_dbind(layout, Product);

// In Controller
Product *product = model_product(model, index);
layout_dbind_obj(layout, product, Product);
```

Es habitual que los datos tengan que ser revisados (filtrados) tras la edición para comprobar que los valores sean coherentes con el modelo. **dbind** admite diferentes formatos para los campos registrados. En (Listado 25.20) hemos aplicado formato al campo `price` de `Product`.

Listado 25.20: Formato del campo `price` de `Product`.

```
dbind_default(Product, real32_t, price, 1);
dbind_range(Product, real32_t, price, .50f, 1e6f);
dbind_precision(Product, real32_t, price, .05f);
dbind_increment(Product, real32_t, price, 5.f);
dbind_suffix(Product, real32_t, price, "€");
```

25.5.3. Cambiar la imagen

Para cambiar la imagen asociada al producto, el controlador ha modificado ligeramente el funcionamiento del `ImageView`, que mostrará un icono sobreimpreso cada vez que el ratón se sitúe encima de la imagen (Listado 25.21), (Figura 25.17).

Listado 25.21: Dibujo de un *overlay* cuando el ratón está encima de la imagen.

```
static void i_OnImgDraw(Ctrl *ctrl, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
```

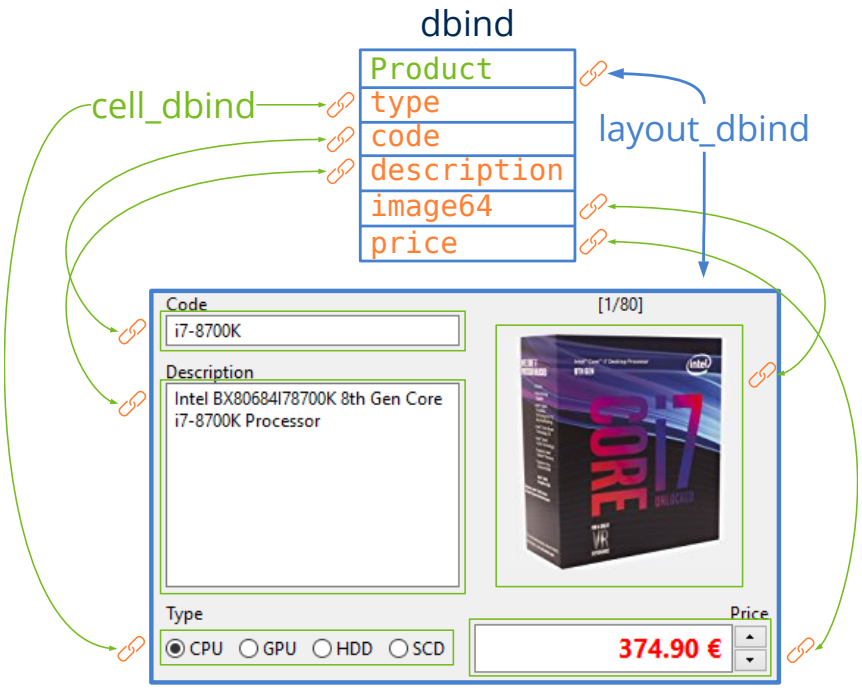


Figura 25.16: Vinculación de datos en el GUI.

```
    const Image *image = gui_respack_image(EDIT_PNG);
    uint32_t w, h;
    image_size(image, &w, &h);
    draw_image(params->context, image, params->width - w - 10, params->height -
        ↪ h - 10);
    unref(ctrl);
}
...
imageview_OnOverDraw(view, listener(ctrl, i_OnImgDraw, Ctrl));
```

Al hacer clic sobre la imagen aparecerá el diálogo de apertura de archivos que nos permitirá seleccionar una nueva. Si se acepta el diálogo, la imagen se cargará y se asignará al control (Listado 25.22). El objeto se actualizará automáticamente.

Listado 25.22: Dibujo de un *overlay* cuando el ratón está encima de la imagen.

```
static void i_OnImgClick(Ctrl *ctrl, Event *e)
{
    const char_t *type[] = { "png", "jpg" };
    const char_t *file = comwin_open_file(type, 2, NULL);
    if (file != NULL)
    {
        Image *image = image_from_file(file, NULL);
        if (image != NULL)
```

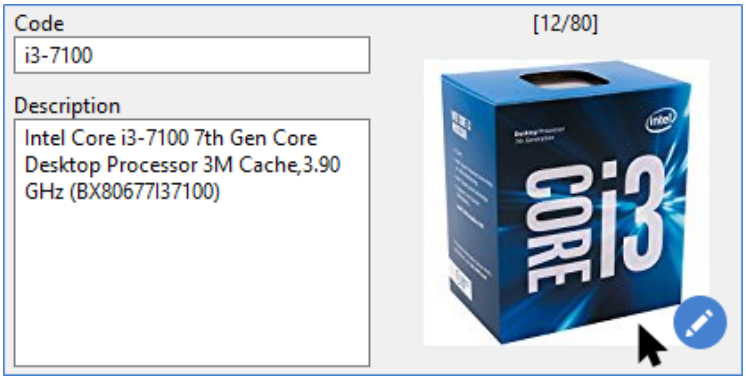


Figura 25.17: Icono sobreimpreso en una vista de imagen.

```
        {
            View *view = cell_view(ctrl->image_cell);
            imageview_image(view, image);
            image_destroy(&image);
        }
    }
    unref(e);
}
...
imageview_OnClick(view, listener(ctrl, i_OnImgClick, Ctrl));
```

25.5.4. Gestión de memoria

Tras el cierre del programa se imprimirá un informe con el uso de la memoria, alertándonos de posibles *memory leaks* (Listado 25.23). No está de más revisarlo periódicamente con el fin de detectar anomalías lo antes posible.

Listado 25.23: Estadísticas del uso de memoria, generadas al cierre de cualquier aplicación NAppGUI.

```
[22:17:21] [OK] Heap Memory Staticstics
[22:17:21] ============================
[22:17:21] Total a/dellocations: 2065, 2065
[22:17:21] Total bytes a/dellocated: 2831766, 2831766
[22:17:21] Max bytes allocated: 1642879
[22:17:21] Effective reallocations: (0/55)
[22:17:21] Real allocations: 13 pages of 65536 bytes
[22:17:21]                   5 pages greater than 65536 bytes
[22:17:21] ============================
```

Si queremos información más detallada, podemos pasar el parámetro `"-hv"` en el campo `options` de `osmain` (Listado 25.24).

```
osmain(i_create, i_destroy, "-hv", App)
```

Listado 25.24: Salida detallada del uso de memoria.

```
[12:01:41] 'App' a/deallocations: 1, 1 (32) bytes
[12:01:41] 'ArrPt::Cell' a/deallocations: 24, 24 (576) bytes
[12:01:41] 'ArrPt::GuiComponent' a/deallocations: 8, 8 (192) bytes
...
[12:01:41] 'Button' a/deallocations: 13, 13 (1664) bytes
[12:01:41] 'View' a/deallocations: 5, 5 (840) bytes
[12:01:41] 'Clock' a/deallocations: 1, 1 (48) bytes
[12:01:41] 'Combo' a/deallocations: 1, 1 (176) bytes
...
[12:01:41] 'UpDown' a/deallocations: 1, 1 (64) bytes
[12:01:41] 'VImgData' a/deallocations: 4, 4 (160) bytes
[12:01:41] 'Window' a/deallocations: 1, 1 (80) bytes
[12:01:41] 'bool_t::arr' a/deallocations: 6, 6 (27) bytes
[12:01:41] 'i_App' a/deallocations: 1, 1 (184) bytes
[12:01:41] 'i_Task' a/deallocations: 1, 1 (64) bytes
```

25.6. El programa completo

Listado 25.25: demo/products/products.hxx

```
/* Products Types */

#ifndef __TYPES_HXX__
#define __TYPES_HXX__

#include <gui/gui.hxx>

typedef enum _wserv_t
{
    ekWS_CONNECT = 1,
    ekWS_JSON,
    ekWS_ACCESS,
    ekWS_OK
} wserv_t;

typedef struct _model_t Model;
typedef struct _product_t Product;
typedef struct _ctrl_t Ctrl;

__EXTERN_C

extern color_t kHOLDER;
extern color_t kEDITBG;
extern color_t kSTATBG;
extern color_t kSTATSK;
extern color_t kTXTRED;

__END_C
```

```
#endif
```

Listado 25.26: demo/products/products.c

```
/* NAppGUI Products Demo */

#include "nappgui.h"
#include "prmodel.h"
#include "prmenu.h"
#include "prctrl.h"
#include "prview.h"
#include "res_products.h"
#include <inet/inet.h>

typedef struct _app_t App;
struct _app_t
{
    Model *model;
    Ctrl *ctrl;
    Window *window;
    Menu *menu;
};

color_t kHOLDER;
color_t kEDITBG;
color_t kSTATBG;
color_t kSTATSK;
color_t kTXTRED;

/*---------------------------------------------------------------------------*/

static void i_OnThemeChanged(App *app, Event *e)
{
    ctrl_theme_images(app->ctrl);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new(App);
    kHOLDER = gui_alt_color(color_bgr(0x4681Cf), color_bgr(0x1569E6));
    kEDITBG = gui_alt_color(color_bgr(0xFFFFe4), color_bgr(0x101010));
    kSTATBG = gui_alt_color(color_bgr(0xFFC165), color_bgr(0x523d1d));
    kSTATSK = gui_alt_color(color_bgr(0xFF8034), color_bgr(0xFF8034));
    kTXTRED = gui_alt_color(color_bgr(0xFF0000), color_bgr(0xEB665A));
    inet_start();
    gui_respack(res_products_respack);
    gui_language("");
```

```
    gui_OnThemeChanged(listener(app, i_OnThemeChanged, App));
    model_bind();
    app->model = model_create();
    app->ctrl = ctrl_create(app->model);
    app->menu = prmenu_create(app->ctrl);
    app->window = prview_create(app->ctrl);
    osapp_menubar(app->menu, app->window);
    ctrl_run(app->ctrl);
    window_origin(app->window, v2df(100.f, 100.f));
    window_show(app->window);
    return app;
}

/*-------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    cassert_no_null(app);
    cassert_no_null(*app);
    ctrl_destroy(&(*app)->ctrl);
    window_destroy(&(*app)->window);
    menu_destroy(&(*app)->menu);
    model_destroy(&(*app)->model);
    inet_finish();
    heap_delete(app, App);
}

/*-------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Listado 25.27: demo/products/prmodel.c

```
/* Products Model */

#include "prmodel.h"
#include "res_products.h"
#include <gui/guiall.h>
#include <inet/httpreq.h>
#include <inet/json.h>

typedef struct _pjson_t PJson;

typedef enum _type_t
{
    ekCPU,
    ekGPU,
    ekHDD,
    ekSCD
} type_t;
```

```
struct _product_t
{
    type_t type;
    String *code;
    String *description;
    Image *image64;
    real32_t price;
};

struct _pjson_t
{
    int32_t code;
    uint32_t size;
    ArrPt(Product) *data;
};

struct _model_t
{
    ArrSt(uint32_t) *filter;
    ArrPt(Product) *products;
};

DeclPt(Product);

/*---------------------------------------------------------------------------*/

Model *model_create(void)
{
    Model *model = heap_new(Model);
    model->filter = arrst_create(uint32_t);
    model->products = arrpt_create(Product);
    return model;
}

/*---------------------------------------------------------------------------*/

void model_destroy(Model **model)
{
    arrst_destroy(&(*model)->filter, NULL, uint32_t);
    dbind_destroy(&(*model)->products, ArrPt(Product));
    heap_delete(model, Model);
}

/*---------------------------------------------------------------------------*/

static Stream *i_http_get(void)
{
    Http *http = http_create("serv.nappgui.com", 80);
    Stream *stm = NULL;
```

```
    if (http_get(http, "/dproducts.php", NULL, 0, NULL) == TRUE)
    {
        uint32_t status = http_response_status(http);
        if (status >= 200 && status <= 299)
        {
            stm = stm_memory(4096);
            if (http_response_body(http, stm, NULL) == FALSE)
                stm_close(&stm);
        }
    }

    http_destroy(&http);
    return stm;
}

/*-------------------------------------------------------------------------*/

wserv_t model_webserv(Model *model)
{
    Stream *stm = i_http_get();
    if (stm != NULL)
    {
        PJson *json = json_read(stm, NULL, PJson);
        stm_close(&stm);

        if (json != NULL)
        {
            cassert(json->size == arrpt_size(json->data, Product));
            dbind_destroy(&model->products, ArrPt(Product));
            model->products = json->data;
            json->data = NULL;
            json_destroy(&json, PJson);
            return ekWS_OK;
        }

        return ekWS_JSON;
    }

    return ekWS_CONNECT;
}

/*-------------------------------------------------------------------------*/

bool_t model_import(Model *model, const char_t *pathname, ferror_t *err)
{
    Stream *stm = stm_from_file(pathname, err);
    if (stm != NULL)
    {
        ArrPt(Product) *products = dbind_read(stm, ArrPt(Product));
        stm_close(&stm);
```

```
        if (products != NULL)
        {
            dbind_destroy(&model->products, ArrPt(Product));
            model->products = products;
            return TRUE;
        }
    }

    return FALSE;
}

/*-----------------------------------------------------------------------------*/

bool_t model_export(Model *model, const char_t *pathname, ferror_t *err)
{
    Stream *stm = stm_to_file(pathname, err);
    if (stm != NULL)
    {
        dbind_write(stm, model->products, ArrPt(Product));
        stm_close(&stm);
        return TRUE;
    }

    return FALSE;
}

/*-----------------------------------------------------------------------------*/

uint32_t model_count(const Model *model)
{
    uint32_t total = arrst_size(model->filter, uint32_t);
    if (total == 0)
        total = arrpt_size(model->products, Product);
    return total;
}

/*-----------------------------------------------------------------------------*/

void model_clear(Model *model)
{
    dbind_destroy(&model->products, ArrPt(Product));
    arrst_clear(model->filter, NULL, uint32_t);
    model->products = dbind_create(ArrPt(Product));
}

/*-----------------------------------------------------------------------------*/

void model_add(Model *model)
{
    Product *product = dbind_create(Product);
    arrpt_append(model->products, product, Product);
```

```
    arrst_clear(model->filter, NULL, uint32_t);
}

/*----------------------------------------------------------------------------*/

static uint32_t i_index(ArrSt(uint32_t) *filter, const uint32_t index)
{
    if (arrst_size(filter, uint32_t) > 0)
        return *arrst_get(filter, index, uint32_t);
    else
        return index;
}

/*----------------------------------------------------------------------------*/

static __INLINE void i_destroy(Product **product)
{
    dbind_destroy(product, Product);
}

/*----------------------------------------------------------------------------*/

void model_delete(Model *model, const uint32_t index)
{
    uint32_t lindex = i_index(model->filter, index);
    arrpt_delete(model->products, lindex, i_destroy, Product);
    arrst_clear(model->filter, NULL, uint32_t);
}

/*----------------------------------------------------------------------------*/

bool_t model_filter(Model *model, const char_t *filter)
{
    ArrSt(uint32_t) *new_filter = arrst_create(uint32_t);

    arrpt_foreach(product, model->products, Product)
        if (str_str(tc(product->description), filter) != NULL)
            arrst_append(new_filter, product_i, uint32_t);
    arrpt_end();

    arrst_destroy(&model->filter, NULL, uint32_t);
    model->filter = new_filter;

    return (bool_t)(arrst_size(new_filter, uint32_t) > 0);
}

/*----------------------------------------------------------------------------*/

Product *model_product(Model *model, const uint32_t product_id)
{
    uint32_t lindex = i_index(model->filter, product_id);
```

```
    return arrpt_get(model->products, lindex, Product);
}

/*---------------------------------------------------------------------------*/

void model_bind(void)
{
    dbind_enum(type_t, ekCPU, "");
    dbind_enum(type_t, ekGPU, "");
    dbind_enum(type_t, ekHDD, "");
    dbind_enum(type_t, ekSCD, "");
    dbind(Product, type_t, type);
    dbind(Product, String*, code);
    dbind(Product, String*, description);
    dbind(Product, Image*, image64);
    dbind(Product, real32_t, price);
    dbind(PJson, int32_t, code);
    dbind(PJson, uint32_t, size);
    dbind(PJson, ArrPt(Product)*, data);
    dbind_default(Product, real32_t, price, 1);
    dbind_range(Product, real32_t, price, .50f, 1e6f);
    dbind_precision(Product, real32_t, price, .05f);
    dbind_increment(Product, real32_t, price, 5.f);
    dbind_suffix(Product, real32_t, price, "€");
    dbind_default(Product, Image*, image64, gui_image(NOIMAGE_PNG));
}

/*---------------------------------------------------------------------------*/

void model_layout(Layout *layout)
{
    layout_dbind(layout, NULL, Product);
}

/*---------------------------------------------------------------------------*/

void model_type(Cell *cell)
{
    cell_dbind(cell, Product, type_t, type);
}

/*---------------------------------------------------------------------------*/

void model_code(Cell *cell)
{
    cell_dbind(cell, Product, String*, code);
}

/*---------------------------------------------------------------------------*/

void model_desc(Cell *cell)
```

```
{
    cell_dbind(cell, Product, String*, description);
}

/*-------------------------------------------------------------------------*/

void model_image(Cell *cell)
{
    cell_dbind(cell, Product, Image*, image64);
}

/*-------------------------------------------------------------------------*/

void model_price(Cell *cell)
{
    cell_dbind(cell, Product, real32_t, price);
}
```

Listado 25.28: demo/products/prview.c

```
/* Products View */

#include "prview.h"
#include "prctrl.h"
#include "res_products.h"
#include <gui/guiall.h>

/*-------------------------------------------------------------------------*/

static Layout *i_toolbar(Ctrl *ctrl)
{
    Layout *layout = layout_create(8, 1);
    Button *button0 = button_flat();
    Button *button1 = button_flat();
    Button *button2 = button_flat();
    Button *button3 = button_flat();
    Button *button4 = button_flat();
    Button *button5 = button_flat();
    Button *button6 = button_flatgle();
    Combo *combo = combo_create();
    button_text(button0, TWIN_FIRST);
    button_text(button1, TWIN_BACK);
    button_text(button2, TWIN_NEXT);
    button_text(button3, TWIN_LAST);
    button_text(button4, TWIN_ADD);
    button_text(button5, TWIN_DEL);
    button_text(button6, TWIN_SETTINGS_PANEL);
    combo_tooltip(combo, TWIN_FILTER_DESC);
    combo_bgcolor_focus(combo, kEDITBG);
    combo_phtext(combo, TWIN_FILTER);
    combo_phcolor(combo, kHOLDER);
```

```
    combo_phstyle(combo, ekFITALIC | ekFUNDERLINE);
    layout_button(layout, button0, 0, 0);
    layout_button(layout, button1, 1, 0);
    layout_button(layout, button2, 2, 0);
    layout_button(layout, button3, 3, 0);
    layout_button(layout, button4, 4, 0);
    layout_button(layout, button5, 5, 0);
    layout_combo(layout, combo, 6, 0);
    layout_button(layout, button6, 7, 0);
    layout_hmargin(layout, 5, 5);
    layout_hmargin(layout, 6, 5);
    layout_hexpand(layout, 6);
    ctrl_first_cell(ctrl, layout_cell(layout, 0, 0));
    ctrl_back_cell(ctrl, layout_cell(layout, 1, 0));
    ctrl_next_cell(ctrl, layout_cell(layout, 2, 0));
    ctrl_last_cell(ctrl, layout_cell(layout, 3, 0));
    ctrl_add_cell(ctrl, layout_cell(layout, 4, 0));
    ctrl_minus_cell(ctrl, layout_cell(layout, 5, 0));
    ctrl_filter_cell(ctrl, layout_cell(layout, 6, 0));
    ctrl_setting_cell(ctrl, layout_cell(layout, 7, 0));
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_code_desc(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 4);
    Label *label0 = label_create();
    Label *label1 = label_create();
    Edit *edit0 = edit_create();
    Edit *edit1 = edit_multiline();
    label_text(label0, TWIN_CODE);
    label_text(label1, TWIN_DESC);
    edit_phtext(edit0, TWIN_TYPE_CODE);
    edit_phtext(edit1, TWIN_TYPE_DESC);
    edit_bgcolor_focus(edit0, kEDITBG);
    edit_bgcolor_focus(edit1, kEDITBG);
    edit_phcolor(edit0, kHOLDER);
    edit_phcolor(edit1, kHOLDER);
    edit_phstyle(edit0, ekFITALIC | ekFUNDERLINE);
    edit_phstyle(edit1, ekFITALIC | ekFUNDERLINE);
    layout_label(layout, label0, 0, 0);
    layout_edit(layout, edit0, 0, 1);
    layout_label(layout, label1, 0, 2);
    layout_edit(layout, edit1, 0, 3);
    layout_vmargin(layout, 1, 10);
    layout_vexpand(layout, 3);
    ctrl_code_cell(ctrl, layout_cell(layout, 0, 1));
    ctrl_desc_cell(ctrl, layout_cell(layout, 0, 3));
    return layout;
```

```
}

/*-------------------------------------------------------------------------*/

static Layout *i_type(void)
{
    Layout *layout = layout_create(4, 1);
    Button *button0 = button_radio();
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    Button *button3 = button_radio();
    button_text(button0, TWIN_CPU);
    button_text(button1, TWIN_GPU);
    button_text(button2, TWIN_HDD);
    button_text(button3, TWIN_SCD);
    layout_button(layout, button0, 0, 0);
    layout_button(layout, button1, 1, 0);
    layout_button(layout, button2, 2, 0);
    layout_button(layout, button3, 3, 0);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_n_img(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 2);
    Label *label = label_create();
    ImageView *view = imageview_create();
    label_align(label, ekCENTER);
    layout_halign(layout, 0, 0, ekJUSTIFY);
    layout_label(layout, label, 0, 0);
    layout_imageview(layout, view, 0, 1);
    layout_vexpand(layout, 1);
    ctrl_counter_cell(ctrl, layout_cell(layout, 0, 0));
    ctrl_image_cell(ctrl, layout_cell(layout, 0, 1));
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_price(void)
{
    Layout *layout = layout_create(2, 1);
    Edit *edit = edit_create();
    Font *font = font_system(18, ekFBOLD);
    UpDown *updown = updown_create();
    edit_phtext(edit, TWIN_TYPE_PRICE);
    edit_font(edit, font);
    edit_align(edit, ekRIGHT);
    edit_color(edit, kTXTRED);
```

```
    edit_bgcolor_focus(edit, kEDITBG);
    edit_phcolor(edit, kHOLDER);
    edit_phstyle(edit, ekFITALIC | ekFUNDERLINE);
    layout_edit(layout, edit, 0, 0);
    layout_updown(layout, updown, 1, 0);
    layout_hsize(layout, 1, 24);
    layout_hexpand(layout, 0);
    font_destroy(&font);
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_product(Ctrl *ctrl)
{
    Layout *layout = layout_create(2, 3);
    Layout *layout0 = i_code_desc(ctrl);
    Layout *layout1 = i_type();
    Layout *layout2 = i_n_img(ctrl);
    Layout *layout3 = i_price();
    Label *label0 = label_create();
    Label *label1 = label_create();
    label_text(label0, TWIN_TYPE);
    label_text(label1, TWIN_PRICE);
    layout_layout(layout, layout0, 0, 0);
    layout_label(layout, label0, 0, 1);
    layout_layout(layout, layout1, 0, 2);
    layout_layout(layout, layout2, 1, 0);
    layout_label(layout, label1, 1, 1);
    layout_layout(layout, layout3, 1, 2);
    layout_halign(layout, 1, 1, ekRIGHT);
    layout_hsize(layout, 1, 200);
    layout_vsize(layout, 0, 200);
    layout_hmargin(layout, 0, 10);
    layout_vmargin(layout, 0, 10);
    layout_margin4(layout, 0, 10, 10, 10);
    layout_vexpand(layout, 0);
    ctrl_type_cell(ctrl, layout_cell(layout, 0, 2));
    ctrl_price_cell(ctrl, layout_cell(layout, 1, 2));
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_form(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 3);
    Layout *layout0 = i_toolbar(ctrl);
    Layout *layout1 = i_product(ctrl);
    Slider *slider = slider_create();
    Cell *cell = NULL;
```

```
    layout_layout(layout, layout0, 0, 0);
    layout_slider(layout, slider, 0, 1);
    layout_layout(layout, layout1, 0, 2);
    layout_vexpand(layout, 2);
    cell = layout_cell(layout, 0, 1);
    cell_padding4(cell, 0, 10, 0, 10);
    ctrl_slider_cell(ctrl, cell);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_login(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 10);
    Label *label0 = label_create();
    Label *label1 = label_multiline();
    Label *label2 = label_create();
    Label *label3 = label_create();
    PopUp *popup0 = popup_create();
    ImageView *view0 = imageview_create();
    Edit *edit0 = edit_create();
    Edit *edit1 = edit_create();
    Button *button = button_push();
    label_text(label0, TWIN_SETLANG);
    label_text(label1, TWIN_LOGIN_MSG);
    label_text(label2, TWIN_USER);
    label_text(label3, TWIN_PASS);
    popup_add_elem(popup0, ENGLISH, (const Image*)USA_PNG);
    popup_add_elem(popup0, SPANISH, (const Image*)SPAIN_PNG);
    popup_add_elem(popup0, PORTUGUESE, (const Image*)PORTUGAL_PNG);
    popup_add_elem(popup0, ITALIAN, (const Image*)ITALY_PNG);
    popup_add_elem(popup0, VIETNAMESE, (const Image*)VIETNAM_PNG);
    popup_add_elem(popup0, RUSSIAN, (const Image*)RUSSIA_PNG);
    popup_add_elem(popup0, JAPANESE, (const Image*)JAPAN_PNG);
    popup_tooltip(popup0, TWIN_SETLANG);
    imageview_image(view0, (const Image*)USER_PNG);
    edit_passmode(edit1, TRUE);
    button_text(button, TWIN_LOGIN);
    layout_label(layout, label0, 0, 0);
    layout_popup(layout, popup0, 0, 1);
    layout_label(layout, label1, 0, 2);
    layout_imageview(layout, view0, 0, 3);
    layout_label(layout, label2, 0, 4);
    layout_edit(layout, edit0, 0, 5);
    layout_label(layout, label3, 0, 6);
    layout_edit(layout, edit1, 0, 7);
    layout_button(layout, button, 0, 9);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 10);
    layout_vmargin(layout, 2, 10);
```

```
    layout_vmargin(layout, 5, 5);
    layout_vmargin(layout, 8, 5);
    layout_margin4(layout, 5, 10, 10, 10);
    layout_hsize(layout, 0, 200);
    layout_vexpand(layout, 8);
    ctrl_lang_cell(ctrl, layout_cell(layout, 0, 1));
    ctrl_user_cell(ctrl, layout_cell(layout, 0, 5));
    ctrl_pass_cell(ctrl, layout_cell(layout, 0, 7));
    ctrl_login_cell(ctrl, layout_cell(layout, 0, 9));
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_logout(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 6);
    ImageView *view = imageview_create();
    Label *label0 = label_create();
    Label *label1 = label_create();
    View *cview = view_create();
    Button *button = button_push();
    label_align(label0, ekCENTER);
    label_align(label1, ekCENTER);
    button_text(button, TWIN_LOGOUT);
    view_size(cview, s2df(160, 160));
    layout_imageview(layout, view, 0, 0);
    layout_label(layout, label0, 0, 1);
    layout_label(layout, label1, 0, 2);
    layout_view(layout, cview, 0, 3);
    layout_button(layout, button, 0, 5);
    layout_halign(layout, 0, 1, ekJUSTIFY);
    layout_halign(layout, 0, 2, ekJUSTIFY);
    layout_halign(layout, 0, 3, ekCENTER);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 2, 5);
    layout_vexpand(layout, 4);
    layout_hsize(layout, 0, 200);
    layout_margin(layout, 10);
    ctrl_stats_cell(ctrl, layout_cell(layout, 0, 3));
    ctrl_logout_cell(ctrl, layout_cell(layout, 0, 5));
    return layout;
}

/*-------------------------------------------------------------------------*/

static Panel *i_login_panel(Ctrl *ctrl)
{
    Panel *panel = panel_create();
    Layout *layout0 = i_login(ctrl);
    Layout *layout1 = i_logout(ctrl);
```

```
    panel_layout(panel, layout0);
    panel_layout(panel, layout1);
    ctrl_login_panel(ctrl, panel);
    return panel;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_status_bar(Ctrl *ctrl)
{
    Layout *layout = layout_create(2, 1);
    ImageView *view = imageview_create();
    Label *label = label_create();
    imageview_size(view, s2df(16, 16));
    layout_imageview(layout, view, 0, 0);
    layout_label(layout, label, 1, 0);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_hexpand(layout, 1);
    layout_hmargin(layout, 0, 5);
    layout_margin(layout, 5);
    layout_bgcolor(layout, kSTATBG);
    layout_skcolor(layout, kSTATSK);
    ctrl_status_layout(ctrl, layout);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_layout(Ctrl *ctrl)
{
    Layout *layout = layout_create(1, 2);
    Layout *layout0 = layout_create(2, 1);
    Layout *layout1 = i_form(ctrl);
    Layout *layout2 = i_status_bar(ctrl);
    Panel *panel1 = i_login_panel(ctrl);
    layout_layout(layout0, layout1, 0, 0);
    layout_panel(layout0, panel1, 1, 0);
    layout_layout(layout, layout0, 0, 0);
    layout_layout(layout, layout2, 0, 1);
    ctrl_main_layout(ctrl, layout0);
    return layout;
}

/*-----------------------------------------------------------------------------*/

Window *prview_create(Ctrl *ctrl)
{
    Panel *panel = panel_create();
    Layout *layout = i_layout(ctrl);
    Window *window = NULL;
    ctrl_theme_images(ctrl);
```

```
    panel_layout(panel, layout);
    window = window_create(ekWINDOW_STD);
    window_panel(window, panel);
    window_title(window, TWIN_TITLE);
    ctrl_window(ctrl, window);
    return window;
}
```

Listado 25.29: demo/products/prmenu.c

```
/* Products Menu */

#include "prmenu.h"
#include "prctrl.h"
#include "res_products.h"
#include <gui/guiall.h>

/*-------------------------------------------------------------------------*/

#if defined (__APPLE__)
static Menu *i_app(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    MenuItem *item1 = menuitem_separator();
    MenuItem *item2 = menuitem_create();
    MenuItem *item3 = menuitem_separator();
    MenuItem *item4 = menuitem_create();
    menuitem_text(item0, TMEN_ABOUT);
    menuitem_text(item2, TMEN_PREFERS);
    menuitem_text(item4, TMEN_QUIT);
    menu_item(menu, item0);
    menu_item(menu, item1);
    menu_item(menu, item2);
    menu_item(menu, item3);
    menu_item(menu, item4);
    ctrl_about_item(ctrl, item0);
    ctrl_exit_item(ctrl, item4);
    return menu;
}
#endif

/*-------------------------------------------------------------------------*/

static Menu *i_file(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    MenuItem *item1 = menuitem_create();
    menuitem_text(item0, TMEN_IMPORT);
    menuitem_text(item1, TMEN_EXPORT);
```

```
    menu_item(menu, item0);
    menu_item(menu, item1);

#if !defined(__APPLE__)
    {
        MenuItem *item2 = menuitem_separator();
        MenuItem *item3 = menuitem_create();
        menuitem_text(item3, TMEN_EXIT);
        menuitem_image(item3, (const Image*)EXIT_PNG);
        menu_item(menu, item2);
        menu_item(menu, item3);
        ctrl_exit_item(ctrl, item3);
    }
#endif

    ctrl_import_item(ctrl, item0);
    ctrl_export_item(ctrl, item1);
    return menu;
}

/*-------------------------------------------------------------------------*/

static Menu *i_navigate(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    MenuItem *item1 = menuitem_create();
    MenuItem *item2 = menuitem_create();
    MenuItem *item3 = menuitem_create();
    menuitem_text(item0, TMEN_FIRST);
    menuitem_text(item1, TMEN_BACK);
    menuitem_text(item2, TMEN_NEXT);
    menuitem_text(item3, TMEN_LAST);
    menuitem_key(item0, ekKEY_F5, 0);
    menuitem_key(item1, ekKEY_F6, 0);
    menuitem_key(item2, ekKEY_F7, 0);
    menuitem_key(item3, ekKEY_F8, 0);
    menu_item(menu, item0);
    menu_item(menu, item1);
    menu_item(menu, item2);
    menu_item(menu, item3);
    ctrl_first_item(ctrl, item0);
    ctrl_back_item(ctrl, item1);
    ctrl_next_item(ctrl, item2);
    ctrl_last_item(ctrl, item3);
    return menu;
}

/*-------------------------------------------------------------------------*/

static Menu *i_view(Ctrl *ctrl)
```

```
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    unref(ctrl);
    menuitem_text(item0, TMEN_LOGIN_PANEL);
    menuitem_image(item0, (const Image*)SETTINGS16_PNG);
    menu_item(menu, item0);
    ctrl_setting_item(ctrl, item0);
    return menu;
}

/*-------------------------------------------------------------------------*/

static Menu *i_server(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    MenuItem *item1 = menuitem_create();
    menuitem_text(item0, TMEN_LOGIN);
    menuitem_text(item1, TMEN_LOGOUT);
    menu_item(menu, item0);
    menu_item(menu, item1);
    ctrl_login_item(ctrl, item0);
    ctrl_logout_item(ctrl, item1);
    return menu;
}

/*-------------------------------------------------------------------------*/

static Menu *i_language(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    MenuItem *item1 = menuitem_create();
    MenuItem *item2 = menuitem_create();
    MenuItem *item3 = menuitem_create();
    MenuItem *item4 = menuitem_create();
    MenuItem *item5 = menuitem_create();
    MenuItem *item6 = menuitem_create();
    menuitem_text(item0, ENGLISH);
    menuitem_text(item1, SPANISH);
    menuitem_text(item2, PORTUGUESE);
    menuitem_text(item3, ITALIAN);
    menuitem_text(item4, VIETNAMESE);
    menuitem_text(item5, RUSSIAN);
    menuitem_text(item6, JAPANESE);
    menuitem_image(item0, (const Image*)USA_PNG);
    menuitem_image(item1, (const Image*)SPAIN_PNG);
    menuitem_image(item2, (const Image*)PORTUGAL_PNG);
    menuitem_image(item3, (const Image*)ITALY_PNG);
    menuitem_image(item4, (const Image*)VIETNAM_PNG);
```

```
    menuitem_image(item5, (const Image*)RUSSIA_PNG);
    menuitem_image(item6, (const Image*)JAPAN_PNG);
    menu_item(menu, item0);
    menu_item(menu, item1);
    menu_item(menu, item2);
    menu_item(menu, item3);
    menu_item(menu, item4);
    menu_item(menu, item5);
    menu_item(menu, item6);
    ctrl_lang_menu(ctrl, menu);
    return menu;
}

/*-----------------------------------------------------------------------------*/

#if !defined (__APPLE__)
static Menu *i_help(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item0 = menuitem_create();
    menuitem_text(item0, TMEN_ABOUT);
    menuitem_image(item0, (const Image*)ABOUT_PNG);
    menu_item(menu, item0);
    ctrl_about_item(ctrl, item0);
    return menu;
}
#endif

/*-----------------------------------------------------------------------------*/

Menu *prmenu_create(Ctrl *ctrl)
{
    Menu *menu = menu_create();
    MenuItem *item1 = menuitem_create();
    MenuItem *item2 = menuitem_create();
    MenuItem *item3 = menuitem_create();
    MenuItem *item4 = menuitem_create();
    MenuItem *item5 = menuitem_create();
    Menu *submenu1 = i_file(ctrl);
    Menu *submenu2 = i_navigate(ctrl);
    Menu *submenu3 = i_view(ctrl);
    Menu *submenu4 = i_server(ctrl);
    Menu *submenu5 = i_language(ctrl);

 #if defined (__APPLE__)
    {
        MenuItem *item0 = menuitem_create();
        Menu *submenu0 = i_app(ctrl);
        menuitem_text(item1, "");
        menuitem_submenu(item0, &submenu0);
        menu_item(menu, item0);
```

```
    }
#endif

    menuitem_text(item1, TMEN_FILE);
    menuitem_text(item2, TMEN_NAVIGATE);
    menuitem_text(item3, TMEN_VIEW);
    menuitem_text(item4, TMEN_SERVER);
    menuitem_text(item5, LANGUAGE);
    menuitem_submenu(item1, &submenu1);
    menuitem_submenu(item2, &submenu2);
    menuitem_submenu(item3, &submenu3);
    menuitem_submenu(item4, &submenu4);
    menuitem_submenu(item5, &submenu5);
    menu_item(menu, item1);
    menu_item(menu, item2);
    menu_item(menu, item3);
    menu_item(menu, item4);
    menu_item(menu, item5);

 #if !defined (__APPLE__)
    {
        MenuItem *item6 = menuitem_create();
        Menu *submenu6 = i_help(ctrl);
        menuitem_text(item6, TMEN_HELP);
        menuitem_submenu(item6, &submenu6);
        menu_item(menu, item6);
    }
#endif
    return menu;
}
```

Listado 25.30: demo/products/prctrl.c

```
/* Products Controller */

#include "prctrl.h"
#include "prmodel.h"
#include "res_products.h"
#include <nappgui.h>
#include <inet/httpreq.h>
#include <inet/json.h>

typedef enum _status_t
{
    ekWAIT_LOGIN,
    ekIN_LOGIN,
    ekERR_LOGIN,
    ekOK_LOGIN
} status_t;

typedef struct _user_t User;
```

```
typedef struct _ujson_t UJson;

struct _user_t
{
    String *name;
    String *mail;
    Image *image64;
};

struct _ujson_t
{
    int32_t code;
    User data;
};

struct _ctrl_t
{
    Model *model;
    status_t status;
    wserv_t err;
    uint32_t selected;
    real32_t stats[12];
    UJson *ujson;
    Window *window;
    Layout *main_layout;
    Layout *status_layout;
    Cell *image_cell;
    Cell *first_cell;
    Cell *back_cell;
    Cell *next_cell;
    Cell *last_cell;
    Cell *add_cell;
    Cell *minus_cell;
    Cell *filter_cell;
    Cell *slider_cell;
    Cell *counter_cell;
    Cell *code_cell;
    Cell *desc_cell;
    Cell *price_cell;
    Cell *lang_cell;
    Cell *setting_cell;
    Cell *user_cell;
    Cell *pass_cell;
    Cell *login_cell;
    Cell *logout_cell;
    Cell *stats_cell;
    Panel *login_panel;
    Menu *lang_menu;
    MenuItem *import_item;
    MenuItem *export_item;
    MenuItem *first_item;
```

```
    MenuItem *back_item;
    MenuItem *next_item;
    MenuItem *last_item;
    MenuItem *setting_item;
    MenuItem *login_item;
    MenuItem *logout_item;
};

/*---------------------------------------------------------------------------*/

static real32_t i_MAX_STATS = 20.f;

/*---------------------------------------------------------------------------*/

Ctrl *ctrl_create(Model *model)
{
    Ctrl *ctrl = heap_new0(Ctrl);
    ctrl->model = model;
    ctrl->status = ekWAIT_LOGIN;
    ctrl->selected = 0;
    dbind(User, String*, name);
    dbind(User, String*, mail);
    dbind(User, Image*, image64);
    dbind(UJson, int32_t, code);
    dbind(UJson, User, data);
    return ctrl;
}

/*---------------------------------------------------------------------------*/

void ctrl_destroy(Ctrl **ctrl)
{
    heap_delete(ctrl, Ctrl);
}

/*---------------------------------------------------------------------------*/

void ctrl_main_layout(Ctrl *ctrl, Layout *layout)
{
    model_layout(layout);
    ctrl->main_layout = layout;
}

/*---------------------------------------------------------------------------*/

void ctrl_status_layout(Ctrl *ctrl, Layout *layout)
{
    ctrl->status_layout = layout;
}

/*---------------------------------------------------------------------------*/
```

```
static void i_update_product(Ctrl *ctrl)
{
    uint32_t total = model_count(ctrl->model);
    bool_t enabled = FALSE;
    bool_t is_first = (total == 0 || ctrl->selected == 0) ? TRUE : FALSE;
    bool_t is_last = (total == 0 || ctrl->selected == (total - 1)) ? TRUE :
        ↪ FALSE;
    Slider *slider = cell_slider(ctrl->slider_cell);
    Label *counter = cell_label(ctrl->counter_cell);
    Product *product = NULL;

    if (total > 0)
    {
        char_t msg[64];
        uint32_t i, n = sizeof(ctrl->stats) / sizeof(real32_t);
        View *vstats = cell_view(ctrl->stats_cell);
        product = model_product(ctrl->model, ctrl->selected);
        bstd_sprintf(msg, 64, "[%d/%d]", ctrl->selected + 1, total);
        label_text(counter, msg);
        slider_value(slider, (real32_t)ctrl->selected / (real32_t)(total > 1 ?
            ↪ total - 1 : 1));
        enabled = TRUE;
        for (i = 0; i < n; ++i)
            ctrl->stats[i] = bmath_randf(2.f, i_MAX_STATS - 2.f);
        view_update(vstats);
    }
    else
    {
        label_text(counter, "[0/0]");
        slider_value(slider, 0.f);
        enabled = FALSE;
    }

    layout_dbind_obj(ctrl->main_layout, product, Product);
    cell_enabled(ctrl->add_cell, enabled);
    cell_enabled(ctrl->minus_cell, enabled);
    cell_enabled(ctrl->slider_cell, enabled);
    cell_enabled(ctrl->filter_cell, enabled);
    cell_enabled(ctrl->first_cell, !is_first);
    cell_enabled(ctrl->back_cell, !is_first);
    cell_enabled(ctrl->next_cell, !is_last);
    cell_enabled(ctrl->last_cell, !is_last);
    menuitem_enabled(ctrl->first_item, !is_first);
    menuitem_enabled(ctrl->back_item, !is_first);
    menuitem_enabled(ctrl->next_item, !is_last);
    menuitem_enabled(ctrl->last_item, !is_last);
}

/*---------------------------------------------------------------------------*/
```

```
static void i_status(Ctrl *ctrl)
{
    ImageView *view = layout_get_imageview(ctrl->status_layout, 0, 0);
    Label *label = layout_get_label(ctrl->status_layout, 1, 0);

    switch (ctrl->status) {
    case ekWAIT_LOGIN:
        imageview_image(view, (const Image*)LOGIN16_PNG);
        label_text(label, WAIT_LOGIN);
        break;

    case ekIN_LOGIN:
        imageview_image(view, (const Image*)SPIN_GIF);
        label_text(label, IN_LOGIN);
        break;

    case ekERR_LOGIN:
        imageview_image(view, (const Image*)ERROR_PNG);
        switch (ctrl->err) {
        case ekWS_CONNECT:
            label_text(label, ERR_CONNECT);
            break;
        case ekWS_JSON:
            label_text(label, ERR_JSON);
            break;
        case ekWS_ACCESS:
            label_text(label, ERR_ACCESS);
            break;
        case ekWS_OK:
        cassert_default();
        }
        break;

    case ekOK_LOGIN:
        imageview_image(view, (const Image*)OK_PNG);
        label_text(label, OK_LOGIN);
        break;

    cassert_default();
    }
}

/*---------------------------------------------------------------------------*/

void ctrl_run(Ctrl *ctrl)
{
    Button *setting_button;
    PopUp *lang_popup;
    MenuItem *lang_item;
    uint32_t lang_index;
    ctrl->status = ekWAIT_LOGIN;
```

```
    setting_button = cell_button(ctrl->setting_cell);
    layout_show_col(ctrl->main_layout, 1, TRUE);
    button_state(setting_button, ekGUI_ON);
    menuitem_state(ctrl->setting_item, ekGUI_ON);
    lang_popup = cell_popup(ctrl->lang_cell);
    lang_index = popup_get_selected(lang_popup);
    lang_item = menu_get_item(ctrl->lang_menu, lang_index);
    menuitem_state(lang_item, ekGUI_ON);
    menuitem_enabled(ctrl->login_item, TRUE);
    menuitem_enabled(ctrl->logout_item, FALSE);
    menuitem_enabled(ctrl->import_item, FALSE);
    menuitem_enabled(ctrl->export_item, FALSE);
    i_status(ctrl);
    cell_focus(ctrl->user_cell);
    i_update_product(ctrl);
    window_defbutton(ctrl->window, cell_button(ctrl->login_cell));
}

/*-----------------------------------------------------------------------*/

static void i_OnFirst(Ctrl *ctrl, Event *e)
{
    ctrl->selected = 0;
    i_update_product(ctrl);
    unref(e);
}

/*-----------------------------------------------------------------------*/

static void i_OnImport(Ctrl *ctrl, Event *e)
{
    const char_t *type[] = { "dbp" };
    const char_t *file = comwin_open_file(ctrl->window, type, 1, NULL);
    if (file != NULL)
    {
        ferror_t err;
        if (model_import(ctrl->model, file, &err) == TRUE)
            i_update_product(ctrl);
    }
    unref(e);
}

/*-----------------------------------------------------------------------*/

void ctrl_import_item(Ctrl *ctrl, MenuItem *item)
{
    ctrl->import_item = item;
    menuitem_OnClick(item, listener(ctrl, i_OnImport, Ctrl));
}

/*-----------------------------------------------------------------------*/
```

```
static void i_OnExport(Ctrl *ctrl, Event *e)
{
    const char_t *type[] = { "dbp" };
    const char_t *file = comwin_save_file(ctrl->window, type, 1, NULL);
    if (file != NULL)
    {
        ferror_t err;
        model_export(ctrl->model, file, &err);
    }
    unref(e);
}

/*-----------------------------------------------------------------------------*/

void ctrl_export_item(Ctrl *ctrl, MenuItem *item)
{
    ctrl->export_item = item;
    menuitem_OnClick(item, listener(ctrl, i_OnExport, Ctrl));
}

/*-----------------------------------------------------------------------------*/

static void i_OnImgDraw(Ctrl *ctrl, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    const Image *image = gui_image(EDIT_PNG);
    uint32_t w = image_width(image);
    uint32_t h = image_height(image);
    draw_image(params->ctx, image, params->width - w - 10, params->height - h -
        ↪ 10);
    unref(ctrl);
}

/*-----------------------------------------------------------------------------*/

static void i_OnImgClick(Ctrl *ctrl, Event *e)
{
    const char_t *type[] = { "png", "jpg" };
    const char_t *file = comwin_open_file(ctrl->window, type, 2, NULL);
    if (file != NULL)
    {
        Image *image = image_from_file(file, NULL);
        if (image != NULL)
        {
            ImageView *view = cell_imageview(ctrl->image_cell);
            imageview_image(view, image);
            image_destroy(&image);
        }
    }
    unref(e);
```

```
}

/*-------------------------------------------------------------------------*/

void ctrl_image_cell(Ctrl *ctrl, Cell *cell)
{
    ImageView *view = cell_imageview(cell);
    model_image(cell);
    imageview_OnOverDraw(view, listener(ctrl, i_OnImgDraw, Ctrl));
    imageview_OnClick(view, listener(ctrl, i_OnImgClick, Ctrl));
    ctrl->image_cell = cell;
}

/*-------------------------------------------------------------------------*/

void ctrl_first_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnFirst, Ctrl));
    ctrl->first_cell = cell;
}

/*-------------------------------------------------------------------------*/

void ctrl_first_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnFirst, Ctrl));
    ctrl->first_item = item;
}

/*-------------------------------------------------------------------------*/

static void i_OnBack(Ctrl *ctrl, Event *e)
{
    if (ctrl->selected > 0)
    {
        ctrl->selected -= 1;
        i_update_product(ctrl);
    }
    unref(e);
}

/*-------------------------------------------------------------------------*/

void ctrl_back_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnBack, Ctrl));
    ctrl->back_cell = cell;
}
```

```
/*-----------------------------------------------------------------------------*/

void ctrl_back_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnBack, Ctrl));
    ctrl->back_item = item;
}

/*-----------------------------------------------------------------------------*/

static void i_OnNext(Ctrl *ctrl, Event *e)
{
    uint32_t total = model_count(ctrl->model);
    if (ctrl->selected < total - 1)
    {
        ctrl->selected += 1;
        i_update_product(ctrl);
    }
    unref(e);
}

/*-----------------------------------------------------------------------------*/

void ctrl_next_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnNext, Ctrl));
    ctrl->next_cell = cell;
}

/*-----------------------------------------------------------------------------*/

void ctrl_next_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnNext, Ctrl));
    ctrl->next_item = item;
}

/*-----------------------------------------------------------------------------*/

static void i_OnLast(Ctrl *ctrl, Event *e)
{
    uint32_t total = model_count(ctrl->model);
    if (ctrl->selected < total - 1)
    {
        ctrl->selected = total - 1;
        i_update_product(ctrl);
    }
    unref(e);
}
```

```
/*---------------------------------------------------------------------------*/

void ctrl_last_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnLast, Ctrl));
    ctrl->last_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_last_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnLast, Ctrl));
    ctrl->last_item = item;
}

/*---------------------------------------------------------------------------*/

static void i_OnAdd(Ctrl *ctrl, Event *e)
{
    model_add(ctrl->model);
    ctrl->selected = model_count(ctrl->model) - 1;
    i_update_product(ctrl);
    cell_focus(ctrl->code_cell);
    unref(e);
}

/*---------------------------------------------------------------------------*/

void ctrl_add_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnAdd, Ctrl));
    ctrl->add_cell = cell;
}

/*---------------------------------------------------------------------------*/

static void i_OnDelete(Ctrl *ctrl, Event *e)
{
    model_delete(ctrl->model, ctrl->selected);
    if (ctrl->selected == model_count(ctrl->model) && ctrl->selected > 0)
        ctrl->selected -= 1;
    i_update_product(ctrl);
    unref(e);
}

/*---------------------------------------------------------------------------*/

void ctrl_minus_cell(Ctrl *ctrl, Cell *cell)
```

```
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnDelete, Ctrl));
    ctrl->minus_cell = cell;
}

/*---------------------------------------------------------------------------*/

static void i_OnFilter(Ctrl *ctrl, Event *e)
{
    const EvText *params = event_params(e, EvText);
    EvTextFilter *result = event_result(e, EvTextFilter);
    Combo *combo = event_sender(e, Combo);
    uint32_t color = color_rgb(255, 0, 0);

    if (unicode_nchars(params->text, ekUTF8) >= 3)
    {
        if (model_filter(ctrl->model, params->text) == TRUE)
        {
            color = UINT32_MAX;
            ctrl->selected = 0;
            i_update_product(ctrl);
        }
    }

    combo_color(combo, color);
    result->apply = FALSE;
}

/*---------------------------------------------------------------------------*/

static void i_OnFilterEnd(Ctrl *ctrl, Event *e)
{
    const EvText *params = event_params(e, EvText);
    Combo *combo = event_sender(e, Combo);

    if (model_filter(ctrl->model, params->text) == TRUE)
        combo_ins_elem(combo, 0, params->text, NULL);
    else
        combo_text(combo, "");

    ctrl->selected = 0;
    i_update_product(ctrl);

    combo_color(combo, UINT32_MAX);
}

/*---------------------------------------------------------------------------*/

void ctrl_filter_cell(Ctrl *ctrl, Cell *cell)
{
```

```
    Combo *combo = cell_combo(cell);
    combo_OnFilter(combo, listener(ctrl, i_OnFilter, Ctrl));
    combo_OnChange(combo, listener(ctrl, i_OnFilterEnd, Ctrl));
    ctrl->filter_cell = cell;
}

/*-----------------------------------------------------------------------------*/

static void i_OnSlider(Ctrl *ctrl, Event *e)
{
    const EvSlider *params = event_params(e, EvSlider);
    uint32_t total = model_count(ctrl->model);
    uint32_t selected = 0;
    if (total > 0)
        selected = (uint32_t)((real32_t)(total - 1) * params->pos);

    if (selected != ctrl->selected)
    {
        ctrl->selected = selected;
        i_update_product(ctrl);
    }
}

/*-----------------------------------------------------------------------------*/

void ctrl_slider_cell(Ctrl *ctrl, Cell *cell)
{
    Slider *slider = cell_slider(cell);
    slider_OnMoved(slider, listener(ctrl, i_OnSlider, Ctrl));
    ctrl->slider_cell = cell;
}

/*-----------------------------------------------------------------------------*/

void ctrl_counter_cell(Ctrl *ctrl, Cell *cell)
{
    ctrl->counter_cell = cell;
}

/*-----------------------------------------------------------------------------*/

void ctrl_type_cell(Ctrl *ctrl, Cell *cell)
{
    model_type(cell);
    unref(ctrl);
}

/*-----------------------------------------------------------------------------*/

void ctrl_code_cell(Ctrl *ctrl, Cell *cell)
{
```

```
    model_code(cell);
    ctrl->code_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_desc_cell(Ctrl *ctrl, Cell *cell)
{
    model_desc(cell);
    ctrl->desc_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_price_cell(Ctrl *ctrl, Cell *cell)
{
    model_price(cell);
    ctrl->price_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_user_cell(Ctrl *ctrl, Cell *cell)
{
    ctrl->user_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_pass_cell(Ctrl *ctrl, Cell *cell)
{
    ctrl->pass_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_login_panel(Ctrl *ctrl, Panel *panel)
{
    ctrl->login_panel = panel;
}

/*---------------------------------------------------------------------------*/

static UJson *i_user_webserv(const char_t *user, const char_t *pass, wserv_t *
    ↪ ret)
{
    Http *http = NULL;
    String *path = NULL;
    UJson *ujson = NULL;

    *ret = ekWS_OK;
```

```
    if (str_empty_c(user) || str_empty_c(pass))
    {
        *ret = ekWS_ACCESS;
        return NULL;
    }

    http = http_create("serv.nappgui.com", 80);
    path = str_printf("/duser.php?user=%s&pass=%s", user, pass);
    if (http_get(http, tc(path), NULL, 0, NULL) == TRUE)
    {
        uint32_t status = http_response_status(http);
        if (status >= 200 && status <= 299)
        {
            Stream *stm = stm_memory(4096);
            http_response_body(http, stm, NULL);
            ujson = json_read(stm, NULL, UJson);

            if (!ujson)
            {
                *ret = ekWS_JSON;
            }
            else if (ujson->code != 0)
            {
                json_destroy(&ujson, UJson);
                *ret = ekWS_ACCESS;
            }

            stm_close(&stm);
        }
        else
        {
            *ret = ekWS_ACCESS;
        }
    }

    str_destroy(&path);
    http_destroy(&http);
    return ujson;
}

/*---------------------------------------------------------------------------*/

static uint32_t i_login_begin(Ctrl *ctrl)
{
    Edit *user = cell_edit(ctrl->user_cell);
    Edit *pass = cell_edit(ctrl->pass_cell);
    wserv_t ret = ekWS_OK;
    ctrl->ujson = i_user_webserv(edit_get_text(user), edit_get_text(pass), &ret
        ↪ );
    if (ctrl->ujson != NULL)
    {
```

```
        ret = model_webserv(ctrl->model);
        if (ret != ekWS_OK)
            json_destroy(&ctrl->ujson, UJson);
    }

    return (uint32_t)ret;
}

/*---------------------------------------------------------------------------*/

static void i_login_end(Ctrl *ctrl, const uint32_t rvalue)
{
    wserv_t ret = (wserv_t)rvalue;
    if (ret == ekWS_OK)
    {
        Layout *layout = panel_get_layout(ctrl->login_panel, 1);
        ImageView *view = layout_get_imageview(layout, 0, 0);
        Label *label0 = layout_get_label(layout, 0, 1);
        Label *label1 = layout_get_label(layout, 0, 2);
        window_defbutton(ctrl->window, NULL);
        imageview_image(view, ctrl->ujson->data.image64);
        label_text(label0, tc(ctrl->ujson->data.name));
        label_text(label1, tc(ctrl->ujson->data.mail));
        menuitem_enabled(ctrl->login_item, FALSE);
        menuitem_enabled(ctrl->logout_item, TRUE);
        menuitem_enabled(ctrl->import_item, TRUE);
        menuitem_enabled(ctrl->export_item, TRUE);
        panel_visible_layout(ctrl->login_panel, 1);
        ctrl->status = ekOK_LOGIN;
        ctrl->selected = 0;
        i_update_product(ctrl);
        json_destroy(&ctrl->ujson, UJson);
        cell_focus(ctrl->code_cell);
        panel_update(ctrl->login_panel);
    }
    else
    {
        cassert(ctrl->ujson == NULL);
        ctrl->status = ekERR_LOGIN;
        ctrl->err = ret;
    }

    i_status(ctrl);
}

/*---------------------------------------------------------------------------*/

static void i_OnLogin(Ctrl *ctrl, Event *e)
{
    if (ctrl->status != ekIN_LOGIN)
    {
```

```
        ctrl->status = ekIN_LOGIN;
        i_status(ctrl);
        osapp_task(ctrl, 0, i_login_begin, NULL, i_login_end, Ctrl);
    }

    unref(e);
}

/*-----------------------------------------------------------------------------*/

void ctrl_login_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnLogin, Ctrl));
    ctrl->login_cell = cell;
}

/*-----------------------------------------------------------------------------*/

void ctrl_login_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnLogin, Ctrl));
    ctrl->login_item = item;
}

/*-----------------------------------------------------------------------------*/

static void i_OnLogout(Ctrl *ctrl, Event *e)
{
    Edit *edit0 = cell_edit(ctrl->user_cell);
    Edit *edit1 = cell_edit(ctrl->pass_cell);
    model_clear(ctrl->model);
    edit_text(edit0, "");
    edit_text(edit1, "");
    menuitem_enabled(ctrl->login_item, TRUE);
    menuitem_enabled(ctrl->logout_item, FALSE);
    menuitem_enabled(ctrl->import_item, FALSE);
    menuitem_enabled(ctrl->export_item, FALSE);
    ctrl->status = ekWAIT_LOGIN;
    panel_visible_layout(ctrl->login_panel, 0);
    i_update_product(ctrl);
    i_status(ctrl);
    cell_focus(ctrl->user_cell);
    panel_update(ctrl->login_panel);
    window_defbutton(ctrl->window, cell_button(ctrl->login_cell));
    unref(e);
}

/*-----------------------------------------------------------------------------*/

void ctrl_logout_cell(Ctrl *ctrl, Cell *cell)
```

```
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnLogout, Ctrl));
    ctrl->logout_cell = cell;
}

/*-------------------------------------------------------------------------*/

void ctrl_logout_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnLogout, Ctrl));
    ctrl->logout_item = item;
}

/*-------------------------------------------------------------------------*/

static void i_OnSetting(Ctrl *ctrl, Event *e)
{
    gui_state_t state = ekGUI_ON;
    if (event_type(e) == ekGUI_EVENT_BUTTON)
    {
        const EvButton *params = event_params(e, EvButton);
        state = params->state;
    }
    else
    {
        Button *button = cell_button(ctrl->setting_cell);
        cassert(event_type(e) == ekGUI_EVENT_MENU);
        state = button_get_state(button);
        state = state == ekGUI_ON ? ekGUI_OFF : ekGUI_ON;
        button_state(button, state);
    }

    menuitem_state(ctrl->setting_item, state);
    layout_show_col(ctrl->main_layout, 1, state == ekGUI_ON ? TRUE : FALSE);
    layout_update(ctrl->main_layout);
}

/*-------------------------------------------------------------------------*/

void ctrl_setting_cell(Ctrl *ctrl, Cell *cell)
{
    Button *button = cell_button(cell);
    button_OnClick(button, listener(ctrl, i_OnSetting, Ctrl));
    ctrl->setting_cell = cell;
}

/*-------------------------------------------------------------------------*/

void ctrl_setting_item(Ctrl *ctrl, MenuItem *item)
{
```

```
    menuitem_OnClick(item, listener(ctrl, i_OnSetting, Ctrl));
    ctrl->setting_item = item;
}

/*---------------------------------------------------------------------------*/

static void i_OnStats(Ctrl *ctrl, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    uint32_t i, n = sizeof(ctrl->stats) / sizeof(real32_t);
    real32_t p = 10.f, x = p, y0 = params->height - p;
    real32_t w = (params->width - p * 2) / n;
    real32_t h = params->height - p * 2;
    real32_t avg = 0, pavg;
    char_t tavg[16];
    color_t c[2];
    real32_t stop[2] = {0, 1};
    c[0] = kHOLDER;
    c[1] = gui_view_color();

    draw_fill_linear(params->ctx, c,stop, 2, 0, p, 0, params->height - p + 1);

    for (i = 0; i < n; ++i)
    {
        real32_t hr = h * (ctrl->stats[i] / i_MAX_STATS);
        real32_t y = p + h - hr;
        draw_rect(params->ctx, ekFILL, x, y, w - 2, hr);
        avg += ctrl->stats[i];
        x += w;
    }

    avg /= n;
    pavg = h * (avg / i_MAX_STATS);
    pavg = p + h - pavg;
    bstd_sprintf(tavg, sizeof(tavg), "%.2f", avg);
    draw_text_color(params->ctx, kTXTRED);
    draw_line_color(params->ctx, kTXTRED);
    draw_line(params->ctx, p - 2, pavg, params->width - p, pavg);
    draw_line_color(params->ctx, gui_label_color());
    draw_line(params->ctx, p - 2, y0 + 2, params->width - p, y0 + 2);
    draw_line(params->ctx, p - 2, y0 + 2, p - 2, p);
    draw_text(params->ctx, tavg, p, pavg);
}

/*---------------------------------------------------------------------------*/

void ctrl_stats_cell(Ctrl *ctrl, Cell *cell)
{
    View *view = cell_view(cell);
    view_OnDraw(view, listener(ctrl, i_OnStats, Ctrl));
    ctrl->stats_cell = cell;
```

```
}

/*---------------------------------------------------------------------------*/

static void i_OnLang(Ctrl *ctrl, Event *e)
{
    MenuItem *item = NULL;
    uint32_t lang_id = 0;
    static const char_t *LANGS[] = { "en_US", "es_ES", "pt_PT", "it_IT", "vi_VN
        ↪ ", "ru_RU", "ja_JP" };
    if (event_type(e) == ekGUI_EVENT_POPUP)
    {
        const EvButton *params = event_params(e, EvButton);
        item = menu_get_item(ctrl->lang_menu, params->index);
        lang_id = params->index;
    }
    else
    {
        const EvMenu *params = event_params(e, EvMenu);
        PopUp *popup = cell_popup(ctrl->lang_cell);
        cassert(event_type(e) == ekGUI_EVENT_MENU);
        popup_selected(popup, params->index);
        item = event_sender(e, MenuItem);
        lang_id = params->index;
    }

    menu_off_items(ctrl->lang_menu);
    menuitem_state(item, ekGUI_ON);
    gui_language(LANGS[lang_id]);
}

/*---------------------------------------------------------------------------*/

void ctrl_lang_cell(Ctrl *ctrl, Cell *cell)
{
    PopUp *popup = cell_popup(cell);
    popup_OnSelect(popup, listener(ctrl, i_OnLang, Ctrl));
    ctrl->lang_cell = cell;
}

/*---------------------------------------------------------------------------*/

void ctrl_lang_menu(Ctrl *ctrl, Menu *menu)
{
    uint32_t i, n = menu_size(menu);
    for (i = 0; i < n; ++i)
    {
        MenuItem *item = menu_get_item(menu, i);
        menuitem_OnClick(item, listener(ctrl, i_OnLang, Ctrl));
    }
    ctrl->lang_menu = menu;
```

```
}

/*---------------------------------------------------------------------------*/

static void i_OnExit(Ctrl *ctrl, Event *e)
{
    osapp_finish();
    unref(ctrl);
    unref(e);
}

/*---------------------------------------------------------------------------*/

void ctrl_exit_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnExit, Ctrl));
}

/*---------------------------------------------------------------------------*/

static void i_OnAbout(Ctrl *ctrl, Event *e)
{
    unref(ctrl);
    unref(e);
    osapp_open_url("https://nappgui.com/en/demo/products.html");
}

/*---------------------------------------------------------------------------*/

void ctrl_about_item(Ctrl *ctrl, MenuItem *item)
{
    menuitem_OnClick(item, listener(ctrl, i_OnAbout, Ctrl));
}

/*---------------------------------------------------------------------------*/

void ctrl_window(Ctrl *ctrl, Window *window)
{
    window_OnClose(window, listener(ctrl, i_OnExit, Ctrl));
    ctrl->window = window;
}

/*---------------------------------------------------------------------------*/

void ctrl_theme_images(Ctrl *ctrl)
{
    bool_t dark = gui_dark_mode();
    button_image(cell_button(ctrl->first_cell), (const Image*)(dark ?
        ↪ FIRSTD_PNG : FIRST_PNG));
    button_image(cell_button(ctrl->back_cell), (const Image*)(dark ? BACKD_PNG
        ↪ : BACK_PNG));
```

```
    button_image(cell_button(ctrl->next_cell), (const Image*)(dark ? NEXTD_PNG
        ↪ : NEXT_PNG));
    button_image(cell_button(ctrl->last_cell), (const Image*)(dark ? LASTD_PNG
        ↪ : LAST_PNG));
    button_image(cell_button(ctrl->add_cell), (const Image*)ADD_PNG);
    button_image(cell_button(ctrl->minus_cell), (const Image*)MINUS_PNG);
    button_image(cell_button(ctrl->setting_cell), (const Image*)SETTINGS_PNG);
    button_image(cell_button(ctrl->login_cell), (const Image*)LOGIN16_PNG);
    button_image(cell_button(ctrl->logout_cell), (const Image*)(dark ?
        ↪ LOGOUT16D_PNG : LOGOUT16_PNG));
    menuitem_image(ctrl->import_item, (const Image*)OPEN_PNG);
    menuitem_image(ctrl->export_item, (const Image*)(dark ? SAVED_PNG :
        ↪ SAVE_PNG));
    menuitem_image(ctrl->first_item, (const Image*)(dark ? FIRST16D_PNG :
        ↪ FIRST16_PNG));
    menuitem_image(ctrl->back_item, (const Image*)(dark ? BACK16D_PNG :
        ↪ BACK16_PNG));
    menuitem_image(ctrl->next_item, (const Image*)(dark ? NEXT16D_PNG :
        ↪ NEXT16_PNG));
    menuitem_image(ctrl->last_item, (const Image*)(dark ? LAST16D_PNG :
        ↪ LAST16_PNG));
    menuitem_image(ctrl->login_item, (const Image*)LOGIN16_PNG);
    menuitem_image(ctrl->logout_item, (const Image*)(dark ? LOGOUT16D_PNG :
        ↪ LOGOUT16_PNG));
}
```

Capítulo 26

¡Hola GUI!

GuiHello es una aplicación, que a modo de ejemplo, muestra las características de la librería *"Gui"* (Página 303) para la creación de interfaces de usuario. El **código fuente** está en la caperta `/src/howto/guihello` de la distribución del SDK.

26.1. ¡Hola Label!

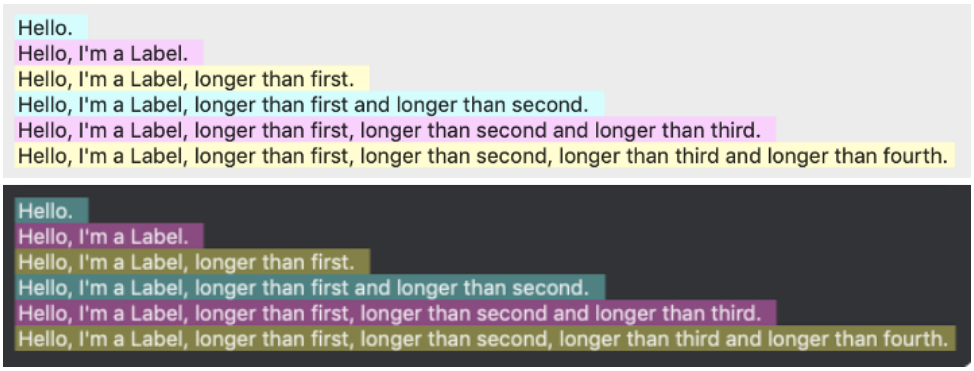


Figura 26.1: Controles `Label`.

Listado 26.1: demo/guihello/labels.c

```
/* Labels basics */

#include "labels.h"
#include <gui/guiall.h>

/*----------------------------------------------------------------------------*/

static const char_t *i_LABEL_01 = "Hello.";
static const char_t *i_LABEL_02 = "Hello, I'm a Label.";
static const char_t *i_LABEL_03 = "Hello, I'm a Label, longer than first.";
static const char_t *i_LABEL_04 = "Hello, I'm a Label, longer than first and
    ↪ longer than second.";
static const char_t *i_LABEL_05 = "Hello, I'm a Label, longer than first,
    ↪ longer than second and longer than third.";
static const char_t *i_LABEL_06 = "Hello, I'm a Label, longer than first,
    ↪ longer than second, longer than third and longer than fourth.";
static const char_t *i_LABEL_07 = "Mouse sensitive label";

/*----------------------------------------------------------------------------*/

static void i_OnLayoutWidth(Layout *layout, Event *event)
{
    const EvButton *p = event_params(event, EvButton);
    real32_t width = 0;
    switch (p->index) {
    case 0:
        width = 0;
        break;
    case 1:
        width = 100;
        break;
    case 2:
        width = 200;
        break;
```

```
    case 3:
        width = 300;
        break;
    case 4:
        width = 400;
        break;
    cassert_default();
    }

    layout_hsize(layout, 0, width);
    layout_update(layout);
}

/*----------------------------------------------------------------------------*/

static PopUp *i_width_popup(Layout *layout)
{
    PopUp *popup = popup_create();
    popup_add_elem(popup, "Natural", NULL);
    popup_add_elem(popup, "100px", NULL);
    popup_add_elem(popup, "200px", NULL);
    popup_add_elem(popup, "300px", NULL);
    popup_add_elem(popup, "400px", NULL);
    popup_OnSelect(popup, listener(layout, i_OnLayoutWidth, Layout));
    return popup;
}

/*----------------------------------------------------------------------------*/

Panel *labels_single_line(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 7);
    PopUp *popup = i_width_popup(layout);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_create();
    color_t c1 = gui_alt_color(color_rgb(192, 255, 255), color_rgb(48, 112,
        ↪ 112));
    color_t c2 = gui_alt_color(color_rgb(255, 192, 255), color_rgb(128, 48,
        ↪ 112));
    color_t c3 = gui_alt_color(color_rgb(255, 255, 192), color_rgb(112, 112,
        ↪ 48));
    label_text(label1, i_LABEL_01);
    label_text(label2, i_LABEL_02);
    label_text(label3, i_LABEL_03);
    label_text(label4, i_LABEL_04);
    label_text(label5, i_LABEL_05);
```

```
    label_text(label6, i_LABEL_06);
    label_bgcolor(label1, c1);
    label_bgcolor(label2, c2);
    label_bgcolor(label3, c3);
    label_bgcolor(label4, c1);
    label_bgcolor(label5, c2);
    label_bgcolor(label6, c3);
    layout_popup(layout, popup, 0, 0);
    layout_label(layout, label1, 0, 1);
    layout_label(layout, label2, 0, 2);
    layout_label(layout, label3, 0, 3);
    layout_label(layout, label4, 0, 4);
    layout_label(layout, label5, 0, 5);
    layout_label(layout, label6, 0, 6);
    layout_vmargin(layout, 0, 5);
    panel_layout(panel, layout);
    return panel;
}

/*-------------------------------------------------------------------------*/

Panel *labels_multi_line(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 7);
    PopUp *popup = i_width_popup(layout);
    Label *label1 = label_multiline();
    Label *label2 = label_multiline();
    Label *label3 = label_multiline();
    Label *label4 = label_multiline();
    Label *label5 = label_multiline();
    Label *label6 = label_multiline();
    color_t c1 = gui_alt_color(color_rgb(192, 255, 255), color_rgb(48, 112,
        ↪ 112));
    color_t c2 = gui_alt_color(color_rgb(255, 192, 255), color_rgb(128, 48,
        ↪ 112));
    color_t c3 = gui_alt_color(color_rgb(255, 255, 192), color_rgb(112, 112,
        ↪ 48));
    label_text(label1, i_LABEL_01);
    label_text(label2, i_LABEL_02);
    label_text(label3, i_LABEL_03);
    label_text(label4, i_LABEL_04);
    label_text(label5, i_LABEL_05);
    label_text(label6, i_LABEL_06);
    label_bgcolor(label1, c1);
    label_bgcolor(label2, c2);
    label_bgcolor(label3, c3);
    label_bgcolor(label4, c1);
    label_bgcolor(label5, c2);
    label_bgcolor(label6, c3);
    label_align(label4, ekLEFT);
```

```
    label_align(label5, ekCENTER);
    label_align(label6, ekRIGHT);
    layout_popup(layout, popup, 0, 0);
    layout_label(layout, label1, 0, 1);
    layout_label(layout, label2, 0, 2);
    layout_label(layout, label3, 0, 3);
    layout_label(layout, label4, 0, 4);
    layout_label(layout, label5, 0, 5);
    layout_label(layout, label6, 0, 6);
    layout_vmargin(layout, 0, 5);
    panel_layout(panel, layout);
    return panel;
}

/*----------------------------------------------------------------------------*/

Panel *labels_mouse_over(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 5);
    Font *font = font_system(20, ekFNORMAL | ekFPIXELS);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    label_text(label1, i_LABEL_07);
    label_text(label2, i_LABEL_07);
    label_text(label3, i_LABEL_07);
    label_text(label4, i_LABEL_07);
    label_text(label5, i_LABEL_07);
    label_font(label1, font);
    label_font(label2, font);
    label_font(label3, font);
    label_font(label4, font);
    label_font(label5, font);
    label_color_over(label1, kCOLOR_RED);
    label_color_over(label2, kCOLOR_RED);
    label_color_over(label3, kCOLOR_RED);
    label_color_over(label4, kCOLOR_RED);
    label_color_over(label5, kCOLOR_RED);
    label_style_over(label1, ekFBOLD);
    label_style_over(label2, ekFITALIC);
    label_style_over(label3, ekFSTRIKEOUT);
    label_style_over(label4, ekFUNDERLINE);
    label_bgcolor_over(label5, kCOLOR_CYAN);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 2);
    layout_label(layout, label4, 0, 3);
    layout_label(layout, label5, 0, 4);
```

```
    panel_layout(panel, layout);
    font_destroy(&font);
    return panel;
}
```

26.2. ¡Hola Button!

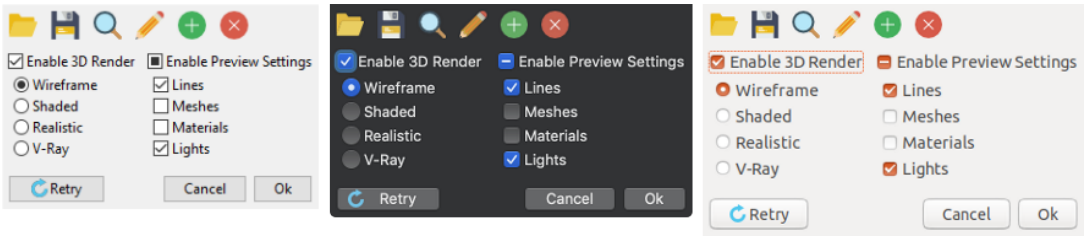


Figura 26.2: Controles `Button`.

Listado 26.2: demo/guihello/buttons.c

```
/* Buttons demo */

#include "buttons.h"
#include "res_guihello.h"
#include <gui/guiall.h>

/*----------------------------------------------------------------------------*/

static Layout *i_flatbuttons(void)
{
    Layout *layout = layout_create(6, 1);
    Button *button1 = button_flat();
    Button *button2 = button_flat();
    Button *button3 = button_flat();
    Button *button4 = button_flat();
    Button *button5 = button_flat();
    Button *button6 = button_flat();
    button_text(button1, "Open File");
    button_text(button2, "Save File");
    button_text(button3, "Search File");
    button_text(button4, "Edit File");
    button_text(button5, "Add File");
    button_text(button6, "Delete File");
    button_image(button1, resid_image(FOLDER24_PNG));
    button_image(button2, resid_image(DISK24_PNG));
    button_image(button3, resid_image(SEARCH24_PNG));
    button_image(button4, resid_image(EDIT24_PNG));
    button_image(button5, resid_image(PLUS24_PNG));
    button_image(button6, resid_image(ERROR24_PNG));
    layout_button(layout, button1, 0, 0);
```

```
    layout_button(layout, button2, 1, 0);
    layout_button(layout, button3, 2, 0);
    layout_button(layout, button4, 3, 0);
    layout_button(layout, button5, 4, 0);
    layout_button(layout, button6, 5, 0);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_radios(void)
{
    Layout *layout = layout_create(1, 4);
    Button *radio1 = button_radio();
    Button *radio2 = button_radio();
    Button *radio3 = button_radio();
    Button *radio4 = button_radio();
    button_text(radio1, "&Wireframe");
    button_text(radio2, "&Shaded");
    button_text(radio3, "&Realistic");
    button_text(radio4, "&V-Ray");
    button_state(radio1, ekGUI_ON);
    layout_button(layout, radio1, 0, 0);
    layout_button(layout, radio2, 0, 1);
    layout_button(layout, radio3, 0, 2);
    layout_button(layout, radio4, 0, 3);
    layout_margin(layout, 5);
    layout_vmargin(layout, 0, 3);
    layout_vmargin(layout, 1, 3);
    layout_vmargin(layout, 2, 3);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_checks(void)
{
    Layout *layout = layout_create(1, 4);
    Button *check1 = button_check();
    Button *check2 = button_check();
    Button *check3 = button_check();
    Button *check4 = button_check();
    button_text(check1, "&Lines");
    button_text(check2, "M&eshes");
    button_text(check3, "M&aterials");
    button_text(check4, "L&ights");
    button_state(check1, ekGUI_ON);
    button_state(check2, ekGUI_OFF);
    button_state(check3, ekGUI_OFF);
    button_state(check4, ekGUI_ON);
    layout_button(layout, check1, 0, 0);
```

```
    layout_button(layout, check2, 0, 1);
    layout_button(layout, check3, 0, 2);
    layout_button(layout, check4, 0, 3);
    layout_margin(layout, 5);
    layout_vmargin(layout, 0, 3);
    layout_vmargin(layout, 1, 3);
    layout_vmargin(layout, 2, 3);
    return layout;
}

/*----------------------------------------------------------------------------*/

static Layout *i_pushes(Button **defbutton)
{
    Layout *layout = layout_create(4, 1);
    Button *button1 = button_push();
    Button *button2 = button_push();
    Button *button3 = button_push();
    button_text(button1, "Re&try");
    button_text(button2, "&Cancel");
    button_text(button3, "&Ok");
    button_image(button1, resid_image(RETRY_PNG));
    layout_button(layout, button1, 0, 0);
    layout_button(layout, button2, 2, 0);
    layout_button(layout, button3, 3, 0);
    layout_hmargin(layout, 2, 5);
    layout_hexpand(layout, 1);
    *defbutton = button1;
    return layout;
}

/*----------------------------------------------------------------------------*/

static Layout *i_buttons(Button **defbutton)
{
    Layout *layout = layout_create(1, 3);
    Layout *layout1 = i_flatbuttons();
    Layout *layout2 = layout_create(2, 2);
    Layout *layout3 = i_radios();
    Layout *layout4 = i_checks();
    Layout *layout5 = i_pushes(defbutton);
    Button *check1 = button_check();
    Button *check2 = button_check3();
    button_text(check1, "Enable 3&D Render");
    button_text(check2, "Enable &Preview Settings");
    button_state(check1, ekGUI_ON);
    button_state(check2, ekGUI_MIXED);
    layout_layout(layout, layout1, 0, 0);
    layout_button(layout2, check1, 0, 0);
    layout_layout(layout2, layout3, 0, 1);
    layout_button(layout2, check2, 1, 0);
```

```
    layout_layout(layout2, layout4, 1, 1);
    layout_layout(layout, layout2, 0, 1);
    layout_layout(layout, layout5, 0, 2);
    layout_halign(layout, 0, 0, ekLEFT);
    layout_margin(layout2, 5);
    layout_hmargin(layout2, 0, 10);
    layout_margin(layout5, 5);
    return layout;
}

/*-------------------------------------------------------------------------*/

Panel *buttons_basics(Button **defbutton)
{
    Layout *layout = i_buttons(defbutton);
    Panel *panel = panel_create();
    panel_layout(panel, layout);
    return panel;
}
```

26.3. ¡Hola PopUp y Combo!

Figura 26.3: Controles `PopUp`.

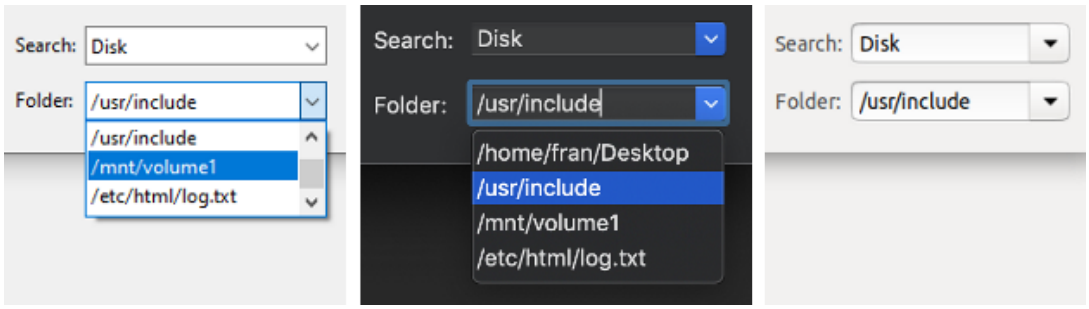


Figura 26.4: Controles `Combo`.

Listado 26.3: demo/guihello/popcom.c

```
/* PopUp and Combo */

#include "popcom.h"
#include "res_guihello.h"
#include <gui/guiall.h>

/*-------------------------------------------------------------------------*/

static void i_popups(Layout *layout)
{
    Label *label1 = label_create();
    Label *label2 = label_create();
    PopUp *popup1 = popup_create();
    PopUp *popup2 = popup_create();
    label_text(label1, "Language:");
    label_text(label2, "Color:");
    popup_add_elem(popup1, "English", (const Image*)UKING_PNG);
    popup_add_elem(popup1, "Español", (const Image*)SPAIN_PNG);
    popup_add_elem(popup1, "Portugues", (const Image*)PORTUGAL_PNG);
    popup_add_elem(popup1, "Italiana", (const Image*)ITALY_PNG);
    popup_add_elem(popup1, "ÊTing ệVit", (const Image*)VIETNAM_PNG);
    popup_add_elem(popup1, "Россиии", (const Image*)RUSSIA_PNG);
    popup_add_elem(popup1, "□□□", (const Image*)JAPAN_PNG);
    popup_add_elem(popup2, "Red", (const Image*)RED_PNG);
    popup_add_elem(popup2, "Blue", (const Image*)BLUE_PNG);
    popup_add_elem(popup2, "Green", (const Image*)GREEN_PNG);
    popup_add_elem(popup2, "Yellow", (const Image*)YELLOW_PNG);
    popup_add_elem(popup2, "Black", (const Image*)BLACK_PNG);
    popup_add_elem(popup2, "White", (const Image*)WHITE_PNG);
    popup_list_height(popup1, 10);
    popup_list_height(popup2, 10);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_popup(layout, popup1, 1, 0);
    layout_popup(layout, popup2, 1, 1);
}

/*-------------------------------------------------------------------------*/

static void i_combos(Layout *layout)
{
    Label *label1 = label_create();
    Label *label2 = label_create();
    Combo *combo1 = combo_create();
    Combo *combo2 = combo_create();
    label_text(label1, "Search:");
    label_text(label2, "Folder:");
    combo_add_elem(combo1, "Search", NULL);
    combo_add_elem(combo1, "Disk", NULL);
    combo_add_elem(combo1, "Edit", NULL);
    combo_add_elem(combo2, "/home/fran/Desktop", NULL);
```

```
    combo_add_elem(combo2, "/usr/include", NULL);
    combo_add_elem(combo2, "/mnt/volume1", NULL);
    combo_add_elem(combo2, "/etc/html/log.txt", NULL);
    layout_label(layout, label1, 2, 0);
    layout_label(layout, label2, 2, 1);
    layout_combo(layout, combo1, 3, 0);
    layout_combo(layout, combo2, 3, 1);
}

/*-------------------------------------------------------------------------*/

Panel *popup_combo(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(4, 2);
    i_popups(layout);
    i_combos(layout);
    layout_margin(layout, 10.f);
    layout_vmargin(layout, 0, 10.f);
    layout_hmargin(layout, 0, 5.f);
    layout_hmargin(layout, 1, 10.f);
    layout_hmargin(layout, 2, 5.f);
    layout_hsize(layout, 1, 150.f);
    layout_hsize(layout, 3, 150.f);
    panel_layout(panel, layout);
    return panel;
}
```

26.4. ¡Hola Edit y UpDown!

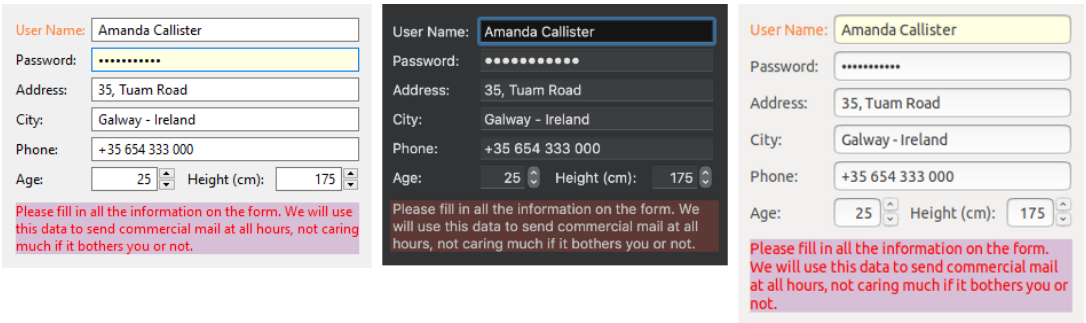


Figura 26.5: Controles `Edit` y `UpDown`.

Listado 26.4: demo/guihello/form.c

```
/* Form demo */

#include "form.h"
#include <gui/guiall.h>
```

```
/*----------------------------------------------------------------------------*/

static void i_OnFilter(void *noused, Event *e)
{
    const EvText *params = event_params(e, EvText);
    EvTextFilter *result = event_result(e, EvTextFilter);
    uint32_t i = 0, j = 0;
    while (params->text[i] != '\0')
    {
        if (params->text[i] >= '0' && params->text[i] <= '9')
        {
            result->text[j] = params->text[i];
            j += 1;
        }

        i += 1;
    }

    result->text[j] = '\0';
    result->apply = TRUE;
    unref(noused);
}

/*----------------------------------------------------------------------------*/

static void i_OnUpDown(Edit *edit, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    int32_t n = str_to_i32(edit_get_text(edit), 10, NULL);
    char_t text[64];
    n += (params->index == 0) ? 1 : -1;
    bstd_sprintf(text, sizeof(text), "%d", n);
    edit_text(edit, text);
}

/*----------------------------------------------------------------------------*/

static Layout *i_numbers(color_t colorbg)
{
    Layout *layout = layout_create(5, 1);
    Label *label = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    UpDown *updown1 = updown_create();
    UpDown *updown2 = updown_create();
    label_text(label, "Height (cm):");
    edit_text(edit1, "25");
    edit_text(edit2, "175");
    edit_align(edit1, ekRIGHT);
    edit_align(edit2, ekRIGHT);
```

```
    edit_OnFilter(edit1, listener(NULL, i_OnFilter, void));
    edit_OnFilter(edit2, listener(NULL, i_OnFilter, void));
    edit_bgcolor_focus(edit1, colorbg);
    edit_bgcolor_focus(edit2, colorbg);
    updown_OnClick(updown1, listener(edit1, i_OnUpDown, Edit));
    updown_OnClick(updown2, listener(edit2, i_OnUpDown, Edit));
    updown_tooltip(updown1, "Increase/Decrease age");
    updown_tooltip(updown2, "Increase/Decrease height");
    layout_label(layout, label, 2, 0);
    layout_edit(layout, edit1, 0, 0);
    layout_edit(layout, edit2, 3, 0);
    layout_updown(layout, updown1, 1, 0);
    layout_updown(layout, updown2, 4, 0);
    layout_hmargin(layout, 1, 10.f);
    layout_hmargin(layout, 2, 10.f);
    layout_hexpand2(layout, 0, 3, .5f);
    return layout;
}

/*----------------------------------------------------------------------------*/

static Layout *i_edits(void)
{
    color_t colorbg = gui_alt_color(color_bgr(0xFFFFe4), color_bgr(0x101010));
    Layout *layout1 = layout_create(2, 6);
    Layout *layout2 = i_numbers(colorbg);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    Edit *edit3 = edit_create();
    Edit *edit4 = edit_create();
    Edit *edit5 = edit_create();
    label_text(label1, "User Name:");
    label_text(label2, "Password:");
    label_text(label3, "Address:");
    label_text(label4, "City:");
    label_text(label5, "Phone:");
    label_text(label6, "Age:");
    label_color_over(label1, color_rgb(255, 128, 52));
    label_color_over(label2, color_rgb(70, 129, 207));
    label_color_over(label3, color_rgb(119, 188, 31));
    label_style_over(label4, ekFITALIC | ekFUNDERLINE);
    edit_text(edit1, "Amanda Callister");
    edit_text(edit2, "aQwe56nhjJk");
    edit_text(edit3, "35, Tuam Road");
    edit_text(edit4, "Galway - Ireland");
```

```
    edit_text(edit5, "+35 654 333 000");
    edit_passmode(edit2, TRUE);
    edit_bgcolor_focus(edit1, colorbg);
    edit_bgcolor_focus(edit2, colorbg);
    edit_bgcolor_focus(edit3, colorbg);
    edit_bgcolor_focus(edit4, colorbg);
    edit_bgcolor_focus(edit5, colorbg);
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 0, 1);
    layout_label(layout1, label3, 0, 2);
    layout_label(layout1, label4, 0, 3);
    layout_label(layout1, label5, 0, 4);
    layout_label(layout1, label6, 0, 5);
    layout_edit(layout1, edit1, 1, 0);
    layout_edit(layout1, edit2, 1, 1);
    layout_edit(layout1, edit3, 1, 2);
    layout_edit(layout1, edit4, 1, 3);
    layout_edit(layout1, edit5, 1, 4);
    layout_layout(layout1, layout2, 1, 5);
    layout_hmargin(layout1, 0, 5);
    layout_hexpand(layout1, 1);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 1, 5);
    layout_vmargin(layout1, 2, 5);
    layout_vmargin(layout1, 3, 5);
    layout_vmargin(layout1, 4, 5);
    return layout1;
}

/*-------------------------------------------------------------------------*/

static Layout *i_form(void)
{
    Layout *layout1 = layout_create(1, 2);
    Layout *layout2 = i_edits();
    Label *label = label_multiline();
    label_text(label, "Please fill in all the information on the form. We will
        ↪ use this data to send commercial mail at all hours, not caring much
        ↪ if it bothers you or not.");
    label_color(label, gui_alt_color(color_rgb(255, 0, 0), color_rgb(180, 180,
        ↪ 180)));
    label_bgcolor(label, gui_alt_color(color_rgb(216, 191, 216), color_rgb(80,
        ↪ 40, 40)));
    label_bgcolor_over(label, gui_alt_color(color_rgb(255, 250, 205), color_rgb
        ↪ (105, 100, 55)));
    label_style_over(label, ekFUNDERLINE);
    layout_layout(layout1, layout2, 0, 0);
    layout_label(layout1, label, 0, 1);
    layout_hsize(layout1, 0, 300);
    layout_vmargin(layout1, 0, 10);
    layout_margin(layout1, 10);
```

```
    return layout1;
}

/*---------------------------------------------------------------------------*/

Panel *form_basic(void)
{
    Layout *layout = i_form();
    Panel *panel = panel_create();
    panel_layout(panel, layout);
    return panel;
}
```

26.5. ¡Hola ListBox!

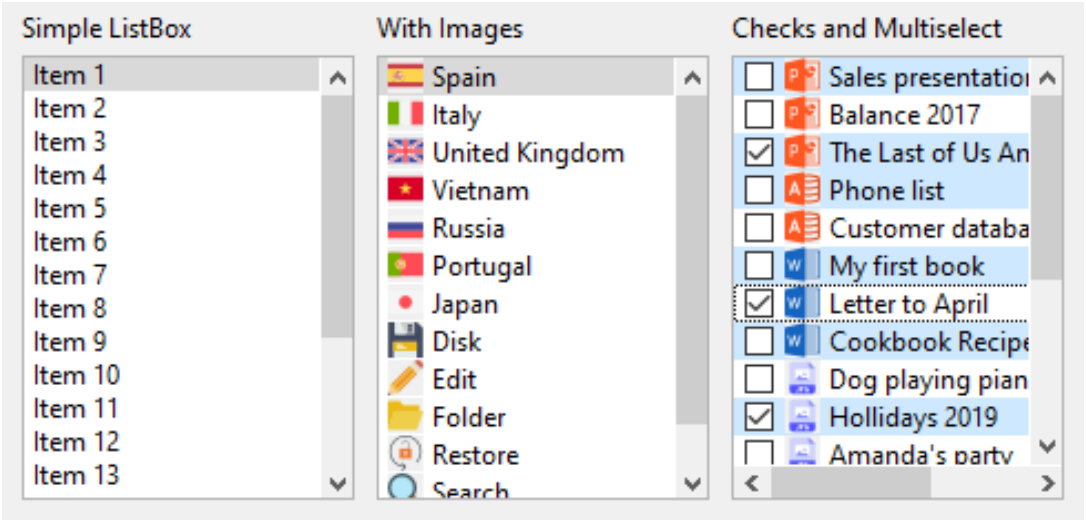


Figura 26.6: Controles `ListBox`.

Listado 26.5: demo/guihello/listboxes.c

```
/* Listboxes */

#include "listboxes.h"
#include "res_guihello.h"
#include <gui/guiall.h>

/*---------------------------------------------------------------------------*/

static ListBox *i_full_listbox(void)
{
    ListBox *listbox = listbox_create();
    listbox_size(listbox, s2df(150, 200));
```

```
    listbox_multisel(listbox, TRUE);
    listbox_checkbox(listbox, TRUE);
    listbox_add_elem(listbox, "Sales presentation", resid_image(POWERPOINT_PNG)
        ↪ );
    listbox_add_elem(listbox, "Balance 2017", resid_image(POWERPOINT_PNG));
    listbox_add_elem(listbox, "The Last of Us Analysis", resid_image(
        ↪ POWERPOINT_PNG));
    listbox_add_elem(listbox, "Phone list", resid_image(ACCESS_PNG));
    listbox_add_elem(listbox, "Customer database", resid_image(ACCESS_PNG));
    listbox_add_elem(listbox, "My first book", resid_image(WORD_PNG));
    listbox_add_elem(listbox, "Letter to April", resid_image(WORD_PNG));
    listbox_add_elem(listbox, "Cookbook Recipes", resid_image(WORD_PNG));
    listbox_add_elem(listbox, "Dog playing piano", resid_image(JPG_PNG));
    listbox_add_elem(listbox, "Hollidays 2019", resid_image(JPG_PNG));
    listbox_add_elem(listbox, "Amanda's party", resid_image(JPG_PNG));
    listbox_add_elem(listbox, "Flying", resid_image(JPG_PNG));
    listbox_add_elem(listbox, "The C Programing Language", resid_image(PDF_PNG)
        ↪ );
    listbox_add_elem(listbox, "Graphics Programing with GDI+", resid_image(
        ↪ PDF_PNG));
    listbox_add_elem(listbox, "Personal finances", resid_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Stocks 2017", resid_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Website Dashboard", resid_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Open Issues", resid_image(DOCUMENT_PNG));
    listbox_add_elem(listbox, "TODO List", resid_image(DOCUMENT_PNG));
    listbox_select(listbox, 0, TRUE);
    return listbox;
}

/*----------------------------------------------------------------------------*/

static ListBox *i_image_listbox(void)
{
    ListBox *listbox = listbox_create();
    listbox_size(listbox, s2df(150, 200));
    listbox_add_elem(listbox, "Spain", resid_image(SPAIN_PNG));
    listbox_add_elem(listbox, "Italy", resid_image(ITALY_PNG));
    listbox_add_elem(listbox, "United Kingdom", resid_image(UKING_PNG));
    listbox_add_elem(listbox, "Vietnam", resid_image(VIETNAM_PNG));
    listbox_add_elem(listbox, "Russia", resid_image(RUSSIA_PNG));
    listbox_add_elem(listbox, "Portugal", resid_image(PORTUGAL_PNG));
    listbox_add_elem(listbox, "Japan", resid_image(JAPAN_PNG));
    listbox_add_elem(listbox, "Disk", resid_image(DISK16_PNG));
    listbox_add_elem(listbox, "Edit", resid_image(EDIT16_PNG));
    listbox_add_elem(listbox, "Folder", resid_image(FOLDER16_PNG));
    listbox_add_elem(listbox, "Restore", resid_image(RESTORE16_PNG));
    listbox_add_elem(listbox, "Search", resid_image(SEARCH16_PNG));
    listbox_add_elem(listbox, "Error", resid_image(ERROR16_PNG));
    listbox_select(listbox, 0, TRUE);
    return listbox;
}
```

```
/*-------------------------------------------------------------------------*/

static ListBox *i_simple_listbox(void)
{
    ListBox *listbox = listbox_create();
    listbox_size(listbox, s2df(150, 200));
    listbox_add_elem(listbox, "Item 1", NULL);
    listbox_add_elem(listbox, "Item 2", NULL);
    listbox_add_elem(listbox, "Item 3", NULL);
    listbox_add_elem(listbox, "Item 4", NULL);
    listbox_select(listbox, 0, TRUE);
    return listbox;
}

/*-------------------------------------------------------------------------*/

Panel *listboxes(void)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(3, 2);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    ListBox *listbox1 = i_simple_listbox();
    ListBox *listbox2 = i_image_listbox();
    ListBox *listbox3 = i_full_listbox();
    label_text(label1, "Simple ListBox");
    label_text(label2, "With Images");
    label_text(label3, "Checks and Multiselect");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 1, 0);
    layout_label(layout, label3, 2, 0);
    layout_listbox(layout, listbox1, 0, 1);
    layout_listbox(layout, listbox2, 1, 1);
    layout_listbox(layout, listbox3, 2, 1);
    layout_hmargin(layout, 0, 10);
    layout_hmargin(layout, 1, 10);
    layout_vmargin(layout, 0, 5);
    panel_layout(panel, layout);
    return panel;
}
```

26.6. ¡Hola Slider y Progress!

Listado 26.6: demo/guihello/sliders.c

```
/* Sliders */

#include "sliders.h"
```

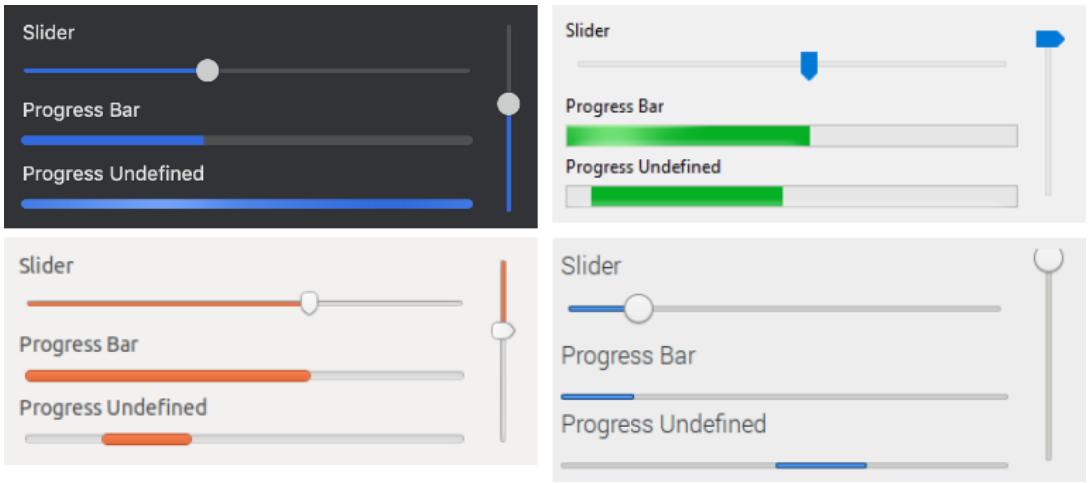


Figura 26.7: Controles `Slider` y `Progress`.

```
#include <gui/guiall.h>

/*----------------------------------------------------------------------------*/

static void i_OnSlider(Progress *prog, Event *event)
{
    const EvSlider *params = event_params(event, EvSlider);
    progress_value(prog, params->pos);
}

/*----------------------------------------------------------------------------*/

Panel *sliders(void)
{
    Layout *layout1 = layout_create(2, 1);
    Layout *layout2 = layout_create(1, 8);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    Slider *slider3 = slider_vertical();
    Progress *prog1 = progress_create();
    Progress *prog2 = progress_create();
    Panel *panel = panel_create();
    label_text(label1, "Slider");
    label_text(label2, "Slider (discrete 6 steps)");
    label_text(label3, "Progress Bar");
    label_text(label4, "Progress Undefined");
    slider_steps(slider2, 6);
```

```
    slider_tooltip(slider1, "Horizontal Slider");
    slider_tooltip(slider2, "Horizontal Discrete Slider");
    slider_tooltip(slider3, "Vertical Slider");
    slider_OnMoved(slider1, listener(prog1, i_OnSlider, Progress));
    progress_undefined(prog2, TRUE);
    layout_label(layout2, label1, 0, 0);
    layout_label(layout2, label2, 0, 2);
    layout_label(layout2, label3, 0, 4);
    layout_label(layout2, label4, 0, 6);
    layout_slider(layout2, slider1, 0, 1);
    layout_slider(layout2, slider2, 0, 3);
    layout_slider(layout1, slider3, 1, 0);
    layout_progress(layout2, prog1, 0, 5);
    layout_progress(layout2, prog2, 0, 7);
    layout_hsize(layout2, 0, 300);
    layout_layout(layout1, layout2, 0, 0);
    layout_vmargin(layout2, 0, 5);
    layout_vmargin(layout2, 1, 5);
    layout_vmargin(layout2, 2, 5);
    layout_vmargin(layout2, 3, 5);
    layout_vmargin(layout2, 4, 5);
    layout_vmargin(layout2, 5, 5);
    layout_vmargin(layout2, 6, 5);
    layout_hmargin(layout1, 0, 10);
    panel_layout(panel, layout1);
    return panel;
}
```

26.7. ¡Hola TextView!

Listado 26.7: demo/guihello/textviews.c

```
/* Use of textviews */

#include "textviews.h"
#include "res_guihello.h"
#include <gui/guiall.h>

/*-----------------------------------------------------------------------------*/

static void i_set_rtf(TextView *text)
{
    ResPack *pack = res_guihello_respack("");
    uint32_t size = 0;
    const byte_t *data = respack_file(pack, TEXTVIEW_RTF, &size);
    Stream *stm = stm_from_block(data, size);
    textview_rtf(text, stm);
    stm_close(&stm);
    respack_destroy(&pack);
}
```

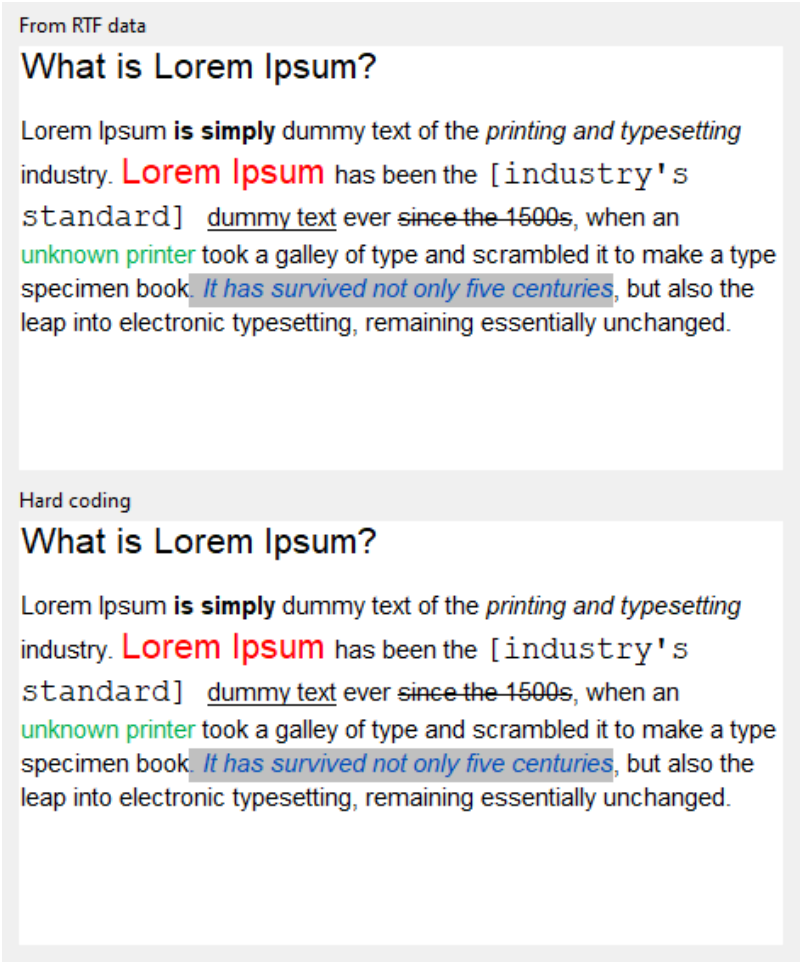


Figura 26.8: Control de texto enriquecido.

```
/*-----------------------------------------------------------------------------*/

static void i_set_hard_coding(TextView *text)
{
    textview_units(text, ekFPOINTS);
    textview_lspacing(text, 1.15f);
    textview_afspace(text, 10);
    textview_family(text, "Arial");
    textview_fsize(text, 16);
    textview_writef(text, "What is Lorem Ipsum?\n");
    textview_fsize(text, 11);
    textview_writef(text, "Lorem Ipsum ");
    textview_fstyle(text, ekFBOLD);
    textview_writef(text, "is simply");
```

```
    textview_fstyle(text, ekFNORMAL);
    textview_writef(text, " dummy text of the ");
    textview_fstyle(text, ekFITALIC);
    textview_writef(text, "printing and typesetting ");
    textview_fstyle(text, ekFNORMAL);
    textview_writef(text, "industry. ");
    textview_fsize(text, 16);
    textview_color(text, color_rgb(255, 0, 0));
    textview_writef(text, "Lorem Ipsum ");
    textview_color(text, kCOLOR_DEFAULT);
    textview_fsize(text, 11);
    textview_writef(text, "has been the ");
    textview_family(text, "Courier New");
    textview_fsize(text, 14);
    textview_writef(text, "[industry's standard] ");
    textview_family(text, "Arial");
    textview_fsize(text, 11);
    textview_fstyle(text, ekFUNDERLINE);
    textview_writef(text, "dummy text");
    textview_fstyle(text, ekFNORMAL);
    textview_writef(text, " ever ");
    textview_fstyle(text, ekFSTRIKEOUT);
    textview_writef(text, "since the 1500s");
    textview_fstyle(text, ekFNORMAL);
    textview_writef(text, ", when an ");
    textview_color(text, color_rgb(0, 176, 80));
    textview_writef(text, "unknown printer ");
    textview_color(text, kCOLOR_DEFAULT);
    textview_writef(text, "took a galley of type and scrambled it to make a
        ↪ type specimen book");
    textview_fstyle(text, ekFITALIC);
    textview_color(text, color_rgb(0, 77, 187));
    textview_bgcolor(text, color_rgb(192, 192, 192));
    textview_writef(text, ". It has survived not only five centuries");
    textview_fstyle(text, ekFNORMAL);
    textview_color(text, kCOLOR_DEFAULT);
    textview_bgcolor(text, kCOLOR_DEFAULT);
    textview_writef(text, ", but also the leap into electronic typesetting,
        ↪ remaining essentially unchanged.");
}

/*----------------------------------------------------------------------------*/

Panel *textviews(void)
{
    Layout *layout = layout_create(1, 4);
    Label *label1 = label_create();
    Label *label2 = label_create();
    TextView *text1 = textview_create();
    TextView *text2 = textview_create();
    Panel *panel = panel_create();
```

```
    label_text(label1, "From RTF data");
    label_text(label2, "Hard coding");
    textview_size(text1, s2df(450, 250));
    textview_size(text2, s2df(450, 250));
    i_set_rtf(text1);
    i_set_hard_coding(text2);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 2);
    layout_textview(layout, text1, 0, 1);
    layout_textview(layout, text2, 0, 3);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 10);
    layout_vmargin(layout, 2, 5);
    panel_layout(panel, layout);
    return panel;
}
```

26.8. ¡Hola TableView!

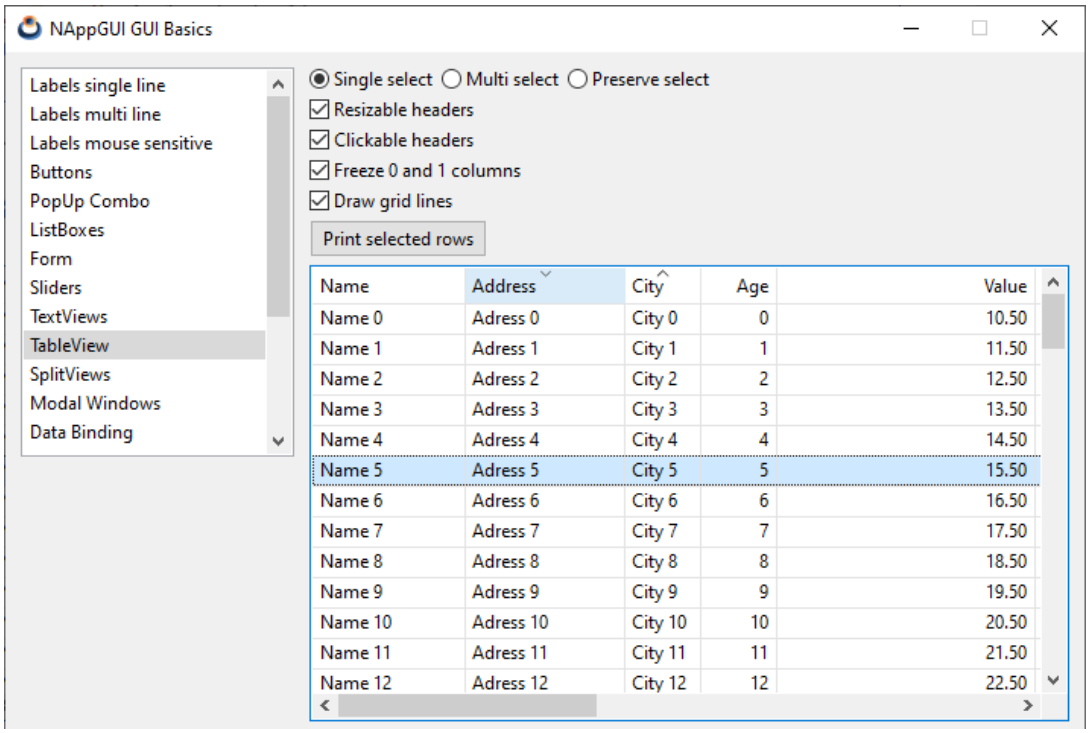


Figura 26.9: Control de tabla.

Listado 26.8: demo/guihello/table.c

```
/* Use of tables */
```

```
#include "table.h"
#include <gui/guiall.h>

typedef struct _appdata_t AppData;

struct _appdata_t
{
    TableView *table;
    TextView *text;
    char_t temp_string[256];
};

/*---------------------------------------------------------------------------*/

static void i_destroy_appdata(AppData** data)
{
    heap_delete(data, AppData);
}

/*---------------------------------------------------------------------------*/

/* AppData must contain the real data access(array, stream, etc) */
static void i_OnTableData(AppData *data, Event *e)
{
    uint32_t etype = event_type(e);

    switch(etype) {
    case ekGUI_EVENT_TBL_NROWS:
    {
        uint32_t *n = event_result(e, uint32_t);
        *n = 100;
        break;
    }

    case ekGUI_EVENT_TBL_CELL:
    {
        const EvTbPos *pos = event_params(e, EvTbPos);
        EvTbCell *cell = event_result(e, EvTbCell);

        switch(pos->col) {
        case 0:
            cell->align = ekLEFT;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Name %d
                ↪ ", pos->row);
            break;

        case 1:
            cell->align = ekLEFT;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Adress
                ↪ %d", pos->row);
```

```
            break;

        case 2:
            cell->align = ekLEFT;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "City %d
                ↪ ", pos->row);
            break;

        case 3:
            cell->align = ekRIGHT;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "%d",
                ↪ pos->row);
            break;

        case 4:
            cell->align = ekRIGHT;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "%.2f",
                ↪ 10.5f + pos->row);
            break;

        case 5:
            cell->align = ekCENTER;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Extra
                ↪ Data 1 %d", pos->row);
            break;

        case 6:
            cell->align = ekCENTER;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Extra
                ↪ Data 2 %d", pos->row);
            break;

        case 7:
            cell->align = ekCENTER;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Extra
                ↪ Data 3 %d", pos->row);
            break;

        case 8:
            cell->align = ekCENTER;
            bstd_sprintf(data->temp_string, sizeof(data->temp_string), "Extra
                ↪ Data 4 %d", pos->row);
            break;

        cassert_default();
        }

        cell->text = data->temp_string;
        break;
    }
    }
```

```
}

/*-----------------------------------------------------------------------------*/

static void i_OnHeaderClick(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    textview_printf(data->text, "Click on Header: %d\n", p->index);
}

/*-----------------------------------------------------------------------------*/

static void i_OnMultisel(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    if (p->index == 0)
        tableview_multisel(data->table, FALSE, FALSE);
    else if (p->index == 1)
        tableview_multisel(data->table, TRUE, FALSE);
    else if (p->index == 2)
        tableview_multisel(data->table, TRUE, TRUE);
}

/*-----------------------------------------------------------------------------*/

static void i_OnResizeCheck(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    bool_t resizable = p->state == ekGUI_ON ? TRUE : FALSE;
    tableview_header_resizable(data->table, resizable);
}

/*-----------------------------------------------------------------------------*/

static void i_OnHeaderCheck(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    bool_t clickable = p->state == ekGUI_ON ? TRUE : FALSE;
    tableview_header_clickable(data->table, clickable);
}

/*-----------------------------------------------------------------------------*/

static void i_OnFreezeCheck(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    uint32_t col_freeze = p->state == ekGUI_ON ? 1 : UINT32_MAX;
    tableview_column_freeze(data->table, col_freeze);
}

/*-----------------------------------------------------------------------------*/
```

```
static void i_OnGridCheck(AppData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    bool_t grid = p->state == ekGUI_ON ? TRUE : FALSE;
    tableview_grid(data->table, grid, grid);
}

/*-------------------------------------------------------------------------*/

static void i_OnPrintsel(AppData *data, Event *e)
{
    const ArrSt(uint32_t) *sel = tableview_selected(data->table);
    uint32_t n = arrst_size(sel, uint32_t);
    textview_writef(data->text, "Selected rows: ");
    arrst_foreach_const(row, sel, uint32_t)
        textview_printf(data->text, "%d", *row);
        if (row_i < n - 1)
            textview_writef(data->text, ", ");
    arrst_end();
    textview_writef(data->text, "\n");
    unref(e);
}

/*-------------------------------------------------------------------------*/

static Layout* i_table_control_layout(AppData *data)
{
    Layout *layout1 = layout_create(3, 1);
    Layout *layout2 = layout_create(1, 6);
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    Button *button3 = button_radio();
    Button *button4 = button_check();
    Button *button5 = button_check();
    Button *button6 = button_check();
    Button *button7 = button_check();
    Button *button8 = button_push();
    button_text(button1, "Single select");
    button_text(button2, "Multi select");
    button_text(button3, "Preserve select");
    button_text(button4, "Resizable headers");
    button_text(button5, "Clickable headers");
    button_text(button6, "Freeze 0 and 1 columns");
    button_text(button7, "Draw grid lines");
    button_text(button8, "Print selected rows");
    button_state(button1, ekGUI_ON);
    button_state(button4, ekGUI_ON);
    button_state(button5, ekGUI_ON);
    button_state(button6, ekGUI_ON);
    button_state(button7, ekGUI_ON);
```

```
    layout_button(layout1, button1, 0, 0);
    layout_button(layout1, button2, 1, 0);
    layout_button(layout1, button3, 2, 0);
    layout_layout(layout2, layout1, 0, 0);
    layout_button(layout2, button4, 0, 1);
    layout_button(layout2, button5, 0, 2);
    layout_button(layout2, button6, 0, 3);
    layout_button(layout2, button7, 0, 4);
    layout_button(layout2, button8, 0, 5);
    layout_hmargin(layout1, 0, 5.f);
    layout_hmargin(layout1, 1, 5.f);
    layout_vmargin(layout2, 0, 5.f);
    layout_vmargin(layout2, 1, 5.f);
    layout_vmargin(layout2, 2, 5.f);
    layout_vmargin(layout2, 3, 5.f);
    layout_vmargin(layout2, 4, 5.f);
    layout_halign(layout2, 0, 0, ekLEFT);
    layout_halign(layout2, 0, 5, ekLEFT);
    button_OnClick(button1, listener(data, i_OnMultisel, AppData));
    button_OnClick(button2, listener(data, i_OnMultisel, AppData));
    button_OnClick(button3, listener(data, i_OnMultisel, AppData));
    button_OnClick(button4, listener(data, i_OnResizeCheck, AppData));
    button_OnClick(button5, listener(data, i_OnHeaderCheck, AppData));
    button_OnClick(button6, listener(data, i_OnFreezeCheck, AppData));
    button_OnClick(button7, listener(data, i_OnGridCheck, AppData));
    button_OnClick(button8, listener(data, i_OnPrintsel, AppData));
    return layout2;
}

/*-----------------------------------------------------------------------------*/

Panel *table_view(void)
{
    Panel *panel = panel_create();
    AppData *data = heap_new0(AppData);
    TableView *table = tableview_create();
    TextView *text = textview_create();
    Layout *layout1 = layout_create(1, 3);
    Layout *layout2 = i_table_control_layout(data);
    data->table = table;
    data->text = text;
    tableview_size(table, s2df(500, 300));
    tableview_OnData(table, listener(data, i_OnTableData, AppData));
    tableview_OnHeaderClick(table, listener(data, i_OnHeaderClick, AppData));
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_new_column_text(table);
```

```
    tableview_new_column_text(table);
    tableview_new_column_text(table);
    tableview_header_clickable(table, TRUE);
    tableview_header_resizable(table, TRUE);
    tableview_header_indicator(table, 1, ekINDDOWN_ARROW);
    tableview_header_indicator(table, 2, ekINDUP_ARROW);
    tableview_header_title(table, 0, "Name");
    tableview_header_title(table, 1, "Address");
    tableview_header_title(table, 2, "City");
    tableview_header_title(table, 3, "Age");
    tableview_header_title(table, 4, "Value");
    tableview_header_title(table, 5, "Extra\nData 1");
    tableview_header_title(table, 6, "Extra\nData 2");
    tableview_header_title(table, 7, "Extra\nData 3");
    tableview_header_title(table, 8, "Extra\nData 4");
    tableview_column_width(table, 0, 100);
    tableview_column_width(table, 1, 105);
    tableview_column_width(table, 2, 50);
    tableview_column_width(table, 3, 50);
    tableview_column_width(table, 4, 170);
    tableview_column_width(table, 5, 200);
    tableview_column_width(table, 6, 200);
    tableview_column_width(table, 7, 200);
    tableview_column_width(table, 8, 200);
    tableview_column_limits(table, 2, 50, 100);
    tableview_column_freeze(table, 1);
    tableview_header_align(table, 0, ekLEFT);
    tableview_header_align(table, 1, ekLEFT);
    tableview_header_align(table, 2, ekLEFT);
    tableview_header_align(table, 3, ekRIGHT);
    tableview_header_align(table, 4, ekRIGHT);
    tableview_header_align(table, 5, ekCENTER);
    tableview_header_align(table, 6, ekCENTER);
    tableview_header_align(table, 7, ekCENTER);
    tableview_header_align(table, 8, ekCENTER);
    tableview_multisel(table, FALSE, FALSE);
    tableview_header_visible(table, TRUE);
    tableview_grid(table, TRUE, TRUE);
    tableview_update(table);

    {
        uint32_t row = 20;
        tableview_select(table, &row, 1);
        tableview_focus_row(table, row, ekBOTTOM);
    }

    layout_layout(layout1, layout2, 0, 0);
    layout_tableview(layout1, table, 0, 1);
    layout_textview(layout1, text, 0, 2);
    layout_vmargin(layout1, 0, 5.f);
    layout_vmargin(layout1, 1, 5.f);
```

```
    panel_data(panel, &data, i_destroy_appdata, AppData);
    panel_layout(panel, layout1);
    return panel;
}
```

26.9. ¡Hola SplitView!

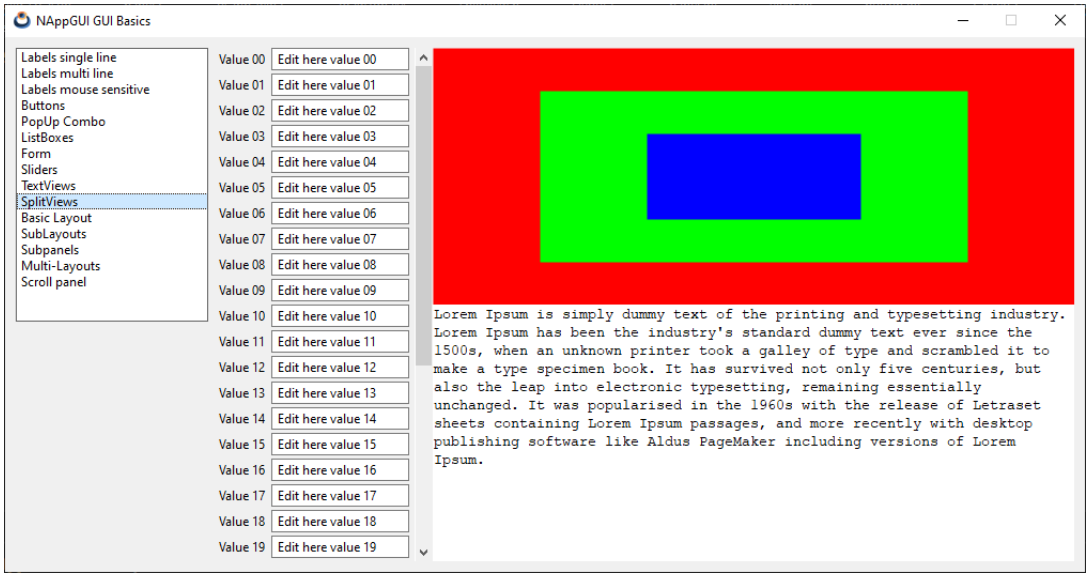


Figura 26.10: SplitView.

Listado 26.9: demo/guihello/splits.c

```
/* Use of splitviews */

#include "splits.h"
#include <gui/guiall.h>

static const char_t *i_LOREM = "Lorem Ipsum is simply dummy text of the
    ↪ printing and typesetting industry. Lorem Ipsum has been the industry's
    ↪ standard dummy text ever since the 1500s, when an unknown printer took a
    ↪  galley of type and scrambled it to make a type specimen book. It has
    ↪ survived not only five centuries, but also the leap into electronic
    ↪ typesetting, remaining essentially unchanged. It was popularised in the
    ↪ 1960s with the release of Letraset sheets containing Lorem Ipsum
    ↪ passages, and more recently with desktop publishing software like Aldus
    ↪ PageMaker including versions of Lorem Ipsum.";

/*---------------------------------------------------------------------------*/

static void i_OnDraw(View *view, Event *e)
```

```
{
    const EvDraw *p = event_params(e, EvDraw);
    real32_t p0 = p->width / 6;
    real32_t p1 = p->height / 6;
    real32_t p2 = p->width / 3;
    real32_t p3 = p->height / 3;
    unref(view);
    draw_fill_color(p->ctx, kCOLOR_RED);
    draw_rect(p->ctx, ekFILL, 0, 0, p->width, p->height);
    draw_fill_color(p->ctx, kCOLOR_GREEN);
    draw_rect(p->ctx, ekFILL, p0, p1, p->width - 2 * p0, p->height - 2 * p1);
    draw_fill_color(p->ctx, kCOLOR_BLUE);
    draw_rect(p->ctx, ekFILL, p2, p3, p->width - 2 * p2, p->height - 2 * p3);
}

/*---------------------------------------------------------------------------*/

static Panel *i_left_panel(void)
{
    uint32_t i, n = 32;
    Panel *panel = panel_scroll(FALSE, TRUE);
    Layout *layout = layout_create(2, n);
    real32_t rmargin = panel_scroll_width(panel);

    for (i = 0; i < n; ++i)
    {
        char_t text[64];
        Label *label = label_create();
        Edit *edit = edit_create();
        bstd_sprintf(text, sizeof(text), "Value %02d", i);
        label_text(label, text);
        bstd_sprintf(text, sizeof(text), "Edit here value %02d", i);
        edit_text(edit, text);
        layout_label(layout, label, 0, i);
        layout_edit(layout, edit, 1, i);
    }

    for (i = 0; i < n - 1; ++i)
        layout_vmargin(layout, i, 3);

    layout_hmargin(layout, 0, 5);
    layout_margin4(layout, 0, rmargin + 5, 0, 0);
    layout_hexpand(layout, 1);
    panel_layout(panel, layout);
    return panel;
}

/*---------------------------------------------------------------------------*/

Panel *split_panel(void)
{
```

```
    Panel *panel1 = panel_create();
    Panel *panel2 = i_left_panel();
    Layout *layout = layout_create(1, 1);
    SplitView *split1 = splitview_vertical();
    SplitView *split2 = splitview_horizontal();
    TextView *text = textview_create();
    View *view = view_create();
    textview_writef(text, i_LOREM);
    view_OnDraw(view, listener(view, i_OnDraw, View));
    splitview_pos(split1, .25f);
    splitview_size(split1, s2df(800, 480));
    splitview_size(split2, s2df(640, 480));
    splitview_view(split2, view);
    splitview_text(split2, text);
    splitview_panel(split1, panel2);
    splitview_split(split1, split2);
    layout_splitview(layout, split1, 0, 0);
    panel_layout(panel1, layout);
    return panel1;
}
```

26.10. ¡Hola Modal Window!

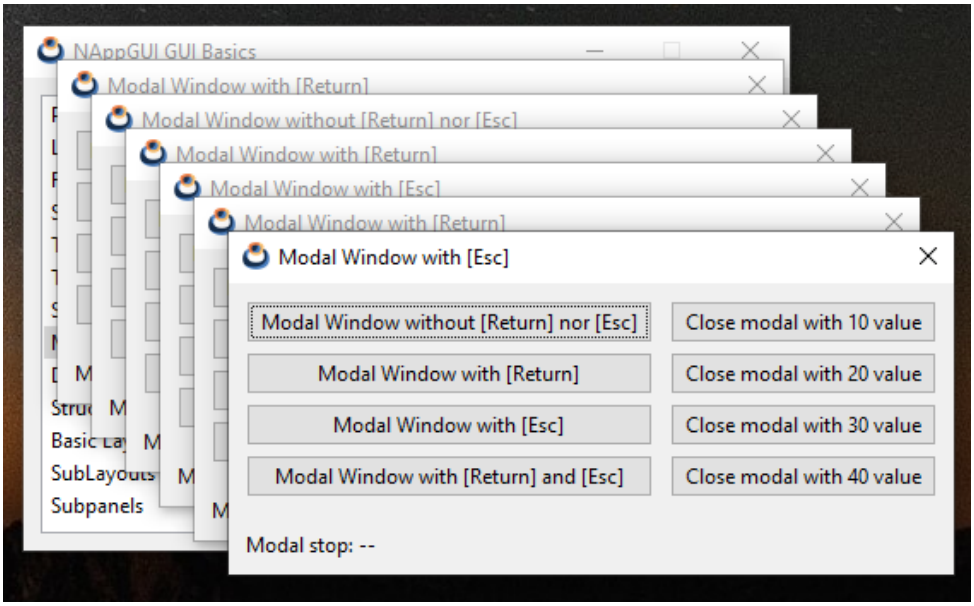


Figura 26.11: Ventanas modales.

Listado 26.10: demo/guihello/modalwin.c

```
/* Listboxes */
```

```
#include "modalwin.h"
#include <gui/guiall.h>

typedef struct _modal_data_t ModalData;

struct _modal_data_t
{
    uint32_t type;
    Label *label;
    Window *parent;
};

/*-------------------------------------------------------------------------*/

static const char_t *i_MODAL0 = "Modal Window without [Return] nor [Esc]";
static const char_t *i_MODAL1 = "Modal Window with [Return]";
static const char_t *i_MODAL2 = "Modal Window with [Esc]";
static const char_t *i_MODAL3 = "Modal Window with [Return] and [Esc]";

/*-------------------------------------------------------------------------*/

static Layout *i_modal_layout(ModalData *data);

/*-------------------------------------------------------------------------*/

static ModalData* i_modal_data(Window* parent)
{
    ModalData *data = heap_new0(ModalData);
    data->parent = parent;
    data->type = UINT32_MAX;
    return data;
}

/*-------------------------------------------------------------------------*/

static void i_destroy_modal_data(ModalData** data)
{
    heap_delete(data, ModalData);
}

/*-------------------------------------------------------------------------*/

static void i_OnCloseModal(Window* window, Event* e)
{
    Button *button = event_sender(e, Button);
    window_stop_modal(window, button_get_tag(button));
}

/*-------------------------------------------------------------------------*/
```

```
static Layout* i_close_layout(Window *window)
{
    Layout *layout = layout_create(1, 4);
    Button *button1 = button_push();
    Button *button2 = button_push();
    Button *button3 = button_push();
    Button *button4 = button_push();
    button_text(button1, "Close modal with 10 value");
    button_text(button2, "Close modal with 20 value");
    button_text(button3, "Close modal with 30 value");
    button_text(button4, "Close modal with 40 value");
    button_tag(button1, 10);
    button_tag(button2, 20);
    button_tag(button3, 30);
    button_tag(button4, 40);
    button_OnClick(button1, listener(window, i_OnCloseModal, Window));
    button_OnClick(button2, listener(window, i_OnCloseModal, Window));
    button_OnClick(button3, listener(window, i_OnCloseModal, Window));
    button_OnClick(button4, listener(window, i_OnCloseModal, Window));
    layout_button(layout, button1, 0, 0);
    layout_button(layout, button2, 0, 1);
    layout_button(layout, button3, 0, 2);
    layout_button(layout, button4, 0, 3);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_vmargin(layout, 2, 5);
    return layout;
}

/*--------------------------------------------------------------------------*/

static uint32_t i_window_flags(const uint32_t type)
{
    uint32_t flags = ekWINDOW_TITLE | ekWINDOW_CLOSE;
    switch(type) {
    case 0:
        return flags;
    case 1:
        return flags | ekWINDOW_RETURN;
    case 2:
        return flags | ekWINDOW_ESC;
    case 3:
        return flags | ekWINDOW_RETURN | ekWINDOW_ESC;
    cassert_default();
    }

    return 0;
}

/*--------------------------------------------------------------------------*/
```

```
static const char_t *i_window_title(const uint32_t type)
{
    switch(type) {
    case 0:
        return i_MODAL0;
    case 1:
        return i_MODAL1;
    case 2:
        return i_MODAL2;
    case 3:
        return i_MODAL3;
    cassert_default();
    }

    return 0;
}

/*---------------------------------------------------------------------------*/

static void i_modal_window(ModalData *data)
{
    uint32_t flags = i_window_flags(data->type);
    Window *window = window_create(flags);
    ModalData *ndata = i_modal_data(window);
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(2, 1);
    Layout *layout2 = i_modal_layout(ndata);
    Layout *layout3 = i_close_layout(window);
    uint32_t retval = UINT32_MAX;
    V2Df pos = window_get_origin(data->parent);
    char_t text[128];
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 1, 0);
    layout_hmargin(layout1, 0, 10);
    layout_valign(layout1, 1, 0, ekTOP);
    layout_margin(layout1, 10);
    panel_data(panel, &ndata, i_destroy_modal_data, ModalData);
    panel_layout(panel, layout1);
    window_panel(window, panel);
    window_title(window, i_window_title(data->type));
    window_origin(window, v2df(pos.x + 20, pos.y + 20));
    retval = window_modal(window, data->parent);

    if (retval == (uint32_t)ekGUI_CLOSE_ESC)
        bstd_sprintf(text, sizeof(text), "Modal stop: [Esc] (%d)", retval);
    else if (retval == (uint32_t)ekGUI_CLOSE_INTRO)
        bstd_sprintf(text, sizeof(text), "Modal stop: [Return] (%d)", retval);
    else if (retval == (uint32_t)ekGUI_CLOSE_BUTTON)
        bstd_sprintf(text, sizeof(text), "Modal stop: [X] (%d)", retval);
    else
        bstd_sprintf(text, sizeof(text), "Modal stop: %d", retval);
```

```
    label_text(data->label, text);
    window_destroy(&window);
}

/*-------------------------------------------------------------------------*/

static void i_OnClickModal(ModalData* data, Event* e)
{
    Button *button = event_sender(e, Button);
    data->type = button_get_tag(button);
    i_modal_window(data);
}

/*-------------------------------------------------------------------------*/

static Layout *i_modal_layout(ModalData *data)
{
    Layout *layout = layout_create(1, 5);
    Button *button1 = button_push();
    Button *button2 = button_push();
    Button *button3 = button_push();
    Button *button4 = button_push();
    Label *label = label_create();
    cassert(data->label == NULL);
    data->label = label;
    button_text(button1, i_MODAL0);
    button_text(button2, i_MODAL1);
    button_text(button3, i_MODAL2);
    button_text(button4, i_MODAL3);
    label_text(label, "Modal stop: --");
    button_tag(button1, 0);
    button_tag(button2, 1);
    button_tag(button3, 2);
    button_tag(button4, 3);
    button_OnClick(button1, listener(data, i_OnClickModal, ModalData));
    button_OnClick(button2, listener(data, i_OnClickModal, ModalData));
    button_OnClick(button3, listener(data, i_OnClickModal, ModalData));
    button_OnClick(button4, listener(data, i_OnClickModal, ModalData));
    layout_button(layout, button1, 0, 0);
    layout_button(layout, button2, 0, 1);
    layout_button(layout, button3, 0, 2);
    layout_button(layout, button4, 0, 3);
    layout_label(layout, label, 0, 4);
    layout_halign(layout, 0, 4, ekJUSTIFY);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_vmargin(layout, 2, 5);
    layout_vmargin(layout, 3, 20);
    return layout;
}
```

```
/*---------------------------------------------------------------------------*/

Panel *modal_windows(Window *parent)
{
    Panel *panel = panel_create();
    ModalData *data = i_modal_data(parent);
    Layout *layout = i_modal_layout(data);
    panel_layout(panel, layout);
    panel_data(panel, &data, i_destroy_modal_data, ModalData);
    return panel;
}
```

26.11. ¡Hola Gui Binding!

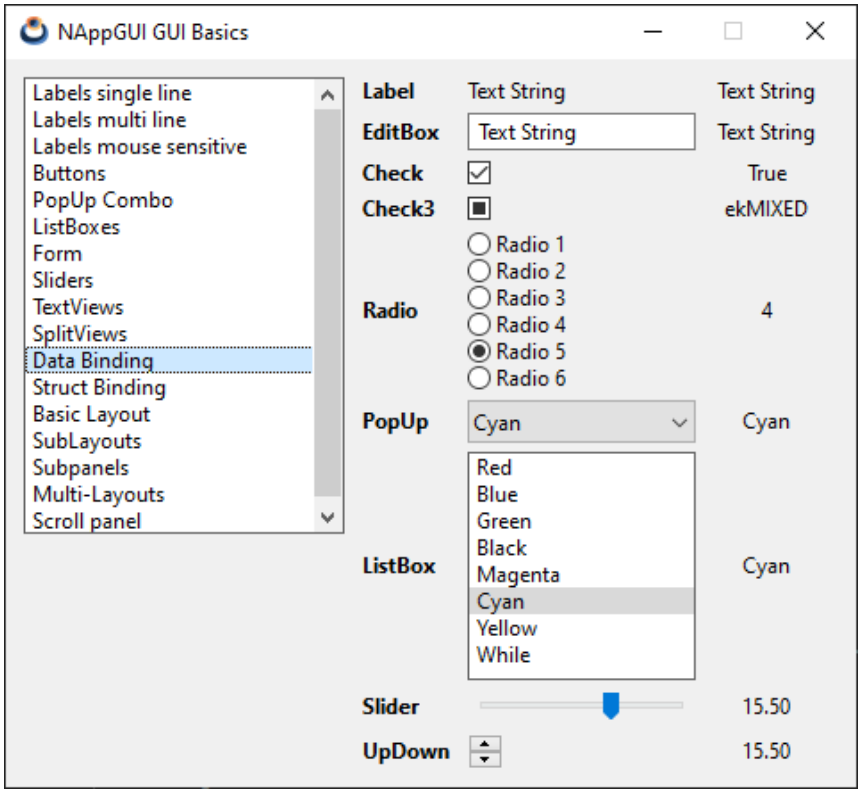


Figura 26.12: Gui Data binding.

Listado 26.11: demo/guihello/guibind.c

```
/* GUI data binding */

#include "guibind.h"
```

```
#include <gui/guiall.h>

typedef struct _basictypes_t BasicTypes;

typedef enum _myenum_t
{
    ekRED,
    ekBLUE,
    ekGREEN,
    ekBLACK,
    ekMAGENTA,
    ekCYAN,
    ekYELLOW,
    ekWHITE
} myenum_t;

struct _basictypes_t
{
    bool_t bool_val;
    uint16_t uint16_val;
    real32_t real32_val;
    myenum_t enum_val;
    gui_state_t enum3_val;
    String* str_val;
};

#define i_NUM_CONTROLS 9

/*---------------------------------------------------------------------------*/

static void i_destroy_data(BasicTypes **data)
{
    str_destroy(&(*data)->str_val);
    heap_delete(data, BasicTypes);
}

/*---------------------------------------------------------------------------*/

static Layout *i_radio_layout(void)
{
    uint32_t i = 0, n = 6;
    Layout *layout = layout_create(1, n);
    for (i = 0; i < n; ++i)
    {
        Button *radio = button_radio();
        char_t str[64];
        bstd_sprintf(str, sizeof(str), "Radio %d", i + 1);
        button_text(radio, str);
        layout_button(layout, radio, 0, i);
    }
```

```
    return layout;
}

/*----------------------------------------------------------------------------*/

static void i_title_labels(Layout* layout)
{
    Font* font = font_system(font_regular_size(), ekFBOLD);
    const char_t* strs[] = { "Label", "EditBox", "Check", "Check3", "Radio", "
        ↪ PopUp", "ListBox", "Slider", "UpDown" };
    uint32_t i = 0;
    for (i = 0; i < i_NUM_CONTROLS; ++i)
    {
        Label* label = label_create();
        label_text(label, strs[i]);
        label_font(label, font);
        layout_label(layout, label, 0, i);
    }

    layout_hmargin(layout, 0, 10);
    font_destroy(&font);
}

/*----------------------------------------------------------------------------*/

static void i_value_labels(Layout* layout)
{
    uint32_t i = 0;
    for (i = 0; i < i_NUM_CONTROLS; ++i)
    {
        Label* label = label_create();
        label_align(label, ekCENTER);
        layout_label(layout, label, 2, i);
        layout_halign(layout, 2, i, ekJUSTIFY);
    }

    layout_hsize(layout, 2, 80);
    layout_hmargin(layout, 0, 10);
    for (i = 0; i < i_NUM_CONTROLS - 1; ++i)
        layout_vmargin(layout, i, 5);

    cell_dbind(layout_cell(layout, 2, 0), BasicTypes, String*, str_val);
    cell_dbind(layout_cell(layout, 2, 1), BasicTypes, String*, str_val);
    cell_dbind(layout_cell(layout, 2, 2), BasicTypes, bool_t, bool_val);
    cell_dbind(layout_cell(layout, 2, 3), BasicTypes, gui_state_t, enum3_val);
    cell_dbind(layout_cell(layout, 2, 4), BasicTypes, uint16_t, uint16_val);
    cell_dbind(layout_cell(layout, 2, 5), BasicTypes, myenum_t, enum_val);
    cell_dbind(layout_cell(layout, 2, 6), BasicTypes, myenum_t, enum_val);
    cell_dbind(layout_cell(layout, 2, 7), BasicTypes, real32_t, real32_val);
    cell_dbind(layout_cell(layout, 2, 8), BasicTypes, real32_t, real32_val);
}
```

```
/*-------------------------------------------------------------------------*/

static Layout *i_layout(void)
{
    Layout *layout = layout_create(3, 9);
    Label *label = label_create();
    Edit *edit = edit_create();
    Button *check = button_check();
    Button *check3 = button_check3();
    Layout *radios = i_radio_layout();
    PopUp *popup = popup_create();
    ListBox *listbox = listbox_create();
    Slider *slider = slider_create();
    UpDown *updown = updown_create();
    layout_label(layout, label, 1, 0);
    layout_edit(layout, edit, 1, 1);
    layout_button(layout, check, 1, 2);
    layout_button(layout, check3, 1, 3);
    layout_layout(layout, radios, 1, 4);
    layout_popup(layout, popup, 1, 5);
    layout_listbox(layout, listbox, 1, 6);
    layout_slider(layout, slider, 1, 7);
    layout_updown(layout, updown, 1, 8);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_halign(layout, 1, 8, ekLEFT);
    cell_dbind(layout_cell(layout, 1, 0), BasicTypes, String*, str_val);
    cell_dbind(layout_cell(layout, 1, 1), BasicTypes, String*, str_val);
    cell_dbind(layout_cell(layout, 1, 2), BasicTypes, bool_t, bool_val);
    cell_dbind(layout_cell(layout, 1, 3), BasicTypes, gui_state_t, enum3_val);
    cell_dbind(layout_cell(layout, 1, 4), BasicTypes, uint16_t, uint16_val);
    cell_dbind(layout_cell(layout, 1, 5), BasicTypes, myenum_t, enum_val);
    cell_dbind(layout_cell(layout, 1, 6), BasicTypes, myenum_t, enum_val);
    cell_dbind(layout_cell(layout, 1, 7), BasicTypes, real32_t, real32_val);
    cell_dbind(layout_cell(layout, 1, 8), BasicTypes, real32_t, real32_val);
    i_title_labels(layout);
    i_value_labels(layout);
    return layout;
}

/*-------------------------------------------------------------------------*/

Panel* guibind(void)
{
    Layout *layout = NULL;
    Panel *panel = NULL;
    BasicTypes *data = heap_new(BasicTypes);
    data->bool_val = TRUE;
    data->uint16_val = 4;
    data->real32_val = 15.5f;
    data->enum3_val = ekGUI_MIXED;
```

```
    data->enum_val = ekCYAN;
    data->str_val = str_c("Text String");

    dbind_enum(gui_state_t, ekGUI_OFF, "");
    dbind_enum(gui_state_t, ekGUI_ON, "");
    dbind_enum(gui_state_t, ekGUI_MIXED, "");
    dbind_enum(myenum_t, ekRED, "Red");
    dbind_enum(myenum_t, ekBLUE, "Blue");
    dbind_enum(myenum_t, ekGREEN, "Green");
    dbind_enum(myenum_t, ekBLACK, "Black");
    dbind_enum(myenum_t, ekMAGENTA, "Magenta");
    dbind_enum(myenum_t, ekCYAN, "Cyan");
    dbind_enum(myenum_t, ekYELLOW, "Yellow");
    dbind_enum(myenum_t, ekWHITE, "While");
    dbind(BasicTypes, bool_t, bool_val);
    dbind(BasicTypes, uint16_t, uint16_val);
    dbind(BasicTypes, real32_t, real32_val);
    dbind(BasicTypes, gui_state_t, enum3_val);
    dbind(BasicTypes, myenum_t, enum_val);
    dbind(BasicTypes, String*, str_val);
    dbind_range(BasicTypes, real32_t, real32_val, -50, 50);
    dbind_increment(BasicTypes, real32_t, real32_val, 5);

    layout = i_layout();
    panel = panel_create();
    layout_dbind(layout, NULL, BasicTypes);
    layout_dbind_obj(layout, data, BasicTypes);
    panel_data(panel, &data, i_destroy_data, BasicTypes);
    panel_layout(panel, layout);
    return panel;
}
```

26.12. ¡Hola Struct Binding!

Listado 26.12: demo/guihello/layoutbind.c

```
/* GUI data binding */

#include "layoutbind.h"
#include <gui/guiall.h>

typedef struct _vector_t Vector;
typedef struct _structtypes_t StructTypes;

struct _vector_t
{
    real32_t x;
    real32_t y;
    real32_t z;
};
```

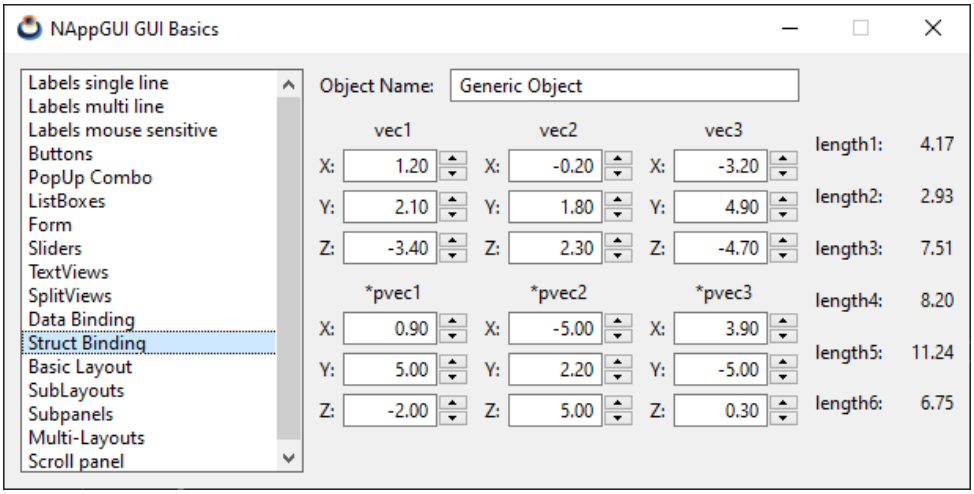


Figura 26.13: Gui Struct binding.

```
struct _structtypes_t
{
    String *name;
    Vector vec1;
    Vector vec2;
    Vector vec3;
    Vector *pvec1;
    Vector *pvec2;
    Vector *pvec3;
    real32_t length1;
    real32_t length2;
    real32_t length3;
    real32_t length4;
    real32_t length5;
    real32_t length6;
};

/*---------------------------------------------------------------------------*/

static void i_destroy_data(StructTypes **data)
{
    str_destroy(&(*data)->name);
    heap_delete(&(*data)->pvec1, Vector);
    heap_delete(&(*data)->pvec2, Vector);
    heap_delete(&(*data)->pvec3, Vector);
    heap_delete(data, StructTypes);
}

/*---------------------------------------------------------------------------*/
```

```
static Vector i_vec_init(const real32_t x, const real32_t y, const real32_t z)
{
    Vector v;
    v.x = x;
    v.y = y;
    v.z = z;
    return v;
}

/*-----------------------------------------------------------------------------*/

static real32_t i_vec_length(const Vector *vec)
{
    real32_t n = vec->x * vec->x + vec->y * vec->y + vec->z * vec->z;
    return bmath_sqrtf(n);
}

/*-----------------------------------------------------------------------------*/

static void i_OnDataChange(void *non_used, Event *e)
{
    StructTypes *data = evbind_object(e, StructTypes);
    Layout *layout = event_sender(e, Layout);
    unref(non_used);

    if (evbind_modify(e, StructTypes, Vector, vec1) == TRUE)
    {
        data->length1 = i_vec_length(&data->vec1);
        layout_dbind_update(layout, StructTypes, real32_t, length1);
    }
    else if (evbind_modify(e, StructTypes, Vector, vec2) == TRUE)
    {
        data->length2 = i_vec_length(&data->vec2);
        layout_dbind_update(layout, StructTypes, real32_t, length2);
    }
    else if (evbind_modify(e, StructTypes, Vector, vec3) == TRUE)
    {
        data->length3 = i_vec_length(&data->vec3);
        layout_dbind_update(layout, StructTypes, real32_t, length3);
    }
    else if (evbind_modify(e, StructTypes, Vector*, pvec1) == TRUE)
    {
        data->length4 = i_vec_length(data->pvec1);
        layout_dbind_update(layout, StructTypes, real32_t, length4);
    }
    else if (evbind_modify(e, StructTypes, Vector*, pvec2) == TRUE)
    {
        data->length5 = i_vec_length(data->pvec2);
        layout_dbind_update(layout, StructTypes, real32_t, length5);
    }
    else if (evbind_modify(e, StructTypes, Vector*, pvec3) == TRUE)
```

```
    {
        data->length6 = i_vec_length(data->pvec3);
        layout_dbind_update(layout, StructTypes, real32_t, length6);
    }
}

/*----------------------------------------------------------------------------*/

static Layout *i_vector_layout(void)
{
    Layout *layout = layout_create(3, 3);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    Edit *edit3 = edit_create();
    UpDown *updown1 = updown_create();
    UpDown *updown2 = updown_create();
    UpDown *updown3 = updown_create();
    label_text(label1, "X:");
    label_text(label2, "Y:");
    label_text(label3, "Z:");
    edit_align(edit1, ekRIGHT);
    edit_align(edit2, ekRIGHT);
    edit_align(edit3, ekRIGHT);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 2);
    layout_edit(layout, edit1, 1, 0);
    layout_edit(layout, edit2, 1, 1);
    layout_edit(layout, edit3, 1, 2);
    layout_updown(layout, updown1, 2, 0);
    layout_updown(layout, updown2, 2, 1);
    layout_updown(layout, updown3, 2, 2);
    layout_hmargin(layout, 0, 5);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_hsize(layout, 1, 60);
    cell_dbind(layout_cell(layout, 1, 0), Vector, real32_t, x);
    cell_dbind(layout_cell(layout, 1, 1), Vector, real32_t, y);
    cell_dbind(layout_cell(layout, 1, 2), Vector, real32_t, z);
    cell_dbind(layout_cell(layout, 2, 0), Vector, real32_t, x);
    cell_dbind(layout_cell(layout, 2, 1), Vector, real32_t, y);
    cell_dbind(layout_cell(layout, 2, 2), Vector, real32_t, z);
    layout_dbind(layout, NULL, Vector);
    return layout;
}

/*----------------------------------------------------------------------------*/
```

```
static Layout *i_name_layout(void)
{
    Layout *layout = layout_create(2, 1);
    Label *label = label_create();
    Edit *edit = edit_create();
    label_text(label, "Object Name:");
    layout_hexpand(layout, 1);
    layout_label(layout, label, 0, 0);
    layout_edit(layout, edit, 1, 0);
    layout_hmargin(layout, 0, 10);
    cell_dbind(layout_cell(layout, 1, 0), StructTypes, String*, name);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_vectors_layout(void)
{
    Layout *layout1 = layout_create(3, 4);
    Layout *layout2 = i_vector_layout();
    Layout *layout3 = i_vector_layout();
    Layout *layout4 = i_vector_layout();
    Layout *layout5 = i_vector_layout();
    Layout *layout6 = i_vector_layout();
    Layout *layout7 = i_vector_layout();
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_create();
    label_text(label1, "vec1");
    label_text(label2, "vec2");
    label_text(label3, "vec3");
    label_text(label4, "*pvec1");
    label_text(label5, "*pvec2");
    label_text(label6, "*pvec3");
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 1, 0);
    layout_label(layout1, label3, 2, 0);
    layout_label(layout1, label4, 0, 2);
    layout_label(layout1, label5, 1, 2);
    layout_label(layout1, label6, 2, 2);
    layout_layout(layout1, layout2, 0, 1);
    layout_layout(layout1, layout3, 1, 1);
    layout_layout(layout1, layout4, 2, 1);
    layout_layout(layout1, layout5, 0, 3);
    layout_layout(layout1, layout6, 1, 3);
    layout_layout(layout1, layout7, 2, 3);
    layout_halign(layout1, 0, 0, ekCENTER);
    layout_halign(layout1, 1, 0, ekCENTER);
```

```
    layout_halign(layout1, 2, 0, ekCENTER);
    layout_halign(layout1, 0, 2, ekCENTER);
    layout_halign(layout1, 1, 2, ekCENTER);
    layout_halign(layout1, 2, 2, ekCENTER);
    layout_hmargin(layout1, 0, 10);
    layout_hmargin(layout1, 1, 10);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 1, 10);
    layout_vmargin(layout1, 2, 5);
    cell_dbind(layout_cell(layout1, 0, 1), StructTypes, Vector, vec1);
    cell_dbind(layout_cell(layout1, 1, 1), StructTypes, Vector, vec2);
    cell_dbind(layout_cell(layout1, 2, 1), StructTypes, Vector, vec3);
    cell_dbind(layout_cell(layout1, 0, 3), StructTypes, Vector*, pvec1);
    cell_dbind(layout_cell(layout1, 1, 3), StructTypes, Vector*, pvec2);
    cell_dbind(layout_cell(layout1, 2, 3), StructTypes, Vector*, pvec3);
    return layout1;
}

/*--------------------------------------------------------------------------*/

static Layout *i_lengths_layout(void)
{
    Layout *layout = layout_create(2, 6);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_create();
    Label *label7 = label_create();
    Label *label8 = label_create();
    Label *label9 = label_create();
    Label *label10 = label_create();
    Label *label11 = label_create();
    Label *label12 = label_create();
    label_text(label1, "length1:");
    label_text(label2, "length2:");
    label_text(label3, "length3:");
    label_text(label4, "length4:");
    label_text(label5, "length5:");
    label_text(label6, "length6:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 2);
    layout_label(layout, label4, 0, 3);
    layout_label(layout, label5, 0, 4);
    layout_label(layout, label6, 0, 5);
    layout_label(layout, label7, 1, 0);
    layout_label(layout, label8, 1, 1);
    layout_label(layout, label9, 1, 2);
    layout_label(layout, label10, 1, 3);
```

```
    layout_label(layout, label11, 1, 4);
    layout_label(layout, label12, 1, 5);
    label_align(label7, ekRIGHT);
    label_align(label8, ekRIGHT);
    label_align(label9, ekRIGHT);
    label_align(label10, ekRIGHT);
    label_align(label11, ekRIGHT);
    label_align(label12, ekRIGHT);
    layout_hsize(layout, 1, 40);
    layout_hmargin(layout, 0, 5);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_halign(layout, 1, 1, ekJUSTIFY);
    layout_halign(layout, 1, 2, ekJUSTIFY);
    layout_halign(layout, 1, 3, ekJUSTIFY);
    layout_halign(layout, 1, 4, ekJUSTIFY);
    layout_halign(layout, 1, 5, ekJUSTIFY);
    cell_dbind(layout_cell(layout, 1, 0), StructTypes, real32_t, length1);
    cell_dbind(layout_cell(layout, 1, 1), StructTypes, real32_t, length2);
    cell_dbind(layout_cell(layout, 1, 2), StructTypes, real32_t, length3);
    cell_dbind(layout_cell(layout, 1, 3), StructTypes, real32_t, length4);
    cell_dbind(layout_cell(layout, 1, 4), StructTypes, real32_t, length5);
    cell_dbind(layout_cell(layout, 1, 5), StructTypes, real32_t, length6);
    return layout;
}

/*-------------------------------------------------------------------------*/

static Layout *i_layout(void)
{
    Layout *layout1 = layout_create(2, 2);
    Layout *layout2 = i_name_layout();
    Layout *layout3 = i_vectors_layout();
    Layout *layout4 = i_lengths_layout();
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 0, 1);
    layout_layout(layout1, layout4, 1, 1);
    layout_hmargin(layout1, 0, 10);
    layout_vmargin(layout1, 0, 10);
    return layout1;
}

/*-------------------------------------------------------------------------*/

Panel* layoutbind(void)
{
    Layout *layout = NULL;
    Panel *panel = NULL;
    StructTypes *data = heap_new(StructTypes);
    data->name = str_c("Generic Object");
    data->pvec1 = heap_new(Vector);
    data->pvec2 = heap_new(Vector);
```

```
    data->pvec3 = heap_new(Vector);
    data->vec1 = i_vec_init(1.2f, 2.1f, -3.4f);
    data->vec2 = i_vec_init(-0.2f, 1.8f, 2.3f);
    data->vec3 = i_vec_init(-3.2f, 4.9f, -4.7f);
    *data->pvec1 = i_vec_init(0.9f, 7.9f, -2.0f);
    *data->pvec2 = i_vec_init(-6.9f, 2.2f, 8.6f);
    *data->pvec3 = i_vec_init(3.9f, -5.5f, 0.3f);
    data->length1 = i_vec_length(&data->vec1);
    data->length2 = i_vec_length(&data->vec2);
    data->length3 = i_vec_length(&data->vec3);
    data->length4 = i_vec_length(data->pvec1);
    data->length5 = i_vec_length(data->pvec2);
    data->length6 = i_vec_length(data->pvec3);

    dbind(Vector, real32_t, x);
    dbind(Vector, real32_t, y);
    dbind(Vector, real32_t, z);
    dbind(StructTypes, String*, name);
    dbind(StructTypes, Vector, vec1);
    dbind(StructTypes, Vector, vec2);
    dbind(StructTypes, Vector, vec3);
    dbind(StructTypes, Vector*, pvec1);
    dbind(StructTypes, Vector*, pvec2);
    dbind(StructTypes, Vector*, pvec3);
    dbind(StructTypes, real32_t, length1);
    dbind(StructTypes, real32_t, length2);
    dbind(StructTypes, real32_t, length3);
    dbind(StructTypes, real32_t, length4);
    dbind(StructTypes, real32_t, length5);
    dbind(StructTypes, real32_t, length6);
    dbind_range(Vector, real32_t, x, -5, 5);
    dbind_range(Vector, real32_t, y, -5, 5);
    dbind_range(Vector, real32_t, z, -5, 5);
    dbind_increment(Vector, real32_t, x, .1f);
    dbind_increment(Vector, real32_t, y, .1f);
    dbind_increment(Vector, real32_t, z, .1f);

    layout = i_layout();
    panel = panel_create();
    layout_dbind(layout, listener(NULL, i_OnDataChange, void), StructTypes);
    layout_dbind_obj(layout, data, StructTypes);
    panel_data(panel, &data, i_destroy_data, StructTypes);
    panel_layout(panel, layout);
    return panel;
}
```

26.13. ¡Hola Sublayout!

Listado 26.13: demo/guihello/sublayout.c

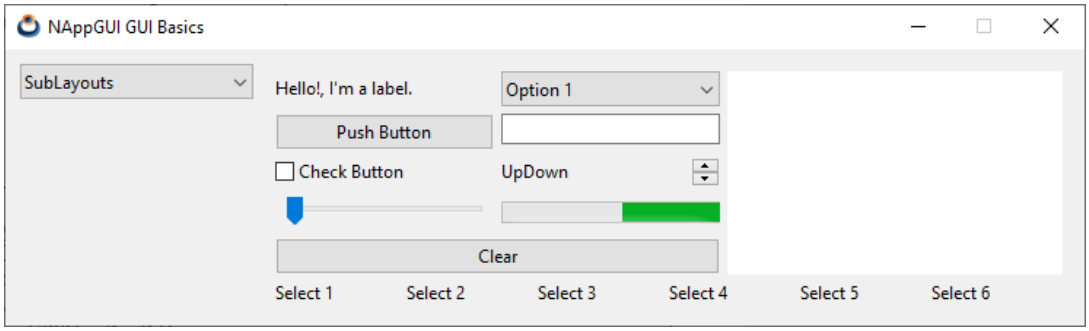


Figura 26.14: Composición de Sublayouts.

```
/* Sublayouts */

#include "sublayout.h"
#include <gui/guiall.h>

/*---------------------------------------------------------------------------*/

static Layout *i_updown_layout(void)
{
    Layout *layout = layout_create(2, 1);
    Label *label = label_create();
    UpDown *updown = updown_create();
    label_text(label, "UpDown");
    layout_label(layout, label, 0, 0);
    layout_updown(layout, updown, 1, 0);
    layout_hexpand(layout, 0);
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_left_grid_layout(void)
{
    Layout *layout1 = layout_create(2, 4);
    Layout *layout2 = i_updown_layout();
    Label *label = label_create();
    Button *button1 = button_push();
    Button *button2 = button_check();
    Slider *slider = slider_create();
    PopUp *popup = popup_create();
    Edit *edit = edit_create();
    Progress *progress = progress_create();
    label_text(label, "Hello!, I'm a label.");
    button_text(button1, "Push Button");
    button_text(button2, "Check Button");
    popup_add_elem(popup, "Option 1", NULL);
```

```
    popup_add_elem(popup, "Option 2", NULL);
    popup_add_elem(popup, "Option 3", NULL);
    popup_add_elem(popup, "Option 4", NULL);
    progress_undefined(progress, TRUE);
    layout_label(layout1, label, 0, 0);
    layout_button(layout1, button1, 0, 1);
    layout_button(layout1, button2, 0, 2);
    layout_slider(layout1, slider, 0, 3);
    layout_popup(layout1, popup, 1, 0);
    layout_edit(layout1, edit, 1, 1);
    layout_layout(layout1, layout2, 1, 2);
    layout_progress(layout1, progress, 1, 3);
    layout_hsize(layout1, 0, 150);
    layout_hsize(layout1, 1, 150);
    layout_hmargin(layout1, 0, 5);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 1, 5);
    layout_vmargin(layout1, 2, 5);
    return layout1;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_left_layout(void)
{
    Layout *layout1 = layout_create(1, 2);
    Layout *layout2 = i_left_grid_layout();
    Button *button = button_push();
    button_text(button, "Clear");
    layout_layout(layout1, layout2, 0, 0);
    layout_button(layout1, button, 0, 1);
    layout_vmargin(layout1, 0, 5);
    return layout1;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_top_layout(void)
{
    Layout *layout1 = layout_create(2, 1);
    Layout *layout2 = i_left_layout();
    TextView *view = textview_create();
    layout_layout(layout1, layout2, 0, 0);
    layout_textview(layout1, view, 1, 0);
    layout_hsize(layout1, 1, 230);
    layout_hmargin(layout1, 0, 5);
    return layout1;
}

/*-----------------------------------------------------------------------------*/
```

```
static Layout *i_bottom_layout(void)
{
    Layout *layout = layout_create(6, 1);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Label *label6 = label_create();
    label_text(label1, "Select 1");
    label_text(label2, "Select 2");
    label_text(label3, "Select 3");
    label_text(label4, "Select 4");
    label_text(label5, "Select 5");
    label_text(label6, "Select 6");
    label_style_over(label1, ekFUNDERLINE);
    label_style_over(label2, ekFUNDERLINE);
    label_style_over(label3, ekFUNDERLINE);
    label_style_over(label4, ekFUNDERLINE);
    label_style_over(label5, ekFUNDERLINE);
    label_style_over(label6, ekFUNDERLINE);
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 1, 0);
    layout_label(layout, label3, 2, 0);
    layout_label(layout, label4, 3, 0);
    layout_label(layout, label5, 4, 0);
    layout_label(layout, label6, 5, 0);
    return layout;
}

/*----------------------------------------------------------------------------*/

static Layout *i_main_layout(void)
{
    Layout *layout1 = layout_create(1, 2);
    Layout *layout2 = i_top_layout();
    Layout *layout3 = i_bottom_layout();
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 0, 1);
    layout_margin(layout1, 5);
    layout_vmargin(layout1, 0, 5);
    return layout1;
}

/*----------------------------------------------------------------------------*/

Panel *sublayouts(void)
{
    Panel *panel = panel_create();
    Layout *layout = i_main_layout();
    panel_layout(panel, layout);
```

```
    return panel;
}
```

26.14. ¡Hola Subpanel!

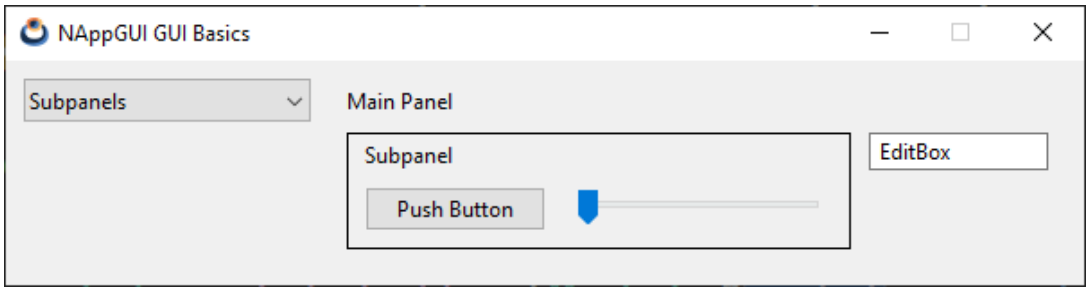


Figura 26.15: Subpaneles.

Listado 26.14: demo/guihello/subpanel.c

```
/* Use of subpanels */

#include "subpanel.h"
#include <gui/guiall.h>

/*---------------------------------------------------------------------------*/

Panel *subpanels(void)
{
    Panel *panel1 = panel_create();
    Panel *panel2 = panel_create();
    Layout *layout1 = layout_create(2, 2);
    Layout *layout2 = layout_create(2, 2);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Button *button = button_push();
    Slider *slider = slider_create();
    Edit *edit = edit_create();
    label_text(label1, "Main Panel");
    label_text(label2, "Subpanel");
    button_text(button, "Push Button");
    edit_text(edit, "EditBox");

    layout_label(layout2, label2, 0, 0);
    layout_button(layout2, button, 0, 1);
    layout_slider(layout2, slider, 1, 1);
    layout_hsize(layout2, 1, 150);
    layout_hmargin(layout2, 0, 10);
    layout_vmargin(layout2, 0, 10);
    layout_margin4(layout2, 5, 10, 10, 10);
```

```
    layout_skcolor(layout2, gui_line_color());
    panel_layout(panel2, layout2);

    layout_label(layout1, label1, 0, 0);
    layout_edit(layout1, edit, 1, 1);
    layout_panel(layout1, panel2, 0, 1);
    layout_hsize(layout1, 1, 100);
    layout_hmargin(layout1, 0, 10);
    layout_vmargin(layout1, 0, 10);
    layout_margin4(layout1, 5, 10, 10, 10);
    panel_layout(panel1, layout1);
    return panel1;
}
```

26.15. ¡Hola Multi-layout!

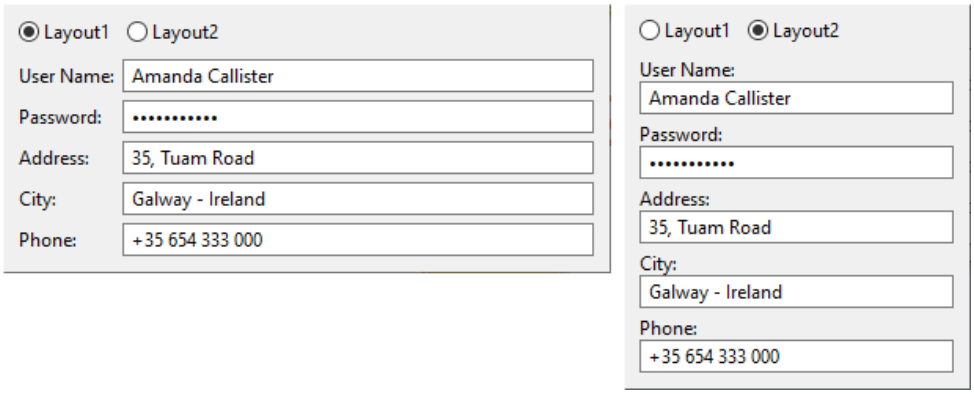


Figura 26.16: `Panel` con dos layouts.

Listado 26.15: demo/guihello/multilayout.c

```
/* Panels with multiple layouts */

#include "multilayout.h"
#include <gui/guiall.h>

/*-------------------------------------------------------------------------*/

static Panel *i_multilayout_panel(void)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(2, 5);
    Layout *layout2 = layout_create(1, 10);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
```

```
Label *label4 = label_create();
Label *label5 = label_create();
Edit *edit1 = edit_create();
Edit *edit2 = edit_create();
Edit *edit3 = edit_create();
Edit *edit4 = edit_create();
Edit *edit5 = edit_create();
label_text(label1, "User Name:");
label_text(label2, "Password:");
label_text(label3, "Address:");
label_text(label4, "City:");
label_text(label5, "Phone:");
edit_text(edit1, "Amanda Callister");
edit_text(edit2, "aQwe56nhjJk");
edit_text(edit3, "35, Tuam Road");
edit_text(edit4, "Galway - Ireland");
edit_text(edit5, "+35 654 333 000");
edit_passmode(edit2, TRUE);

layout_label(layout1, label1, 0, 0);
layout_label(layout1, label2, 0, 1);
layout_label(layout1, label3, 0, 2);
layout_label(layout1, label4, 0, 3);
layout_label(layout1, label5, 0, 4);
layout_edit(layout1, edit1, 1, 0);
layout_edit(layout1, edit2, 1, 1);
layout_edit(layout1, edit3, 1, 2);
layout_edit(layout1, edit4, 1, 3);
layout_edit(layout1, edit5, 1, 4);
layout_hsize(layout1, 1, 300);
layout_hmargin(layout1, 0, 5);
layout_vmargin(layout1, 0, 5);
layout_vmargin(layout1, 1, 5);
layout_vmargin(layout1, 2, 5);
layout_vmargin(layout1, 3, 5);

layout_label(layout2, label1, 0, 0);
layout_label(layout2, label2, 0, 2);
layout_label(layout2, label3, 0, 4);
layout_label(layout2, label4, 0, 6);
layout_label(layout2, label5, 0, 8);
layout_edit(layout2, edit1, 0, 1);
layout_edit(layout2, edit2, 0, 3);
layout_edit(layout2, edit3, 0, 5);
layout_edit(layout2, edit4, 0, 7);
layout_edit(layout2, edit5, 0, 9);
layout_hsize(layout2, 0, 200);
layout_vmargin(layout2, 1, 5);
layout_vmargin(layout2, 3, 5);
layout_vmargin(layout2, 5, 5);
layout_vmargin(layout2, 7, 5);
```

```
    panel_layout(panel, layout1);
    panel_layout(panel, layout2);
    return panel;
}

/*---------------------------------------------------------------------------*/

static void i_OnLayout(Panel *panel, Event *e)
{
    const EvButton *params = event_params(e, EvButton);
    panel_visible_layout(panel, params->index);
    panel_update(panel);
}

/*---------------------------------------------------------------------------*/

Panel *multilayouts(void)
{
    Panel *panel1 = panel_create();
    Panel *panel2 = i_multilayout_panel();
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    Layout *layout1 = layout_create(1, 2);
    Layout *layout2 = layout_create(2, 1);
    button_text(button1, "Layout1");
    button_text(button2, "Layout2");
    button_state(button1, ekGUI_ON);
    button_OnClick(button1, listener(panel2, i_OnLayout, Panel));
    layout_button(layout2, button1, 0, 0);
    layout_button(layout2, button2, 1, 0);
    layout_layout(layout1, layout2, 0, 0);
    layout_panel(layout1, panel2, 0, 1);
    layout_vmargin(layout1, 0, 10);
    layout_hmargin(layout2, 0, 10);
    layout_halign(layout1, 0, 0, ekLEFT);
    panel_layout(panel1, layout1);
    return panel1;
}
```

26.16. ¡Hola Scroll-Panel!

Listado 26.16: demo/guihello/scrollpanel.c

```
/* Panel with scroll */

#include "scrollpanel.h"
#include <gui/guiall.h>

static const uint32_t i_ROWS = 100;
```

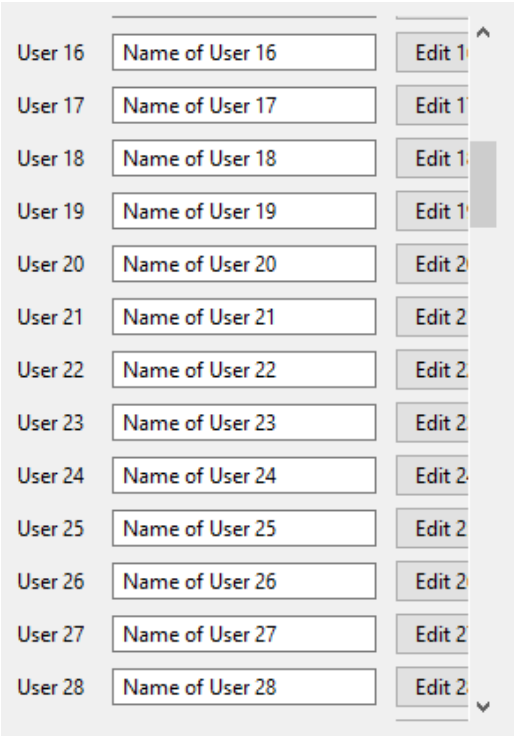


Figura 26.17: `Panel` con barras de scroll.

```
/*--------------------------------------------------------------------------*/

Panel *scrollpanel(void)
{
    Panel *panel = panel_scroll(FALSE, TRUE);
    Layout *layout = layout_create(3, i_ROWS);
    real32_t margin = panel_scroll_width(panel);
    uint32_t i = 0;
    panel_size(panel, s2df(-1, 400));
    for (i = 0; i < i_ROWS; ++i)
    {
        char_t text[128];
        Label *label = label_create();
        Edit *edit = edit_create();
        Button *button = button_push();
        bstd_sprintf(text, sizeof(text), "User %d", i + 1);
        label_text(label, text);
        bstd_sprintf(text, sizeof(text), "Name of User %d", i + 1);
        edit_text(edit, text);
        bstd_sprintf(text, sizeof(text), "Edit %d", i + 1);
        button_text(button, text);
        layout_label(layout, label, 0, i);
        layout_edit(layout, edit, 1, i);
```

```
        layout_button(layout, button, 2, i);
    }

    for (i = 0; i < i_ROWS - 1; ++i)
        layout_vmargin(layout, i, 5);

    layout_hmargin(layout, 0, 10);
    layout_hmargin(layout, 1, 10);
    layout_hsize(layout, 1, 150);
    layout_margin4(layout, 0, margin, 0, 0);
    panel_layout(panel, layout);
    return panel;
}
```

26.17. ¡Hola IP-Input!

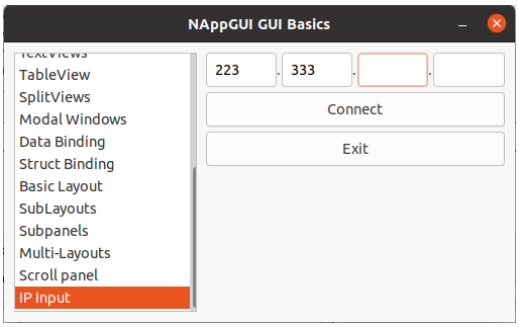


Figura 26.18: Los `Edit` cambian automáticamente el foco del teclado tras insertar el tercer carácter.

Listado 26.17: demo/guihello/ipinput.c

```
/* IP input */

#include "ipinput.h"
#include <gui/guiall.h>

/*---------------------------------------------------------------------------*/

static void i_OnEditFilter(Layout* layout, Event* e)
{
    const EvText *p = event_params(e, EvText);
    EvTextFilter *filter = event_result(e, EvTextFilter);
    uint32_t i, j = 0, n = str_len_c(p->text);

    /* We only accept numbers in IP controls */
    for(i = 0; i < n; ++i)
    {
        if (p->text[i] >= '0' && p->text[i] <= '9')
            filter->text[j++] = p->text[i];
    }
```

```
    if (j > 3)
        j = 3;

    filter->text[j] = '\0';
    filter->apply = TRUE;

    /* We wrote the third character --> Jump to next control */
    if (j == 3)
        layout_next_tabstop(layout);
}

/*--------------------------------------------------------------------------*/

Panel *ip_input(void)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(7, 1);
    Layout *layout2 = layout_create(1, 3);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    Edit *edit3 = edit_create();
    Edit *edit4 = edit_create();
    Button *button1 = button_push();
    Button *button2 = button_push();
    label_text(label1, ".");
    label_text(label2, ".");
    label_text(label3, ".");
    button_text(button1, "Connect");
    button_text(button2, "Exit");
    edit_OnFilter(edit1, listener(layout2, i_OnEditFilter, Layout));
    edit_OnFilter(edit2, listener(layout2, i_OnEditFilter, Layout));
    edit_OnFilter(edit3, listener(layout2, i_OnEditFilter, Layout));
    edit_OnFilter(edit4, listener(layout2, i_OnEditFilter, Layout));
    layout_label(layout1, label1, 1, 0);
    layout_label(layout1, label2, 3, 0);
    layout_label(layout1, label3, 5, 0);
    layout_edit(layout1, edit1, 0, 0);
    layout_edit(layout1, edit2, 2, 0);
    layout_edit(layout1, edit3, 4, 0);
    layout_edit(layout1, edit4, 6, 0);
    layout_layout(layout2, layout1, 0, 0);
    layout_button(layout2, button1, 0, 1);
    layout_button(layout2, button2, 0, 2);
    layout_vmargin(layout2, 0, 5.f);
    layout_vmargin(layout2, 1, 5.f);
    layout_hsize(layout2, 0, 200.f);
    panel_layout(panel, layout2);
    return panel;
```

```
}
```

Capítulo 27

¡Hola Draw2d!

DrawHello es una aplicación, que a modo de ejemplo, muestra las características de la librería *"Draw2D"* (Página 262) para el dibujo vectorial en 2D. Implementa el dibujo de líneas, relleno de regiones, textos e imágenes. El **código fuente** está en la carpeta `/src/howto/drawhello` de la distribución del SDK.

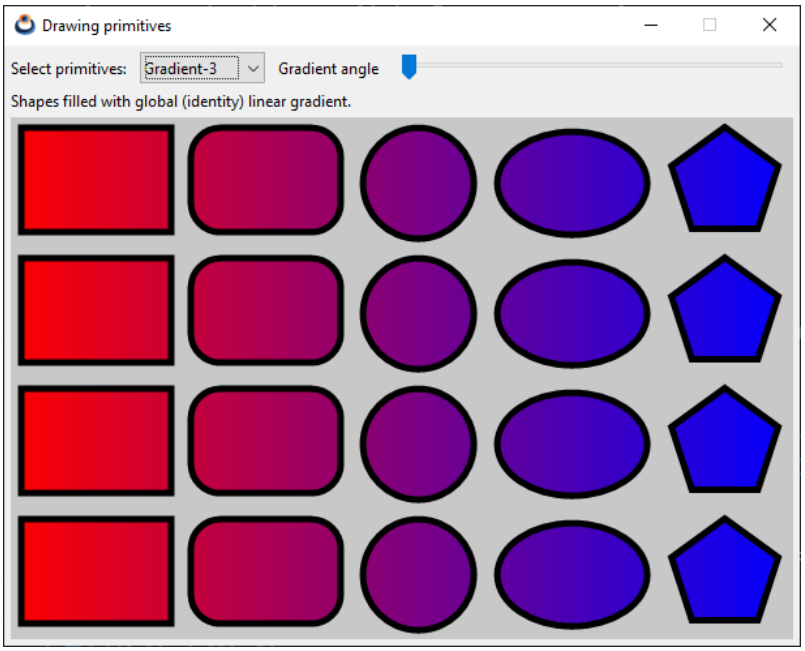


Figura 27.1: Versión Windows.

Listado 27.1: demo/drawhello/drawhello.c

```
/* Drawing primitives */

#include "res_drawhello.h"
```

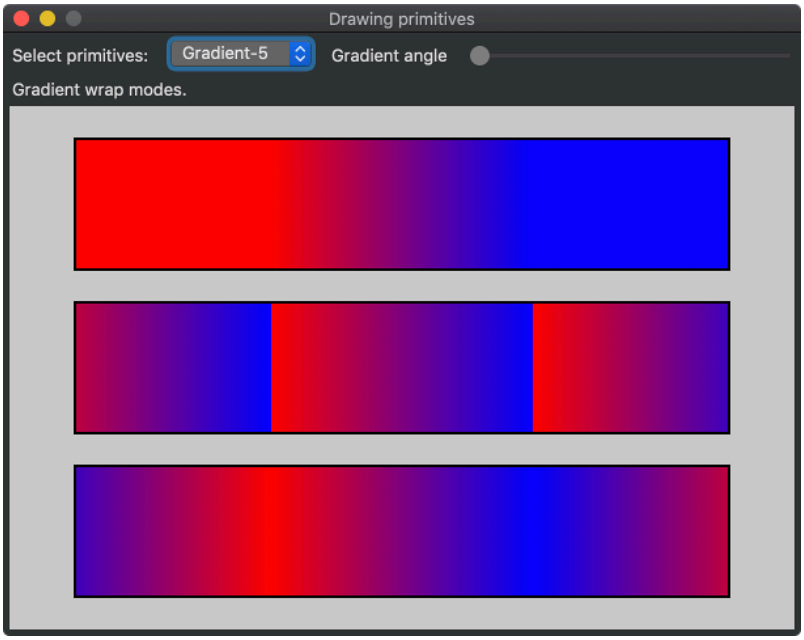


Figura 27.2: Versión macOS.

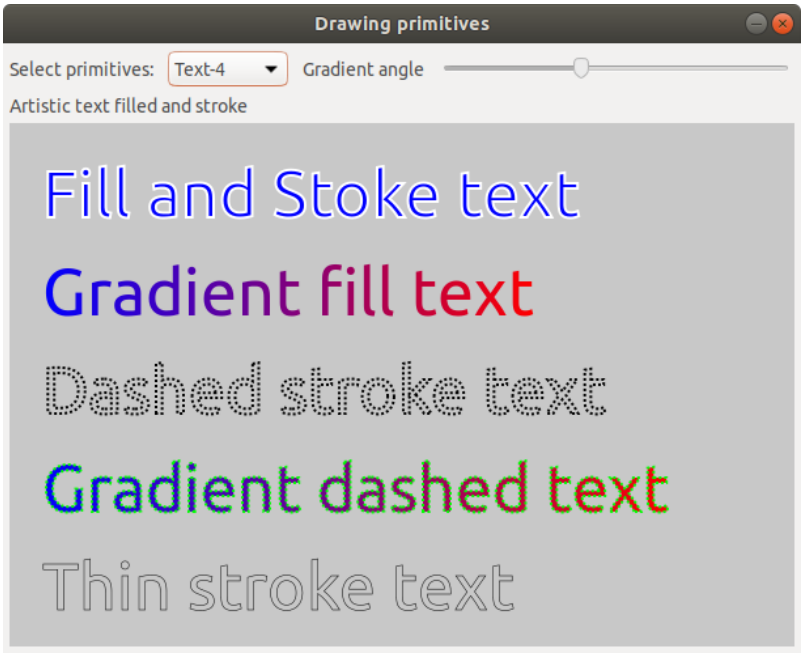


Figura 27.3: Versión Linux.

```
#include <nappgui.h>

typedef struct _app_t App;

struct _app_t
{
    Window *window;
    View *view;
    Label *label;
    Cell *slider;
    uint32_t option;
    real32_t gradient;
};

/*---------------------------------------------------------------------------*/

static void i_draw_lines(DCtx *ctx)
{
    const V2Df poly1[] = { { 10, 190}, { 90, 110}, {110, 190}, {190, 110},
        ↪ {210, 190}, {290, 110} };
    const V2Df poly2[] = { {310, 190}, {390, 110}, {410, 190}, {490, 110},
        ↪ {510, 190}, {590, 110} };
    const V2Df poly3[] = { { 10, 290}, { 90, 210}, {110, 290}, {190, 210},
        ↪ {210, 290}, {290, 210} };
    const real32_t pattern1[] = { 5, 5, 10, 5 };
    const real32_t pattern2[] = { 1, 1 };
    const real32_t pattern3[] = { 2, 1 };
    const real32_t pattern4[] = { 1, 2 };

    /* Line widths */
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_line_width(ctx, 5);
    draw_line(ctx, 10, 90, 90, 10);
    draw_line_width(ctx, 10);
    draw_line(ctx, 110, 90, 190, 10);
    draw_line_width(ctx, 15);
    draw_line(ctx, 210, 90, 290, 10);

    /* Line caps */
    draw_line_cap(ctx, ekLCFLAT);
    draw_line(ctx, 310, 90, 390, 10);
    draw_line_cap(ctx, ekLCSQUARE);
    draw_line(ctx, 410, 90, 490, 10);
    draw_line_cap(ctx, ekLCROUND);
    draw_line(ctx, 510, 90, 590, 10);

    /* Line joins */
    draw_line_width(ctx, 15);
    draw_line_cap(ctx, ekLCFLAT);
    draw_line_join(ctx, ekLJMITER);
    draw_polyline(ctx, FALSE, poly1, 6);
```

```
    draw_line_cap(ctx, ekLCSQUARE);
    draw_line_join(ctx, ekLJROUND);
    draw_polyline(ctx, FALSE, poly2, 6);
    draw_line_cap(ctx, ekLCROUND);
    draw_line_join(ctx, ekLJBEVEL);
    draw_polyline(ctx, FALSE, poly3, 6);

    /* Line colors */
    draw_line_width(ctx, 10);
    draw_line_cap(ctx, ekLCFLAT);
    draw_line_color(ctx, kCOLOR_RED);
    draw_line(ctx, 310, 215, 590, 215);
    draw_line_color(ctx, kCOLOR_GREEN);
    draw_line(ctx, 310, 235, 590, 235);
    draw_line_color(ctx, kCOLOR_BLUE);
    draw_line(ctx, 310, 255, 590, 255);
    draw_line_width(ctx, 5);
    draw_line_color(ctx, kCOLOR_YELLOW);
    draw_line(ctx, 310, 270, 590, 270);
    draw_line_color(ctx, kCOLOR_CYAN);
    draw_line(ctx, 310, 280, 590, 280);
    draw_line_color(ctx, kCOLOR_MAGENTA);
    draw_line(ctx, 310, 290, 590, 290);

    /* Line patterns */
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_line_width(ctx, 5);
    draw_line_cap(ctx, ekLCFLAT);
    draw_line_dash(ctx, pattern1, 4);
    draw_line(ctx, 10, 310, 590, 310);
    draw_line_dash(ctx, pattern2, 2);
    draw_line(ctx, 10, 330, 590, 330);
    draw_line_dash(ctx, pattern3, 2);
    draw_line(ctx, 10, 350, 590, 350);
    draw_line_dash(ctx, pattern4, 2);
    draw_line_width(ctx, 2);
    draw_line(ctx, 10, 365, 590, 365);
    draw_line_dash(ctx, pattern1, 4);
    draw_line_width(ctx, 1);
    draw_line(ctx, 10, 375, 590, 375);
    draw_line_dash(ctx, NULL, 0);
    draw_line(ctx, 10, 385, 590, 385);

    /* Thin lines in centers */
    draw_line_dash(ctx, NULL, 0);
    draw_line_color(ctx, color_rgb(255, 255, 255));
    draw_line_width(ctx, 1);
    draw_line(ctx, 10, 90, 90, 10);
    draw_line(ctx, 110, 90, 190, 10);
    draw_line(ctx, 210, 90, 290, 10);
    draw_line(ctx, 310, 90, 390, 10);
```

```
    draw_line(ctx, 410, 90, 490, 10);
    draw_line(ctx, 510, 90, 590, 10);
    draw_polyline(ctx, FALSE, poly1, 6);
    draw_polyline(ctx, FALSE, poly2, 6);
    draw_polyline(ctx, FALSE, poly3, 6);
}

/*--------------------------------------------------------------------------*/

static void i_draw_shapes_row(DCtx *ctx, const drawop_t op, const T2Df *origin)
{
    const V2Df poly[] = { {40, 0}, {12.36f, 38.04f}, {-32.36f, 23.52f},
                          {-32.36f, -23.52f}, {12.36f, -38.04f} };
    T2Df matrix;
    draw_rect(ctx, op, 10, 10, 110, 75);
    draw_rndrect(ctx, op, 140, 10, 110, 75, 20);
    draw_circle(ctx, op, 312, 50, 40);
    draw_ellipse(ctx, op, 430, 50, 55, 37);
    t2d_movef(&matrix, origin, 547, 50);
    t2d_rotatef(&matrix, &matrix, - kBMATH_PIf / 10);
    draw_matrixf(ctx, &matrix);
    draw_polygon(ctx, op, poly, 5);
}

/*--------------------------------------------------------------------------*/

static void i_draw_shapes(DCtx *ctx, const bool_t grad)
{
    T2Df origin = *kT2D_IDENTf;
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_line_width(ctx, 10);
    draw_matrixf(ctx, &origin);
    i_draw_shapes_row(ctx, grad ? ekSKFILL : ekSTROKE, &origin);
    t2d_movef(&origin, &origin, 0, 100);
    draw_matrixf(ctx, &origin);
    i_draw_shapes_row(ctx, grad ? ekSKFILL : ekFILL, &origin);
    t2d_movef(&origin, &origin, 0, 100);
    draw_matrixf(ctx, &origin);
    i_draw_shapes_row(ctx, grad ? ekSKFILL : ekSKFILL, &origin);
    t2d_movef(&origin, &origin, 0, 100);
    draw_matrixf(ctx, &origin);
    i_draw_shapes_row(ctx, grad ? ekSKFILL : ekFILLSK, &origin);
}

/*--------------------------------------------------------------------------*/

static void i_draw_gradient(DCtx *ctx, const real32_t gradient, const bool_t
    ↪ back, const bool_t shapes)
{
    color_t c[2];
    real32_t stop[2] = {0, 1};
```

```
    real32_t gpos;
    real32_t gx, gy;
    c[0] = kCOLOR_RED;
    c[1] = kCOLOR_BLUE;

    gpos = gradient * (600 + 400);

    if (gpos < 400)
    {
        gx = 600;
        gy = gpos;
    }
    else
    {
        gx = 600 - (gpos - 400);
        gy = 400;
    }

    draw_fill_linear(ctx, c, stop, 2, 0, 0, gx, gy);

    if (back == TRUE)
        draw_rect(ctx, ekFILL, 0, 0, 600, 400);

    if (shapes == TRUE)
        i_draw_shapes(ctx, TRUE);

    draw_matrixf(ctx, kT2D_IDENTf);
    draw_line_width(ctx, 3);
    draw_line_color(ctx, color_rgb(200, 200, 200));
    draw_line(ctx, 3, 3, gx + 3, gy + 3);
}

/*----------------------------------------------------------------------------*/

static void i_draw_lines_gradient(DCtx *ctx, const real32_t gradient)
{
    color_t c[2];
    real32_t stop[2] = {0, 1};
    real32_t gpos;
    real32_t gx, gy;
    const real32_t pattern1[] = { 5, 5, 10, 5 };
    const real32_t pattern2[] = { 1, 1 };
    const real32_t pattern3[] = { 2, 1 };
    const real32_t pattern4[] = { 1, 2 };

    c[0] = kCOLOR_RED;
    c[1] = kCOLOR_BLUE;

    gpos = gradient * (600 + 400);

    if (gpos < 400)
```

```
{
    gx = 600;
    gy = gpos;
}
else
{
    gx = 600 - (gpos - 400);
    gy = 400;
}

draw_line_width(ctx, 10);
draw_line_fill(ctx);
draw_fill_linear(ctx, c, stop, 2, 0, 0, gx, gy);
i_draw_shapes_row(ctx, ekSTROKE, kT2D_IDENTf);

draw_matrixf(ctx, kT2D_IDENTf);
draw_line_width(ctx, 1);
draw_bezier(ctx, 30, 190, 140, 50, 440, 110, 570, 190);
draw_line_width(ctx, 4);
draw_bezier(ctx, 30, 210, 140, 70, 440, 130, 570, 210);
draw_line_width(ctx, 7);
draw_bezier(ctx, 30, 230, 140, 90, 440, 150, 570, 230);
draw_line_width(ctx, 10);
draw_bezier(ctx, 30, 250, 140, 110, 440, 170, 570, 250);

draw_line_width(ctx, 8);
draw_arc(ctx, 100, 280, 60, 0, - kBMATH_PIf / 2);
draw_arc(ctx, 250, 280, 60, kBMATH_PIf, kBMATH_PIf / 2);
draw_arc(ctx, 300, 220, 60, kBMATH_PIf / 2, - kBMATH_PIf / 2);
draw_arc(ctx, 450, 220, 60, kBMATH_PIf / 2, kBMATH_PIf / 2);

draw_line_width(ctx, 5);
draw_line_cap(ctx, ekLCFLAT);
draw_line_dash(ctx, pattern1, 4);
draw_line(ctx, 10, 310, 590, 310);
draw_line_dash(ctx, pattern2, 2);
draw_line(ctx, 10, 330, 590, 330);
draw_line_dash(ctx, pattern3, 2);
draw_line(ctx, 10, 350, 590, 350);
draw_line_dash(ctx, pattern4, 2);
draw_line_width(ctx, 2);
draw_line(ctx, 10, 365, 590, 365);
draw_line_dash(ctx, pattern1, 4);
draw_line_width(ctx, 1);
draw_line(ctx, 10, 375, 590, 375);
draw_line_dash(ctx, NULL, 0);
draw_line(ctx, 10, 385, 590, 385);

draw_line_width(ctx, 1);
draw_line_color(ctx, color_rgb(50, 50, 50));
draw_line(ctx, 3, 3, gx + 3, gy + 3);
```

```
}

/*---------------------------------------------------------------------------*/

static void i_draw_local_gradient(DCtx *ctx, const real32_t gradient)
{
    color_t c[2];
    real32_t stop[2] = {0, 1};
    real32_t gpos;
    real32_t gx, gy;
    T2Df matrix;

    c[0] = kCOLOR_RED;
    c[1] = kCOLOR_BLUE;

    gpos = gradient * (200 + 100);

    if (gpos < 100)
    {
        gx = 200;
        gy = gpos;
    }
    else
    {
        gx = 200 - (gpos - 100);
        gy = 100;
    }

    draw_line_join(ctx, ekLJROUND);
    draw_fill_linear(ctx, c, stop, 2, 0, 0, gx, gy);

    t2d_movef(&matrix, kT2D_IDENTf, 50, 40);
    draw_matrixf(ctx, &matrix);
    draw_fill_matrix(ctx, &matrix);
    draw_line_width(ctx, 10);
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_rect(ctx, ekSKFILL, 0, 0, 200, 100);
    draw_line_width(ctx, 3);
    draw_line_color(ctx, color_rgb(200, 200, 200));
    draw_line(ctx, 0, 0, gx, gy);

    t2d_movef(&matrix, kT2D_IDENTf, 400, 40);
    t2d_rotatef(&matrix, &matrix, kBMATH_PIf / 6);
    draw_matrixf(ctx, &matrix);
    draw_fill_matrix(ctx, &matrix);
    draw_line_width(ctx, 10);
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_rect(ctx, ekSKFILL, 0, 0, 200, 100);
    draw_line_width(ctx, 3);
    draw_line_color(ctx, color_rgb(200, 200, 200));
    draw_line(ctx, 0, 0, gx, gy);
```

```
    t2d_movef(&matrix, kT2D_IDENTf, 250, 280);
    t2d_rotatef(&matrix, &matrix, - kBMATH_PIf / 10);
    draw_matrixf(ctx, &matrix);
    t2d_movef(&matrix, &matrix, -100, -50);
    draw_fill_matrix(ctx, &matrix);
    draw_line_width(ctx, 10);
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_ellipse(ctx, ekSKFILL, 0, 0, 100, 50);
    draw_matrixf(ctx, &matrix);
    draw_line_width(ctx, 3);
    draw_line_color(ctx, color_rgb(200, 200, 200));
    draw_line(ctx, 0, 0, gx, gy);
}

/*----------------------------------------------------------------------------*/

static void i_draw_wrap_gradient(DCtx *ctx)
{
    color_t c[2];
    real32_t stop[2] = {0, 1};
    c[0] = kCOLOR_RED;
    c[1] = kCOLOR_BLUE;
    draw_line_width(ctx, 2);
    draw_fill_linear(ctx, c, stop, 2, 200, 0, 400, 0);
    draw_fill_wrap(ctx, ekFCLAMP);
    draw_rect(ctx, ekFILLSK, 50, 25, 500, 100);
    draw_fill_wrap(ctx, ekFTILE);
    draw_rect(ctx, ekFILLSK, 50, 150, 500, 100);
    draw_fill_wrap(ctx, ekFFLIP);
    draw_rect(ctx, ekFILLSK, 50, 275, 500, 100);
}

/*----------------------------------------------------------------------------*/

static void i_text_single(DCtx *ctx)
{
    Font *font = font_system(20, 0);
    const char_t *text = "Text □□Κείμενο ";
    real32_t width, height;
    T2Df matrix;

    draw_font(ctx, font);
    draw_text_extents(ctx, text, -1, &width, &height);
    draw_text_color(ctx, kCOLOR_BLUE);
    draw_text_align(ctx, ekLEFT, ekTOP);
    draw_text(ctx, text, 25, 25);
    draw_text_align(ctx, ekCENTER, ekTOP);
    draw_text(ctx, text, 300, 25);
    draw_text_align(ctx, ekRIGHT, ekTOP);
    draw_text(ctx, text, 575, 25);
```

```
    draw_text_align(ctx, ekLEFT, ekCENTER);
    draw_text(ctx, text, 25, 100);
    draw_text_align(ctx, ekCENTER, ekCENTER);
    draw_text(ctx, text, 300, 100);
    draw_text_align(ctx, ekRIGHT, ekCENTER);
    draw_text(ctx, text, 575, 100);
    draw_text_align(ctx, ekLEFT, ekBOTTOM);
    draw_text(ctx, text, 25, 175);
    draw_text_align(ctx, ekCENTER, ekBOTTOM);
    draw_text(ctx, text, 300, 175);
    draw_text_align(ctx, ekRIGHT, ekBOTTOM);
    draw_text(ctx, text, 575, 175);

    draw_line_color(ctx, kCOLOR_RED);
    draw_fill_color(ctx, kCOLOR_RED);
    draw_circle(ctx, ekFILL, 25, 25, 3);
    draw_circle(ctx, ekFILL, 300, 25, 3);
    draw_circle(ctx, ekFILL, 575, 25, 3);
    draw_circle(ctx, ekFILL, 25, 100, 3);
    draw_circle(ctx, ekFILL, 300, 100, 3);
    draw_circle(ctx, ekFILL, 575, 100, 3);
    draw_circle(ctx, ekFILL, 25, 175, 3);
    draw_circle(ctx, ekFILL, 300, 175, 3);
    draw_circle(ctx, ekFILL, 575, 175, 3);
    draw_circle(ctx, ekFILL, 25, 200, 3);
    draw_circle(ctx, ekFILL, 300, 250, 3);
    draw_circle(ctx, ekFILL, 25, 325, 3);
    draw_circle(ctx, ekFILL, 575, 200, 3);
    draw_circle(ctx, ekFILL, 575, 230, 3);
    draw_circle(ctx, ekFILL, 575, 260, 3);
    draw_rect(ctx, ekSTROKE, 25, 25, width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 25, width, height);
    draw_rect(ctx, ekSTROKE, 575 - width, 25, width, height);
    draw_rect(ctx, ekSTROKE, 25, 100 - (height / 2), width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 100 - (height / 2), width,
        ↪ height);
    draw_rect(ctx, ekSTROKE, 575 - width, 100 - (height / 2), width, height);
    draw_rect(ctx, ekSTROKE, 25, 175 - height, width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 175 - height, width, height);
    draw_rect(ctx, ekSTROKE, 575 - width, 175 - height, width, height);

    draw_fill_color(ctx, kCOLOR_BLUE);
    t2d_movef(&matrix, kT2D_IDENTf, 25, 200);
    t2d_rotatef(&matrix, &matrix, kBMATH_PIf / 8);
    draw_matrixf(ctx, &matrix);
    draw_text_align(ctx, ekLEFT, ekTOP);
    draw_text(ctx, text, 0, 0);

    t2d_movef(&matrix, kT2D_IDENTf, 300, 250);
    t2d_rotatef(&matrix, &matrix, - kBMATH_PIf / 8);
    draw_matrixf(ctx, &matrix);
```

```
    draw_text_align(ctx, ekCENTER, ekCENTER);
    draw_text(ctx, text, 0, 0);

    t2d_movef(&matrix, kT2D_IDENTf, 25, 325);
    t2d_scalef(&matrix, &matrix, 3, 1);
    draw_matrixf(ctx, &matrix);
    draw_text_align(ctx, ekLEFT, ekTOP);
    draw_text(ctx, text, 0, 0);

    t2d_movef(&matrix, kT2D_IDENTf, 575, 200);
    t2d_scalef(&matrix, &matrix, .5f, 1);
    draw_matrixf(ctx, &matrix);
    draw_text_align(ctx, ekRIGHT, ekTOP);
    draw_text(ctx, text, 0, 0);

    t2d_movef(&matrix, kT2D_IDENTf, 575, 230);
    t2d_scalef(&matrix, &matrix, .75f, 1);
    draw_matrixf(ctx, &matrix);
    draw_text_align(ctx, ekRIGHT, ekTOP);
    draw_text(ctx, text, 0, 0);

    t2d_movef(&matrix, kT2D_IDENTf, 575, 260);
    t2d_scalef(&matrix, &matrix, 1.25f, 1);
    draw_matrixf(ctx, &matrix);
    draw_text_align(ctx, ekRIGHT, ekTOP);
    draw_text(ctx, text, 0, 0);

    font_destroy(&font);
}

/*----------------------------------------------------------------------------*/

static void i_text_newline(DCtx *ctx)
{
    Font *font = font_system(20, 0);
    const char_t *text = "Text new line\□□□□n\Γραμμήη κειμένου";
    real32_t width, height;
    draw_font(ctx, font);
    draw_text_extents(ctx, text, -1, &width, &height);

    draw_text_color(ctx, kCOLOR_BLUE);
    draw_text_align(ctx, ekLEFT, ekTOP);
    draw_text_halign(ctx, ekLEFT);
    draw_text(ctx, text, 25, 25);
    draw_text_align(ctx, ekCENTER, ekTOP);
    draw_text_halign(ctx, ekCENTER);
    draw_text(ctx, text, 300, 25);

    draw_text_align(ctx, ekRIGHT, ekTOP);
    draw_text_halign(ctx, ekRIGHT);
    draw_text(ctx, text, 575, 25);
```

```
    draw_text_align(ctx, ekLEFT, ekCENTER);
    draw_text_halign(ctx, ekLEFT);
    draw_text(ctx, text, 25, 175);
    draw_text_align(ctx, ekCENTER, ekCENTER);
    draw_text_halign(ctx, ekCENTER);
    draw_text(ctx, text, 300, 175);
    draw_text_align(ctx, ekRIGHT, ekCENTER);
    draw_text_halign(ctx, ekRIGHT);
    draw_text(ctx, text, 575, 175);
    draw_text_align(ctx, ekLEFT, ekBOTTOM);
    draw_text_halign(ctx, ekLEFT);
    draw_text(ctx, text, 25, 325);
    draw_text_align(ctx, ekCENTER, ekBOTTOM);
    draw_text_halign(ctx, ekCENTER);
    draw_text(ctx, text, 300, 325);
    draw_text_align(ctx, ekRIGHT, ekBOTTOM);
    draw_text_halign(ctx, ekRIGHT);
    draw_text(ctx, text, 575, 325);

    draw_line_color(ctx, kCOLOR_RED);
    draw_fill_color(ctx, kCOLOR_RED);
    draw_circle(ctx, ekFILL, 25, 25, 3);
    draw_circle(ctx, ekFILL, 300, 25, 3);
    draw_circle(ctx, ekFILL, 575, 25, 3);
    draw_circle(ctx, ekFILL, 25, 175, 3);
    draw_circle(ctx, ekFILL, 300, 175, 3);
    draw_circle(ctx, ekFILL, 575, 175, 3);
    draw_circle(ctx, ekFILL, 25, 325, 3);
    draw_circle(ctx, ekFILL, 300, 325, 3);
    draw_circle(ctx, ekFILL, 575, 325, 3);
    draw_rect(ctx, ekSTROKE, 25, 25, width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 25, width, height);
    draw_rect(ctx, ekSTROKE, 575 - width, 25, width, height);
    draw_rect(ctx, ekSTROKE, 25, 175 - (height / 2), width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 175 - (height / 2), width,
        ↪ height);
    draw_rect(ctx, ekSTROKE, 575 - width, 175 - (height / 2), width, height);
    draw_rect(ctx, ekSTROKE, 25, 325 - height, width, height);
    draw_rect(ctx, ekSTROKE, 300 - (width / 2), 325 - height, width, height);
    draw_rect(ctx, ekSTROKE, 575 - width, 325 - height, width, height);
    font_destroy(&font);
}

/*-------------------------------------------------------------------------*/

static void i_text_block(DCtx *ctx)
{
    const char_t *text = "Lorem ipsum dolor sit amet, consectetur adipiscing
        ↪ elit, sed do eiusmod tempor incididunt ut labore et dolore magna
        ↪ aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
        ↪ laboris nisi ut aliquip ex ea commodo consequat.";
```

```
    real32_t dash[2] = {1, 1};
    real32_t width1, height1;
    real32_t width2, height2;
    real32_t width3, height3;
    real32_t width4, height4;

    draw_text_color(ctx, kCOLOR_BLUE);
    draw_text_align(ctx, ekLEFT, ekTOP);
    draw_text_halign(ctx, ekLEFT);
    draw_text_width(ctx, 200);
    draw_text_extents(ctx, text, 200, &width1, &height1);
    draw_text(ctx, text, 25, 25);
    draw_text_width(ctx, 300);
    draw_text_extents(ctx, text, 300, &width2, &height2);
    draw_text(ctx, text, 250, 25);
    draw_text_width(ctx, 400);
    draw_text_extents(ctx, text, 400, &width3, &height3);
    draw_text(ctx, text, 25, 200);
    draw_text_width(ctx, 500);
    draw_text_extents(ctx, text, 500, &width4, &height4);
    draw_text(ctx, text, 25, 315);

    draw_line_color(ctx, kCOLOR_RED);
    draw_fill_color(ctx, kCOLOR_RED);
    draw_circle(ctx, ekFILL, 25, 25, 3);
    draw_circle(ctx, ekFILL, 250, 25, 3);
    draw_circle(ctx, ekFILL, 25, 200, 3);
    draw_circle(ctx, ekFILL, 25, 315, 3);
    draw_rect(ctx, ekSTROKE, 25, 25, 200, height1);
    draw_rect(ctx, ekSTROKE, 250, 25, 300, height2);
    draw_rect(ctx, ekSTROKE, 25, 200, 400, height3);
    draw_rect(ctx, ekSTROKE, 25, 315, 500, height4);
    draw_line_dash(ctx, dash, 2);
    draw_rect(ctx, ekSTROKE, 25, 25, width1, height1);
    draw_rect(ctx, ekSTROKE, 250, 25, width2, height2);
    draw_rect(ctx, ekSTROKE, 25, 200, width3, height3);
    draw_rect(ctx, ekSTROKE, 25, 315, width4, height4);
}

/*----------------------------------------------------------------------------*/

static void i_text_art(DCtx *ctx)
{
    Font *font = font_system(50, 0);
    color_t c[2];
    real32_t stop[2] = {0, 1};
    real32_t dash[2] = {1, 1};
    real32_t width, height;
    c[0] = kCOLOR_BLUE;
    c[1] = kCOLOR_RED;
    draw_font(ctx, font);
```

```
    draw_line_width(ctx, 2);
    draw_line_color(ctx, kCOLOR_WHITE);
    draw_fill_color(ctx, kCOLOR_BLUE);
    draw_text_path(ctx, ekFILLSK, "Fill and Stoke text", 25, 25);
    draw_text_extents(ctx, "Gradient fill text", -1, &width, &height);
    draw_fill_linear(ctx, c, stop, 2, 25, 0, 25 + width, 0);
    draw_fill_matrix(ctx, kT2D_IDENTf);
    draw_text_path(ctx, ekFILL, "Gradient fill text", 25, 100);
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_line_dash(ctx, dash, 2);
    draw_text_path(ctx, ekSTROKE, "Dashed stroke text", 25, 175);
    draw_line_color(ctx, kCOLOR_GREEN);
    draw_text_extents(ctx, "Gradient dashed text", -1, &width, &height);
    draw_fill_linear(ctx, c, stop, 2, 25, 0, 25 + width, 0);
    draw_text_path(ctx, ekFILLSK, "Gradient dashed text", 25, 250);
    draw_line_color(ctx, kCOLOR_BLACK);
    draw_line_width(ctx, .5f);
    draw_line_dash(ctx, NULL, 0);
    draw_text_path(ctx, ekSTROKE, "Thin stroke text", 25, 325);
    font_destroy(&font);
}

/*-----------------------------------------------------------------------------*/

static void i_image(DCtx *ctx)
{
    ResPack *pack = res_drawhello_respack("");
    const Image *image = image_from_resource(pack, IMAGE_PNG);
    T2Df matrix;

    draw_image_align(ctx, ekLEFT, ekTOP);
    draw_image(ctx, image, 25, 25);
    t2d_movef(&matrix, kT2D_IDENTf, 300, 200);
    t2d_rotatef(&matrix, &matrix, kBMATH_PIf / 8);
    draw_image_align(ctx, ekCENTER, ekCENTER);
    draw_matrixf(ctx, &matrix);
    draw_image(ctx, image, 0, 0);
    draw_matrixf(ctx, kT2D_IDENTf);
    draw_image_align(ctx, ekRIGHT, ekTOP);
    draw_image(ctx, image, 575, 25);
    draw_image_align(ctx, ekLEFT, ekBOTTOM);
    draw_image(ctx, image, 25, 375);
    draw_image_align(ctx, ekRIGHT, ekBOTTOM);
    draw_image(ctx, image, 575, 375);

    draw_fill_color(ctx, kCOLOR_BLUE);
    draw_circle(ctx, ekFILL, 25, 25, 3);
    draw_circle(ctx, ekFILL, 300, 200, 3);
    draw_circle(ctx, ekFILL, 575, 25, 3);
    draw_circle(ctx, ekFILL, 25, 375, 3);
    draw_circle(ctx, ekFILL, 575, 375, 3);
```

```
    respack_destroy(&pack);
}

/*-----------------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);
    draw_clear(p->ctx, color_rgb(200, 200, 200));
    switch (app->option) {
    case 0:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Different line styles: width, join, cap, dash
            ↪ ...");
        i_draw_lines(p->ctx);
        break;
    case 1:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Basic shapes filled and stroke.");
        draw_fill_color(p->ctx, kCOLOR_BLUE);
        i_draw_shapes(p->ctx, FALSE);
        break;
    case 2:
        cell_enabled(app->slider, TRUE);
        label_text(app->label, "Global linear gradient.");
        i_draw_gradient(p->ctx, app->gradient, TRUE, FALSE);
        break;
    case 3:
        cell_enabled(app->slider, TRUE);
        label_text(app->label, "Shapes filled with global (identity) linear
            ↪ gradient.");
        i_draw_gradient(p->ctx, app->gradient, TRUE, TRUE);
        break;
    case 4:
        cell_enabled(app->slider, TRUE);
        label_text(app->label, "Shapes filled with global (identity) linear
            ↪ gradient.");
        i_draw_gradient(p->ctx, app->gradient, FALSE, TRUE);
        break;
    case 5:
        cell_enabled(app->slider, TRUE);
        label_text(app->label, "Lines with global (identity) linear gradient.")
            ↪ ;
        i_draw_lines_gradient(p->ctx, app->gradient);
        break;
    case 6:
        cell_enabled(app->slider, TRUE);
        label_text(app->label, "Shapes filled with local (transformed) gradient
            ↪ .");
        i_draw_local_gradient(p->ctx, app->gradient);
        break;
```

```
    case 7:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Gradient wrap modes.");
        i_draw_wrap_gradient(p->ctx);
        break;
    case 8:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Single line text with alignment and transforms"
            ↪ );
        i_text_single(p->ctx);
        break;
    case 9:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Text with newline '\\n' character and internal
            ↪ alignment");
        i_text_newline(p->ctx);
        break;
    case 10:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Text block in a constrained width area");
        i_text_block(p->ctx);
        break;
    case 11:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Artistic text filled and stroke");
        i_text_art(p->ctx);
        break;
    case 12:
        cell_enabled(app->slider, FALSE);
        label_text(app->label, "Drawing images with alignment");
        i_image(p->ctx);
        break;
    }
}

/*-----------------------------------------------------------------------------*/

static void i_OnSelect(App *app, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    app->option = p->index;
    view_update(app->view);
}

/*-----------------------------------------------------------------------------*/

static void i_OnSlider(App *app, Event *e)
{
    const EvSlider *p = event_params(e, EvSlider);
    app->gradient = p->pos;
    view_update(app->view);
```

```
}

/*---------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(1, 3);
    Layout *layout2 = layout_create(4, 1);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_multiline();
    PopUp *popup = popup_create();
    Slider *slider = slider_create();
    View *view = view_create();
    label_text(label1, "Select primitives:");
    label_text(label2, "Gradient angle");
    popup_add_elem(popup, "Lines", NULL);
    popup_add_elem(popup, "Shapes", NULL);
    popup_add_elem(popup, "Gradient-1", NULL);
    popup_add_elem(popup, "Gradient-2", NULL);
    popup_add_elem(popup, "Gradient-3", NULL);
    popup_add_elem(popup, "Gradient-4", NULL);
    popup_add_elem(popup, "Gradient-5", NULL);
    popup_add_elem(popup, "Gradient-6", NULL);
    popup_add_elem(popup, "Text-1", NULL);
    popup_add_elem(popup, "Text-2", NULL);
    popup_add_elem(popup, "Text-3", NULL);
    popup_add_elem(popup, "Text-4", NULL);
    popup_add_elem(popup, "Image", NULL);
    popup_list_height(popup, 6);
    popup_OnSelect(popup, listener(app, i_OnSelect, App));
    slider_OnMoved(slider, listener(app, i_OnSlider, App));
    view_size(view, s2df(600, 400));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    layout_label(layout2, label1, 0, 0);
    layout_popup(layout2, popup, 1, 0);
    layout_label(layout2, label2, 2, 0);
    layout_slider(layout2, slider, 3, 0);
    layout_layout(layout1, layout2, 0, 0);
    layout_label(layout1, label3, 0, 1);
    layout_view(layout1, view, 0, 2);
    layout_margin(layout1, 5);
    layout_hmargin(layout2, 0, 10);
    layout_hmargin(layout2, 1, 10);
    layout_hmargin(layout2, 2, 10);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 1, 5);
    layout_halign(layout1, 0, 1, ekJUSTIFY);
    layout_hexpand(layout2, 3);
    panel_layout(panel, layout1);
```

```
    app->slider = layout_cell(layout2, 3, 0);
    app->view = view;
    app->label = label3;
    return panel;
}

/*---------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STD);
    app->gradient = 0;
    app->option = 0;
    window_panel(app->window, panel);
    window_title(app->window, "Drawing primitives");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 28

¡Hola Colisiones 2D!

Col2dHello es un pequeño entorno para la experimentación con algoritmos de detección de colisiones en 2D. Permite crear diferentes tipos de volúmenes, moverlos con el ratón y editarlos mediante el panel lateral. El detalle de las funciones lo puedes encontrar en *"Colisiones 2D"* (Página 259).

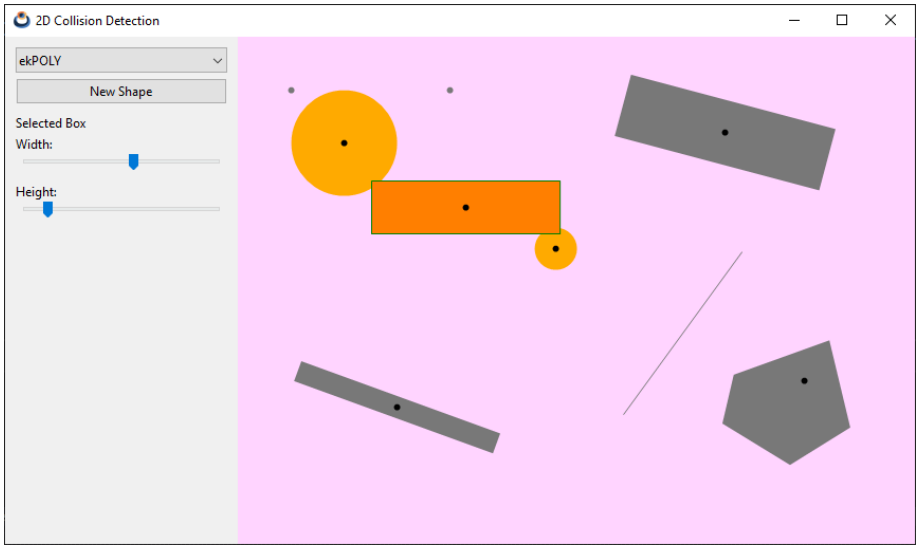


Figura 28.1: Versión Windows.

Listado 28.1: demo/col2dhello/col2dhello.c

```
/* 2D collision detection demo */

#include "col2dgui.h"
#include <nappgui.h>

/*---------------------------------------------------------------------------*/
```

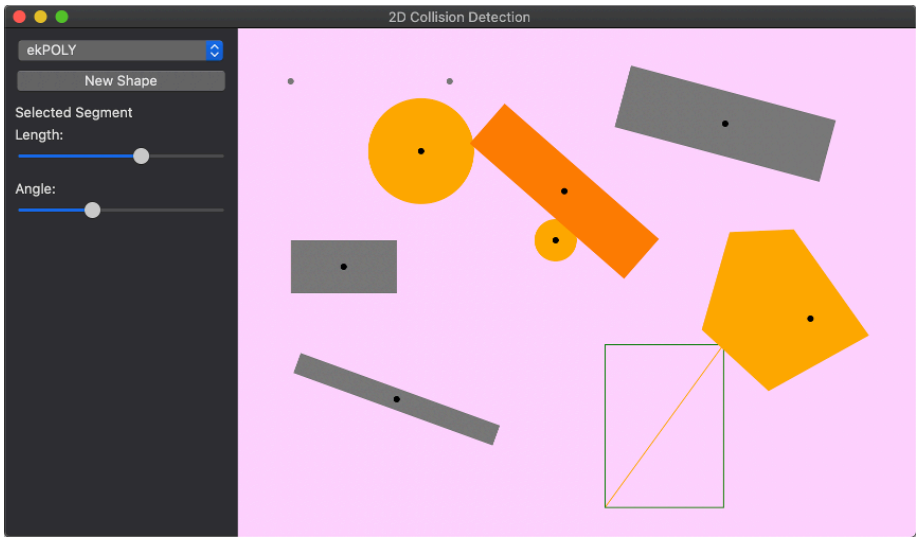


Figura 28.2: Versión macOS.

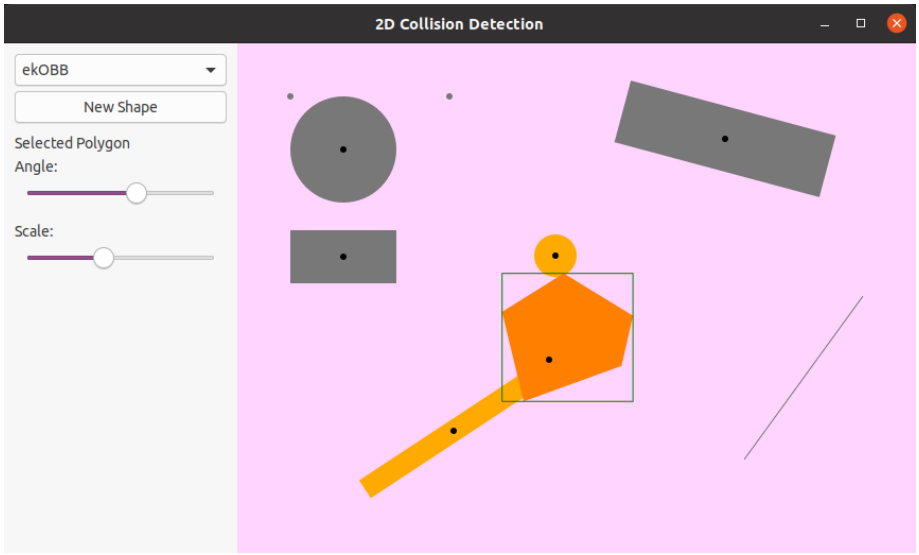


Figura 28.3: Versión Linux.

```
static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}
```

```
/*-----------------------------------------------------------------------------*/

static Tri2Df i_triangle(void)
{
    Tri2Df tri = tri2df(-3, 4, -1, -2, 7, -2);
    cassert(tri2d_ccwf(&tri) == TRUE);
    return tri;
}

/*-----------------------------------------------------------------------------*/

static Pol2Df *i_convex_pol(void)
{
    V2Df pt[] = { {4,1}, {2,5}, {-3,5}, {-4,2}, {0,-3} };
    Pol2Df *pol = NULL;
    bmem_rev_elems(pt, sizeof(pt) / sizeof(V2Df), V2Df);
    pol = pol2d_createf(pt, sizeof(pt) / sizeof(V2Df));
    cassert(pol2d_convexf(pol) == TRUE);
    cassert(pol2d_ccwf(pol) == FALSE);
    return pol;
}

/*-----------------------------------------------------------------------------*/

static Pol2Df *i_simple_pol(void)
{
    V2Df pt[] = { {9.78f, 12.17f}, {-10.00f, 11.01f}, {-9.68f, 3.20f}, {-9.30f,
        ↪  -5.98f}, {-4.27f, -5.84f}, {-4.03f, -12.17f}, {2.72f, -12.12f},
        ↪ {2.47f, -6.36f}, {2.04f, 3.26f}, {-1.45f, 3.05f}, {-1.08f, -2.08f},
        ↪ {-3.98f, -2.38f}, {-4.23f, 2.88f}, {-1.45f, 3.05f}, {2.04f, 3.26f},
        ↪ {10.00f, 3.75f} };
    Pol2Df *pol = NULL;
    bmem_rev_elems(pt, sizeof(pt) / sizeof(V2Df), V2Df);
    pol = pol2d_createf(pt, sizeof(pt) / sizeof(V2Df));
    cassert(pol2d_convexf(pol) == FALSE);
    cassert(pol2d_ccwf(pol) == FALSE);
    return pol;
}

/*-----------------------------------------------------------------------------*/

static Shape *i_new_shape(ArrSt(Shape) *shapes, const shtype_t type)
{
    Shape *shape = arrst_new(shapes, Shape);
    shape->type = type;
    shape->mouse = FALSE;
    shape->collisions = 0;
    return shape;
}

/*-----------------------------------------------------------------------------*/
```

```c
static void i_new_pnt(ArrSt(Shape) *shapes, const real32_t x, const real32_t y)
{
    Shape *shape = i_new_shape(shapes, ekPOINT);
    shape->body.pnt.x = x;
    shape->body.pnt.y = y;
}

/*-----------------------------------------------------------------------*/

static void i_new_cloud(ArrSt(Shape) *shapes, const real32_t x, const real32_t
    ↪ y, const real32_t w, const real32_t h, const real32_t a)
{
    Shape *shape = i_new_shape(shapes, ekPOINT_CLOUD);
    shape->body.cloud.pnts = arrst_create(V2Df);
    shape->body.cloud.center.x = x;
    shape->body.cloud.center.y = y;
    shape->body.cloud.width = w;
    shape->body.cloud.height = h;
    shape->body.cloud.angle = a;
    shape->body.cloud.ctype = 0;
    shape->body.cloud.type = 0;
    (void)arrst_new_n(shape->body.cloud.pnts, POINT_CLOUD_N, V2Df);
    col2dhello_update_cloud(&shape->body.cloud);
}

/*-----------------------------------------------------------------------*/

static void i_new_seg(ArrSt(Shape) *shapes, const real32_t x, const real32_t y,
    ↪  const real32_t l, const real32_t a)
{
    Shape *shape = i_new_shape(shapes, ekSEGMENT);
    shape->body.seg.center.x = x;
    shape->body.seg.center.y = y;
    shape->body.seg.length = l;
    shape->body.seg.angle = a;
    col2dhello_update_seg(&shape->body.seg);
}

/*-----------------------------------------------------------------------*/

static void i_new_cir(ArrSt(Shape) *shapes, const real32_t x, const real32_t y,
    ↪  const real32_t r)
{
    Shape *shape = i_new_shape(shapes, ekCIRCLE);
    shape->body.cir.r = r;
    shape->body.cir.c.x = x;
    shape->body.cir.c.y = y;
}

/*-----------------------------------------------------------------------*/
```

```c
static void i_new_box(ArrSt(Shape) *shapes, const real32_t x, const real32_t y,
    ↪  const real32_t w, const real32_t h)
{
    Shape *shape = i_new_shape(shapes, ekBOX);
    shape->body.box.center.x = x;
    shape->body.box.center.y = y;
    shape->body.box.width = w;
    shape->body.box.height = h;
    col2dhello_update_box(&shape->body.box);
}

/*-----------------------------------------------------------------------------*/

static void i_new_obb(ArrSt(Shape) *shapes, const real32_t x, const real32_t y,
    ↪  const real32_t w, const real32_t h, const real32_t a)
{
    Shape *shape = i_new_shape(shapes, ekOBB);
    shape->body.obb.center.x = x;
    shape->body.obb.center.y = y;
    shape->body.obb.angle = a;
    shape->body.obb.width = w;
    shape->body.obb.height = h;
    shape->body.obb.obb = NULL;
    col2dhello_update_obb(&shape->body.obb);
}

/*-----------------------------------------------------------------------------*/

static void i_new_tri(ArrSt(Shape) *shapes, const real32_t x, const real32_t y,
    ↪  const real32_t a, const real32_t s)
{
    Shape *shape = i_new_shape(shapes, ekTRIANGLE);
    shape->body.tri.center.x = x;
    shape->body.tri.center.y = y;
    shape->body.tri.angle = a;
    shape->body.tri.scale = s;
    shape->body.tri.t2d = *kT2D_IDENTf;
    shape->body.tri.tri = i_triangle();
    col2dhello_update_tri(&shape->body.tri);
}

/*-----------------------------------------------------------------------------*/

static void i_new_pol(ArrSt(Shape) *shapes, const shtype_t type, const real32_t
    ↪  x, const real32_t y, const real32_t a, const real32_t s)
{
    Shape *shape = i_new_shape(shapes, type);
    shape->body.pol.center.x = x;
    shape->body.pol.center.y = y;
    shape->body.pol.angle = a;
```

```
    shape->body.pol.scale = s;
    shape->body.pol.t2d = *kT2D_IDENTf;
    shape->body.pol.pol = type == ekCONVEX_POLY ? i_convex_pol() : i_simple_pol
        ↪ ();
    col2dhello_update_pol(&shape->body.pol);
}

/*-----------------------------------------------------------------------------*/

static ArrSt(Shape) *i_shapes(void)
{
    ArrSt(Shape) *shapes = arrst_create(Shape);
    i_new_pnt(shapes, 520, 230);
    i_new_pnt(shapes, 220, 205);
    i_new_seg(shapes, 420, 280, 190, 125 * kBMATH_DEG2RADf);
    i_new_cir(shapes, 100, 100, 50);
    i_new_cir(shapes, 300, 200, 20);
    i_new_box(shapes, 100, 225, 100, 50);
    i_new_obb(shapes, 150, 350, 200, 20, 200 * kBMATH_DEG2RADf);
    i_new_obb(shapes, 460, 90, 200, 60, 15 * kBMATH_DEG2RADf);
    i_new_tri(shapes, 550, 475, 75 * kBMATH_DEG2RADf, 15);
    i_new_tri(shapes, 90, 480, 355 * kBMATH_DEG2RADf, 18);
    i_new_pol(shapes, ekCONVEX_POLY, 535, 325, 30 * kBMATH_DEG2RADf, 15);
    i_new_pol(shapes, ekSIMPLE_POLY, 370, 450, 45 * kBMATH_DEG2RADf, 7);
    return shapes;
}

/*-----------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    col2dhello_dbind();
    app->shapes = i_shapes();
    app->dists = arrst_create(Dist);
    app->seltype = ekOBB;
    app->selshape = UINT32_MAX;
    app->show_seg_pt = TRUE;
    app->show_triangles = FALSE;
    app->show_convex_parts = FALSE;
    app->sel_area = 0;
    app->window = col2dhello_window(app);
    window_title(app->window, "2D Collision Detection");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    col2dhello_dbind_shape(app);
    col2dhello_collisions(app);
    return app;
}
```

```
/*---------------------------------------------------------------------------*/

static void i_remove_bounds(Cloud *cloud)
{
    cassert_no_null(cloud);
    switch(cloud->ctype) {
    case 0:
    case 1:
    case 2:
        break;
    case 3:
        obb2d_destroyf(&cloud->bound.obb);
        break;
    case 4:
        pol2d_destroyf(&cloud->bound.poly);
        break;
    cassert_default();
    }
}

/*---------------------------------------------------------------------------*/

static void i_remove_shape(Shape *shape)
{
    cassert_no_null(shape);
    switch(shape->type){
    case ekPOINT_CLOUD:
        arrst_destroy(&shape->body.cloud.pnts, NULL, V2Df);
        i_remove_bounds(&shape->body.cloud);
        break;

    case ekOBB:
        obb2d_destroyf(&shape->body.obb.obb);
        break;

    case ekCONVEX_POLY:
    case ekSIMPLE_POLY:
        pol2d_destroyf(&shape->body.pol.pol);
        break;

    case ekPOINT:
    case ekSEGMENT:
    case ekCIRCLE:
    case ekBOX:
    case ekTRIANGLE:
        break;

    cassert_default();
    }
}
```

```
/*-----------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    arrst_destroy(&(*app)->shapes, i_remove_shape, Shape);
    arrst_destroy(&(*app)->dists, NULL, Dist);
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*-----------------------------------------------------------------------------*/

void col2dhello_new_shape(App *app, const V2Df pos)
{
    switch(app->seltype) {
    case ekPOINT:
        i_new_pnt(app->shapes, pos.x, pos.y);
        break;

    case ekPOINT_CLOUD:
        i_new_cloud(app->shapes, pos.x, pos.y, 100, 50, 15 * kBMATH_DEG2RADf);
        break;

    case ekSEGMENT:
        i_new_seg(app->shapes, pos.x, pos.y, 100, 15 * kBMATH_DEG2RADf);
        break;

    case ekCIRCLE:
        i_new_cir(app->shapes, pos.x, pos.y, 30);
        break;

    case ekBOX:
        i_new_box(app->shapes, pos.x, pos.y, 100, 50);
        break;

    case ekOBB:
        i_new_obb(app->shapes, pos.x, pos.y, 100, 50, 15 * kBMATH_DEG2RADf);
        break;

    case ekTRIANGLE:
        i_new_tri(app->shapes, pos.x, pos.y, 15 * kBMATH_DEG2RADf, 15);
        break;

    case ekCONVEX_POLY:
        i_new_pol(app->shapes, ekCONVEX_POLY, pos.x, pos.y, 0, 10);
        break;

    case ekSIMPLE_POLY:
        i_new_pol(app->shapes, ekSIMPLE_POLY, pos.x, pos.y, 0, 10);
        break;
```

```
    cassert_default();
    }

    app->selshape = arrst_size(app->shapes, Shape) - 1;
}

/*-----------------------------------------------------------------------------*/

void col2dhello_update_gui(App *app)
{
    cassert_no_null(app);
    if (app->selshape != UINT32_MAX)
    {
        Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
        switch(shape->type) {
        case ekPOINT:
        case ekPOINT_CLOUD:
        case ekSEGMENT:
            app->sel_area = 0;
            break;

        case ekCIRCLE:
            app->sel_area = cir2d_areaf(&shape->body.cir);
            break;

        case ekBOX:
            app->sel_area = box2d_areaf(&shape->body.box.box);
            break;

        case ekOBB:
            app->sel_area = obb2d_areaf(shape->body.obb.obb);
            break;

        case ekTRIANGLE:
            app->sel_area = tri2d_areaf(&shape->body.tri.tri);
            break;

        case ekCONVEX_POLY:
        case ekSIMPLE_POLY:
            app->sel_area = pol2d_areaf(shape->body.pol.pol);
            break;

        cassert_default();
        }
    }
    else
    {
        app->sel_area = 0;
    }

    layout_dbind_obj(app->main_layout, app, App);
```

```
    panel_update(app->obj_panel);
    view_update(app->view);
}

/*-----------------------------------------------------------------------------*/

void col2dhello_update_seg(Seg *seg)
{
    V2Df hvec;
    cassert_no_null(seg);
    hvec.x = seg->length / 2;
    hvec.y = 0;
    v2d_rotatef(&hvec, seg->angle);
    seg->seg.p0.x = seg->center.x - hvec.x;
    seg->seg.p0.y = seg->center.y - hvec.y;
    seg->seg.p1.x = seg->center.x + hvec.x;
    seg->seg.p1.y = seg->center.y + hvec.y;
}

/*-----------------------------------------------------------------------------*/

Box2Df col2dhello_cloud_box(const Cloud *cloud)
{
    Box2Df box = cloud->box;
    box.min = v2d_addf(&cloud->box.min, &cloud->center);
    box.max = v2d_addf(&cloud->box.max, &cloud->center);
    return box;
}

/*-----------------------------------------------------------------------------*/

void col2dhello_update_cloud(Cloud *cloud)
{
    V2Df *pt = NULL;
    uint32_t i, n;
    real32_t hw, hh;
    cassert_no_null(cloud);
    pt = arrst_all(cloud->pnts, V2Df);
    n = arrst_size(cloud->pnts, V2Df);
    hw = cloud->width / 2;
    hh = cloud->height / 2;

    for (i = 0; i < n; ++i)
    {
        real32_t ox = bmath_randf(- .3f * hw, .3f * hw);
        real32_t oy = bmath_randf(- .3f * hh, .3f * hh);
        pt[i].x = bmath_randf(-hw, hw) + ox;
        pt[i].y = bmath_randf(-hh, hh) + oy;
    }

    if (cloud->angle != 0)
```

```
    {
        T2Df t2d;
        t2d_rotatef(&t2d, kT2D_IDENTf, cloud->angle);
        t2d_vmultnf(pt, &t2d, pt, n);
    }

    cloud->box = box2d_from_pointsf(pt, n);
    col2dhello_update_cloud_bounds(cloud);
}

/*-------------------------------------------------------------------------*/

void col2dhello_update_cloud_bounds(Cloud *cloud)
{
    const V2Df *p = arrst_all(cloud->pnts, V2Df);
    uint32_t n = arrst_size(cloud->pnts, V2Df);

    i_remove_bounds(cloud);
    switch(cloud->type) {
    case 0:
        cloud->bound.cir = cir2d_from_boxf(&cloud->box);
        break;

    case 1:
        cloud->bound.cir = cir2d_from_pointsf(p, n);
        break;

    case 2:
        cloud->bound.cir = cir2d_minimumf(p, n);
        break;

    case 3:
        cloud->bound.obb = obb2d_from_pointsf(p, n);
        break;

    case 4:
        cloud->bound.poly = pol2d_convex_hullf(p, n);
        break;
    cassert_default();
    }

    cloud->ctype = cloud->type;
}

/*-------------------------------------------------------------------------*/

void col2dhello_update_box(Box *box)
{
    cassert_no_null(box);
    box->box.min.x = box->center.x - box->width / 2;
    box->box.min.y = box->center.y - box->height / 2;
```

```
    box->box.max.x = box->center.x + box->width / 2;
    box->box.max.y = box->center.y + box->height / 2;
}

/*----------------------------------------------------------------------------*/

void col2dhello_update_obb(OBB *obb)
{
    cassert_no_null(obb);
    if (obb->obb == NULL)
        obb->obb = obb2d_createf(&obb->center, obb->width, obb->height, obb->
            ↪ angle);
    else
        obb2d_updatef(obb->obb, &obb->center, obb->width, obb->height, obb->
            ↪ angle);
}

/*----------------------------------------------------------------------------*/

void col2dhello_update_tri(Tri *tri)
{
    T2Df t2d, nt2d;
    cassert_no_null(tri);
    t2d_inversef(&t2d, &tri->t2d);
    t2d_movef(&nt2d, kT2D_IDENTf, tri->center.x, tri->center.y);
    t2d_rotatef(&nt2d, &nt2d, tri->angle);
    t2d_scalef(&nt2d, &nt2d, tri->scale, tri->scale);
    t2d_multf(&t2d, &nt2d, &t2d);
    tri2d_transformf(&tri->tri, &t2d);
    tri->t2d = nt2d;
}

/*----------------------------------------------------------------------------*/

void col2dhello_update_pol(Pol *pol)
{
    T2Df t2d, nt2d;
    cassert_no_null(pol);
    cassert_no_null(pol->pol);
    t2d_inversef(&t2d, &pol->t2d);
    t2d_movef(&nt2d, kT2D_IDENTf, pol->center.x, pol->center.y);
    t2d_rotatef(&nt2d, &nt2d, pol->angle);
    t2d_scalef(&nt2d, &nt2d, pol->scale, pol->scale);
    t2d_multf(&t2d, &nt2d, &t2d);
    pol2d_transformf(pol->pol, &t2d);
    pol->t2d = nt2d;
}

/*----------------------------------------------------------------------------*/

static bool_t i_mouse_inside(const Shape *shape, const real32_t mouse_x, const
```

```
    ↪ real32_t mouse_y)
{
    V2Df m = v2df(mouse_x, mouse_y);

    switch(shape->type) {
    case ekPOINT:
        return col2d_point_pointf(&shape->body.pnt, &m, CENTER_RADIUS, NULL);

    case ekPOINT_CLOUD:
    {
        Box2Df box = col2dhello_cloud_box(&shape->body.cloud);
        return col2d_box_pointf(&box, &m, NULL);
    }

    case ekSEGMENT:
        return col2d_segment_pointf(&shape->body.seg.seg, &m, CENTER_RADIUS,
            ↪ NULL);

    case ekCIRCLE:
        return col2d_circle_pointf(&shape->body.cir, &m, NULL);

    case ekBOX:
        return col2d_box_pointf(&shape->body.box.box, &m, NULL);

    case ekOBB:
        return col2d_obb_pointf(shape->body.obb.obb, &m, NULL);

    case ekTRIANGLE:
        return col2d_tri_pointf(&shape->body.tri.tri, &m, NULL);

    case ekCONVEX_POLY:
    case ekSIMPLE_POLY:
        return col2d_poly_pointf(shape->body.pol.pol, &m, NULL);

    cassert_default();
    }

    return FALSE;
}

/*---------------------------------------------------------------------------*/

void col2dhello_mouse_collisions(App *app, const real32_t mouse_x, const
    ↪ real32_t mouse_y)
{
    arrst_foreach(shape, app->shapes, Shape)
        shape->mouse = i_mouse_inside(shape, mouse_x, mouse_y);
    arrst_end();
}

/*---------------------------------------------------------------------------*/
```

```
static void i_point_segment_dist(const Seg2Df *seg, const V2Df *pnt, ArrSt(Dist
    ↪ ) *dists)
{
    Dist *dist = arrst_new(dists, Dist);
    real32_t t = seg2d_close_paramf(seg, pnt);
    dist->p0 = *pnt;
    dist->p1 = seg2d_evalf(seg, t);
}

/*-----------------------------------------------------------------------------*/

void col2dhello_collisions(App *app)
{
    Shape *shape = arrst_all(app->shapes, Shape);
    uint32_t n = arrst_size(app->shapes, Shape);
    uint32_t i, j;

    arrst_clear(app->dists, NULL, Dist);

    for (i = 0; i < n; ++i)
        shape[i].collisions = 0;

    for (i = 0; i < n; ++i)
    for (j = i + 1; j < n; ++j)
    {
        const Shape *shape1 = shape[i].type < shape[j].type ? &shape[i] : &
            ↪ shape[j];
        const Shape *shape2 = shape[i].type < shape[j].type ? &shape[j] : &
            ↪ shape[i];
        bool_t col = FALSE;

        switch(shape1->type) {
        case ekPOINT:
            switch(shape2->type) {
            case ekPOINT:
                col = col2d_point_pointf(&shape1->body.pnt, &shape2->body.pnt,
                    ↪ CENTER_RADIUS, NULL);
                break;

            case ekPOINT_CLOUD:
                col = FALSE;
                break;

            case ekSEGMENT:
                col = col2d_segment_pointf(&shape2->body.seg.seg, &shape1->body
                    ↪ .pnt, CENTER_RADIUS, NULL);
                i_point_segment_dist(&shape2->body.seg.seg, &shape1->body.pnt,
                    ↪ app->dists);
                break;
```

```
            case ekCIRCLE:
                col = col2d_circle_pointf(&shape2->body.cir, &shape1->body.pnt,
                    ↪  NULL);
                break;

            case ekBOX:
                col = col2d_box_pointf(&shape2->body.box.box, &shape1->body.pnt
                    ↪ , NULL);
                break;

            case ekOBB:
                col = col2d_obb_pointf(shape2->body.obb.obb, &shape1->body.pnt,
                    ↪  NULL);
                break;

            case ekTRIANGLE:
                col = col2d_tri_pointf(&shape2->body.tri.tri, &shape1->body.pnt
                    ↪ , NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_pointf(shape2->body.pol.pol, &shape1->body.pnt
                    ↪ , NULL);
                break;

            cassert_default();
            }
            break;

        case ekPOINT_CLOUD:
            col = FALSE;
            break;

        case ekSEGMENT:
            switch(shape2->type) {
            case ekSEGMENT:
                col = col2d_segment_segmentf(&shape1->body.seg.seg, &shape2->
                    ↪ body.seg.seg, NULL);
                break;

            case ekCIRCLE:
                col = col2d_circle_segmentf(&shape2->body.cir, &shape1->body.
                    ↪ seg.seg, NULL);
                break;

            case ekBOX:
                col = col2d_box_segmentf(&shape2->body.box.box, &shape1->body.
                    ↪ seg.seg, NULL);
                break;
```

```
            case ekOBB:
                col = col2d_obb_segmentf(shape2->body.obb.obb, &shape1->body.
                    ↪ seg.seg, NULL);
                break;

            case ekTRIANGLE:
                col = col2d_tri_segmentf(&shape2->body.tri.tri, &shape1->body.
                    ↪ seg.seg, NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_segmentf(shape2->body.pol.pol, &shape1->body.
                    ↪ seg.seg, NULL);
                break;

            case ekPOINT:
            case ekPOINT_CLOUD:
            cassert_default();
            }
            break;

        case ekCIRCLE:
            switch(shape2->type) {
            case ekCIRCLE:
                col = col2d_circle_circlef(&shape1->body.cir, &shape2->body.cir
                    ↪ , NULL);
                break;

            case ekBOX:
                col = col2d_box_circlef(&shape2->body.box.box, &shape1->body.
                    ↪ cir, NULL);
                break;

            case ekOBB:
                col = col2d_obb_circlef(shape2->body.obb.obb, &shape1->body.cir
                    ↪ , NULL);
                break;

            case ekTRIANGLE:
                col = col2d_tri_circlef(&shape2->body.tri.tri, &shape1->body.
                    ↪ cir, NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_circlef(shape2->body.pol.pol, &shape1->body.
                    ↪ cir, NULL);
                break;

            case ekPOINT:
```

```
            case ekPOINT_CLOUD:
            case ekSEGMENT:
            cassert_default();
            }
            break;

        case ekBOX:
            switch(shape2->type) {
            case ekBOX:
                col = col2d_box_boxf(&shape1->body.box.box, &shape2->body.box.
                    ↪ box, NULL);
                break;

            case ekOBB:
                col = col2d_obb_boxf(shape2->body.obb.obb, &shape1->body.box.
                    ↪ box, NULL);
                break;

            case ekTRIANGLE:
                col = col2d_tri_boxf(&shape2->body.tri.tri, &shape1->body.box.
                    ↪ box, NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_boxf(shape2->body.pol.pol, &shape1->body.box.
                    ↪ box, NULL);
                break;

            case ekPOINT:
            case ekPOINT_CLOUD:
            case ekSEGMENT:
            case ekCIRCLE:
            cassert_default();
            }
            break;

        case ekOBB:
            switch(shape2->type) {
            case ekOBB:
                col = col2d_obb_obbf(shape1->body.obb.obb, shape2->body.obb.obb
                    ↪ , NULL);
                break;

            case ekTRIANGLE:
                col = col2d_tri_obbf(&shape2->body.tri.tri, shape1->body.obb.
                    ↪ obb, NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
```

```
                col = col2d_poly_obbf(shape2->body.pol.pol, shape1->body.obb.
                    ↪ obb, NULL);
                break;

            case ekPOINT:
            case ekPOINT_CLOUD:
            case ekSEGMENT:
            case ekCIRCLE:
            case ekBOX:
            cassert_default();
            }
            break;

        case ekTRIANGLE:
            switch(shape2->type) {
            case ekTRIANGLE:
                col = col2d_tri_trif(&shape1->body.tri.tri, &shape2->body.tri.
                    ↪ tri, NULL);
                break;

            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_trif(shape2->body.pol.pol, &shape1->body.tri.
                    ↪ tri, NULL);
                break;

            case ekPOINT:
            case ekPOINT_CLOUD:
            case ekSEGMENT:
            case ekCIRCLE:
            case ekBOX:
            case ekOBB:
            cassert_default();
            }
            break;

        case ekCONVEX_POLY:
        case ekSIMPLE_POLY:
            switch(shape2->type) {
            case ekCONVEX_POLY:
            case ekSIMPLE_POLY:
                col = col2d_poly_polyf(shape1->body.pol.pol, shape2->body.pol.
                    ↪ pol, NULL);
                break;

            case ekPOINT:
            case ekPOINT_CLOUD:
            case ekSEGMENT:
            case ekCIRCLE:
            case ekBOX:
            case ekOBB:
```

```
            case ekTRIANGLE:
            cassert_default();
            }
            break;

        cassert_default();
        }

        if (col == TRUE)
        {
            shape[i].collisions += 1;
            shape[j].collisions += 1;
        }
    }
}

/*---------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Listado 28.2: demo/col2dhello/col2dhello.hxx

```
/* 2D collision detection demo */

#ifndef __COL2DHELLO_HXX__
#define __COL2DHELLO_HXX__

#include <gui/gui.hxx>

#define CENTER_RADIUS       3
#define POINT_CLOUD_N       100

typedef struct _cloud_t Cloud;
typedef struct _seg_t Seg;
typedef struct _box_t Box;
typedef struct _obb_t OBB;
typedef struct _tri_t Tri;
typedef struct _pol_t Pol;
typedef struct _shape_t Shape;
typedef struct _dist_t Dist;
typedef struct _app_t App;

typedef enum _shtype_t
{
    ekPOINT,
    ekPOINT_CLOUD,
    ekSEGMENT,
    ekCIRCLE,
    ekBOX,
    ekOBB,
```

```
    ekTRIANGLE,
    ekCONVEX_POLY,
    ekSIMPLE_POLY
} shtype_t;

struct _cloud_t
{
    ArrSt(V2Df) *pnts;
    Box2Df box;
    V2Df center;
    real32_t width;
    real32_t height;
    real32_t angle;
    uint32_t ctype, type;

    union
    {
        Cir2Df cir;
        OBB2Df *obb;
        Pol2Df *poly;
    } bound;
};

struct _seg_t
{
    V2Df center;
    real32_t length;
    real32_t angle;
    Seg2Df seg;
};

struct _box_t
{
    V2Df center;
    real32_t width;
    real32_t height;
    Box2Df box;
};

struct _obb_t
{
    V2Df center;
    real32_t width;
    real32_t height;
    real32_t angle;
    OBB2Df *obb;
};

struct _tri_t
{
    V2Df center;
```

```
    real32_t angle;
    real32_t scale;
    T2Df t2d;
    Tri2Df tri;
};

struct _pol_t
{
    V2Df center;
    real32_t angle;
    real32_t scale;
    T2Df t2d;
    Pol2Df *pol;
};

struct _shape_t
{
    shtype_t type;
    bool_t mouse;
    uint32_t collisions;

    union {
        V2Df pnt;
        Cloud cloud;
        Seg seg;
        Cir2Df cir;
        Box box;
        OBB obb;
        Tri tri;
        Pol pol;
    } body;
};

struct _dist_t
{
    V2Df p0;
    V2Df p1;
};

struct _app_t
{
    Window *window;
    View *view;
    Layout *main_layout;
    Layout *pnt_layout;
    Layout *cld_layout;
    Layout *seg_layout;
    Layout *cir_layout;
    Layout *box_layout;
    Layout *obb_layout;
    Layout *tri_layout;
```

```
    Layout *pol_layout;
    Panel *obj_panel;
    ArrSt(Shape) *shapes;
    ArrSt(Dist) *dists;
    shtype_t seltype;
    uint32_t selshape;
    bool_t show_seg_pt;
    bool_t show_triangles;
    bool_t show_convex_parts;
    real32_t sel_area;
    V2Df mouse_pos;
    V2Df obj_pos;
};

DeclSt(Shape);
DeclSt(Dist);

#endif
```

Listado 28.3: demo/col2dhello/col2dgui.c

```
/* Col2D Hello GUI */

#include "col2dgui.h"
#include <nappgui.h>

/*---------------------------------------------------------------------------*/

void col2dhello_dbind(void)
{
    dbind_enum(shtype_t, ekPOINT, "");
    dbind_enum(shtype_t, ekPOINT_CLOUD, "");
    dbind_enum(shtype_t, ekSEGMENT, "");
    dbind_enum(shtype_t, ekCIRCLE, "");
    dbind_enum(shtype_t, ekBOX, "");
    dbind_enum(shtype_t, ekOBB, "");
    dbind_enum(shtype_t, ekTRIANGLE, "");
    dbind_enum(shtype_t, ekCONVEX_POLY, "");
    dbind_enum(shtype_t, ekSIMPLE_POLY, "");
    dbind(App, shtype_t, seltype);
    dbind(App, bool_t, show_seg_pt);
    dbind(App, bool_t, show_triangles);
    dbind(App, bool_t, show_convex_parts);
    dbind(App, real32_t, sel_area);
    dbind(Cloud, real32_t, width);
    dbind(Cloud, real32_t, height);
    dbind(Cloud, real32_t, angle);
    dbind(Cloud, uint32_t, type);
    dbind(Seg, real32_t, length);
    dbind(Seg, real32_t, angle);
    dbind(Cir2Df, real32_t, r);
```

```
    dbind(Box, real32_t, width);
    dbind(Box, real32_t, height);
    dbind(OBB, real32_t, width);
    dbind(OBB, real32_t, height);
    dbind(OBB, real32_t, angle);
    dbind(Tri, real32_t, angle);
    dbind(Tri, real32_t, scale);
    dbind(Pol, real32_t, angle);
    dbind(Pol, real32_t, scale);
    dbind_range(Cloud, real32_t, width, 50, 200);
    dbind_range(Cloud, real32_t, height, 50, 200);
    dbind_range(Cloud, real32_t, angle, 0, 360 * kBMATH_DEG2RADf);
    dbind_range(Seg, real32_t, length, 20, 300);
    dbind_range(Seg, real32_t, angle, 0, 360 * kBMATH_DEG2RADf);
    dbind_range(Cir2Df, real32_t, r, 5, 100);
    dbind_range(Box, real32_t, width, 20, 300);
    dbind_range(Box, real32_t, height, 20, 300);
    dbind_range(OBB, real32_t, width, 20, 300);
    dbind_range(OBB, real32_t, height, .2f, 300);
    dbind_range(OBB, real32_t, angle, 0, 360 * kBMATH_DEG2RADf);
    dbind_range(Tri, real32_t, angle, 0, 360 * kBMATH_DEG2RADf);
    dbind_range(Tri, real32_t, scale, 5, 30);
    dbind_range(Pol, real32_t, angle, 0, 360 * kBMATH_DEG2RADf);
    dbind_range(Pol, real32_t, scale, 5, 30);
}

/*-----------------------------------------------------------------------------*/

static void i_OnCloud(App *app, Event *e)
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekPOINT_CLOUD);

    if (evbind_modify(e, Cloud, uint32_t, type) == TRUE)
        col2dhello_update_cloud_bounds(&shape->body.cloud);
    else
        col2dhello_update_cloud(&shape->body.cloud);

    col2dhello_collisions(app);
    col2dhello_update_gui(app);
}

/*-----------------------------------------------------------------------------*/

static void i_OnSeg(App *app, Event *e)
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekSEGMENT);
    col2dhello_update_seg(&shape->body.seg);
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
```

```
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnCircle(App *app, Event *e)
{
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnBox(App *app, Event *e)
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekBOX);
    col2dhello_update_box(&shape->body.box);
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnOBB(App *app, Event *e)
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekOBB);
    col2dhello_update_obb(&shape->body.obb);
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnTri(App *app, Event *e)
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekTRIANGLE);
    col2dhello_update_tri(&shape->body.tri);
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnPoly(App *app, Event *e)
```

```
{
    Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
    cassert(shape->type == ekCONVEX_POLY || shape->type == ekSIMPLE_POLY);
    col2dhello_update_pol(&shape->body.pol);
    col2dhello_collisions(app);
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnOpt(App *app, Event *e)
{
    col2dhello_update_gui(app);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static Layout *i_empty_layout(void)
{
    Layout *layout = layout_create(1, 1);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_point_layout(App *app)
{
    Layout *layout = layout_create(1, 1);
    Label *label = label_create();
    label_text(label, "Selected Point");
    layout_label(layout, label, 0, 0);
    app->pnt_layout = layout;
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_bounding_layout(void)
{
    Layout *layout = layout_create(1, 5);
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    Button *button3 = button_radio();
    Button *button4 = button_radio();
    Button *button5 = button_radio();
    button_text(button1, "BBox Circle");
    button_text(button2, "Points Circle");
    button_text(button3, "Minimum Circle");
    button_text(button4, "Gaussian OBB");
```

```
    button_text(button5, "Convex Hull");
    layout_button(layout, button1, 0, 0);
    layout_button(layout, button2, 0, 1);
    layout_button(layout, button3, 0, 2);
    layout_button(layout, button4, 0, 3);
    layout_button(layout, button5, 0, 4);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 1, 5);
    layout_vmargin(layout, 2, 5);
    layout_vmargin(layout, 3, 5);
    cell_dbind(layout_cell(layout, 0, 0), Cloud, uint32_t, type);
    return layout;
}

/*----------------------------------------------------------------------------*/

static Layout *i_cloud_layout(App *app)
{
    Layout *layout1 = layout_create(1, 9);
    Layout *layout2 = i_bounding_layout();
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Label *label5 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    Slider *slider3 = slider_create();
    label_text(label1, "Selected Point Cloud");
    label_text(label2, "Width:");
    label_text(label3, "Height:");
    label_text(label4, "Angle:");
    label_text(label5, "Bounding Volume");
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 0, 1);
    layout_label(layout1, label3, 0, 3);
    layout_label(layout1, label4, 0, 5);
    layout_label(layout1, label5, 0, 7);
    layout_slider(layout1, slider1, 0, 2);
    layout_slider(layout1, slider2, 0, 4);
    layout_slider(layout1, slider3, 0, 6);
    layout_layout(layout1, layout2, 0, 8);
    layout_vmargin(layout1, 0, 5);
    layout_vmargin(layout1, 2, 10);
    layout_vmargin(layout1, 4, 10);
    layout_vmargin(layout1, 6, 5);
    layout_vmargin(layout1, 7, 8);
    cell_dbind(layout_cell(layout1, 0, 2), Cloud, real32_t, width);
    cell_dbind(layout_cell(layout1, 0, 4), Cloud, real32_t, height);
    cell_dbind(layout_cell(layout1, 0, 6), Cloud, real32_t, angle);
    layout_dbind(layout1, listener(app, i_OnCloud, App), Cloud);
```

```
    app->cld_layout = layout1;
    return layout1;
}

/*---------------------------------------------------------------------------*/

static Layout *i_segment_layout(App *app)
{
    Layout *layout = layout_create(1, 5);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    label_text(label1, "Selected Segment");
    label_text(label2, "Length:");
    label_text(label3, "Angle:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 3);
    layout_slider(layout, slider1, 0, 2);
    layout_slider(layout, slider2, 0, 4);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 2, 10);
    cell_dbind(layout_cell(layout, 0, 2), Seg, real32_t, length);
    cell_dbind(layout_cell(layout, 0, 4), Seg, real32_t, angle);
    layout_dbind(layout, listener(app, i_OnSeg, App), Seg);
    app->seg_layout = layout;
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_circle_layout(App *app)
{
    Layout *layout = layout_create(1, 3);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Slider *slider = slider_create();
    label_text(label1, "Selected Circle");
    label_text(label2, "Radix:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_slider(layout, slider, 0, 2);
    layout_vmargin(layout, 0, 5);
    cell_dbind(layout_cell(layout, 0, 2), Cir2Df, real32_t, r);
    layout_dbind(layout, listener(app, i_OnCircle, App), Cir2Df);
    app->cir_layout = layout;
    return layout;
}
```

```
/*-----------------------------------------------------------------------------*/

static Layout *i_box_layout(App *app)
{
    Layout *layout = layout_create(1, 5);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    label_text(label1, "Selected Box");
    label_text(label2, "Width:");
    label_text(label3, "Height:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 3);
    layout_slider(layout, slider1, 0, 2);
    layout_slider(layout, slider2, 0, 4);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 2, 10);
    cell_dbind(layout_cell(layout, 0, 2), Box, real32_t, width);
    cell_dbind(layout_cell(layout, 0, 4), Box, real32_t, height);
    layout_dbind(layout, listener(app, i_OnBox, App), Box);
    app->box_layout = layout;
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_obb_layout(App *app)
{
    Layout *layout = layout_create(1, 7);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    Slider *slider3 = slider_create();
    label_text(label1, "Selected Oriented Box");
    label_text(label2, "Width:");
    label_text(label3, "Height:");
    label_text(label4, "Angle:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 3);
    layout_label(layout, label4, 0, 5);
    layout_slider(layout, slider1, 0, 2);
    layout_slider(layout, slider2, 0, 4);
    layout_slider(layout, slider3, 0, 6);
    layout_vmargin(layout, 0, 5);
```

```
    layout_vmargin(layout, 2, 10);
    layout_vmargin(layout, 4, 10);
    cell_dbind(layout_cell(layout, 0, 2), OBB, real32_t, width);
    cell_dbind(layout_cell(layout, 0, 4), OBB, real32_t, height);
    cell_dbind(layout_cell(layout, 0, 6), OBB, real32_t, angle);
    layout_dbind(layout, listener(app, i_OnOBB, App), OBB);
    app->obb_layout = layout;
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_tri_layout(App *app)
{
    Layout *layout = layout_create(1, 5);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    label_text(label1, "Selected Triangle");
    label_text(label2, "Angle:");
    label_text(label3, "Scale:");
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 3);
    layout_slider(layout, slider1, 0, 2);
    layout_slider(layout, slider2, 0, 4);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 2, 10);
    cell_dbind(layout_cell(layout, 0, 2), Tri, real32_t, angle);
    cell_dbind(layout_cell(layout, 0, 4), Tri, real32_t, scale);
    layout_dbind(layout, listener(app, i_OnTri, App), Tri);
    app->tri_layout = layout;
    return layout;
}

/*---------------------------------------------------------------------------*/

static Layout *i_pol_layout(App *app)
{
    Layout *layout = layout_create(1, 5);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    label_text(label1, "Selected Polygon");
    label_text(label2, "Angle:");
    label_text(label3, "Scale:");
    layout_label(layout, label1, 0, 0);
```

```
    layout_label(layout, label2, 0, 1);
    layout_label(layout, label3, 0, 3);
    layout_slider(layout, slider1, 0, 2);
    layout_slider(layout, slider2, 0, 4);
    layout_vmargin(layout, 0, 5);
    layout_vmargin(layout, 2, 10);
    cell_dbind(layout_cell(layout, 0, 2), Pol, real32_t, angle);
    cell_dbind(layout_cell(layout, 0, 4), Pol, real32_t, scale);
    layout_dbind(layout, listener(app, i_OnPoly, App), Pol);
    app->pol_layout = layout;
    return layout;
}

/*-----------------------------------------------------------------------------*/

static void i_OnNewShape(App *app, Event *e)
{
    S2Df size;
    view_get_size(app->view, &size);
    col2dhello_new_shape(app, v2df(size.width / 2, size.height / 2));
    col2dhello_dbind_shape(app);
    col2dhello_collisions(app);
    view_update(app->view);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static Layout *i_new_layout(App *app)
{
    Layout *layout = layout_create(1, 2);
    PopUp *popup = popup_create();
    Button *button = button_push();
    button_text(button, "New Shape");
    button_OnClick(button, listener(app, i_OnNewShape, App));
    layout_popup(layout, popup, 0, 0);
    layout_button(layout, button, 0, 1);
    layout_vmargin(layout, 0, 5);
    cell_dbind(layout_cell(layout, 0, 0), App, shtype_t, seltype);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_area_layout(void)
{
    Layout *layout = layout_create(2, 1);
    Label *label1 = label_create();
    Label *label2 = label_create();
    label_text(label1, "Area:");
    layout_label(layout, label1, 0, 0);
```

```
    layout_label(layout, label2, 1, 0);
    layout_hmargin(layout, 0, 5);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_hexpand(layout, 1);
    cell_dbind(layout_cell(layout, 1, 0), App, real32_t, sel_area);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_left_layout(App *app)
{
    Layout *layout1 = layout_create(1, 6);
    Layout *layout2 = i_new_layout(app);
    Layout *layout3 = i_area_layout();
    Layout *layout4 = i_empty_layout();
    Layout *layout5 = i_point_layout(app);
    Layout *layout6 = i_cloud_layout(app);
    Layout *layout7 = i_segment_layout(app);
    Layout *layout8 = i_circle_layout(app);
    Layout *layout9 = i_box_layout(app);
    Layout *layout10 = i_obb_layout(app);
    Layout *layout11 = i_tri_layout(app);
    Layout *layout12 = i_pol_layout(app);
    Button *button1 = button_check();
    Button *button2 = button_check();
    Button *button3 = button_check();
    Panel *panel = panel_create();
    button_text(button1, "Show Segment-Point distance");
    button_text(button2, "Show Polygon triangles");
    button_text(button3, "Show Convex partition");
    panel_layout(panel, layout4);
    panel_layout(panel, layout5);
    panel_layout(panel, layout6);
    panel_layout(panel, layout7);
    panel_layout(panel, layout8);
    panel_layout(panel, layout9);
    panel_layout(panel, layout10);
    panel_layout(panel, layout11);
    panel_layout(panel, layout12);
    layout_layout(layout1, layout2, 0, 0);
    layout_button(layout1, button1, 0, 1);
    layout_button(layout1, button2, 0, 2);
    layout_button(layout1, button3, 0, 3);
    layout_layout(layout1, layout3, 0, 4);
    layout_panel(layout1, panel, 0, 5);
    layout_vmargin(layout1, 0, 10);
    layout_vmargin(layout1, 1, 5);
    layout_vmargin(layout1, 2, 5);
    layout_vmargin(layout1, 3, 5);
    layout_vmargin(layout1, 4, 10);
```

```
    layout_margin(layout1, 10);
    app->obj_panel = panel;
    app->main_layout = layout1;
    cell_dbind(layout_cell(layout1, 0, 1), App, bool_t, show_seg_pt);
    cell_dbind(layout_cell(layout1, 0, 2), App, bool_t, show_triangles);
    cell_dbind(layout_cell(layout1, 0, 3), App, bool_t, show_convex_parts);
    layout_dbind(layout1, listener(app, i_OnOpt, App), App);
    layout_dbind_obj(layout1, app, App);
    return layout1;
}

/*-----------------------------------------------------------------------*/

static color_t i_color(const uint32_t collision, const bool_t mouse)
{
    if (collision > 0)
    {
        if (collision == 1)
            return color_rgb(255, 170, 0);

        if (collision == 2)
            return color_rgb(255, 127, 0);

        return color_rgb(255, 42, 0);
    }
    else
    {
        if (mouse == TRUE)
            return color_rgb(127, 85, 255);

        return color_gray(120);
    }
}

/*-----------------------------------------------------------------------*/

static void i_draw_point(DCtx *ctx, const V2Df *pt)
{
    draw_v2df(ctx, ekFILL, pt, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------*/

static void i_draw_cloud(DCtx *ctx, const Cloud *cloud)
{
    arrst_foreach(pt, cloud->pnts, V2Df)
        draw_circle(ctx, ekSTROKE, pt->x + cloud->center.x, pt->y + cloud->
            ↪ center.y, 1);
    arrst_end();

    switch(cloud->type) {
```

```
    case 0:
    case 1:
    case 2:
    {
        real32_t cx = cloud->bound.cir.c.x + cloud->center.x;
        real32_t cy = cloud->bound.cir.c.y + cloud->center.y;
        draw_circle(ctx, ekSTROKE, cx, cy, cloud->bound.cir.r);
        draw_fill_color(ctx, kCOLOR_BLACK);
        draw_circle(ctx, ekFILL, cx, cy, CENTER_RADIUS);
        break;
    }

    case 3:
    {
        T2Df t2d;
        V2Df center = obb2d_centerf(cloud->bound.obb);
        t2d_movef(&t2d, kT2D_IDENTf, cloud->center.x, cloud->center.y);
        draw_matrixf(ctx, &t2d);
        draw_obb2df(ctx, ekSTROKE, cloud->bound.obb);
        draw_fill_color(ctx, kCOLOR_BLACK);
        draw_circle(ctx, ekFILL, center.x, center.y, CENTER_RADIUS);
        draw_matrixf(ctx, kT2D_IDENTf);
        break;
    }

    case 4:
    {
        T2Df t2d;
        V2Df center = pol2d_centroidf(cloud->bound.poly);
        t2d_movef(&t2d, kT2D_IDENTf, cloud->center.x, cloud->center.y);
        draw_matrixf(ctx, &t2d);
        draw_pol2df(ctx, ekSTROKE, cloud->bound.poly);
        draw_fill_color(ctx, kCOLOR_BLACK);
        draw_circle(ctx, ekFILL, center.x, center.y, CENTER_RADIUS);
        draw_matrixf(ctx, kT2D_IDENTf);
        break;
    }

    cassert_default();
    }
}

/*---------------------------------------------------------------------------*/

static void i_draw_segment(DCtx *ctx, const Seg *seg)
{
    draw_seg2df(ctx, &seg->seg);
}

/*---------------------------------------------------------------------------*/
```

```
static void i_draw_circle(DCtx *ctx, const Cir2Df *circle)
{
    draw_cir2df(ctx, ekFILL, circle);
    draw_fill_color(ctx, kCOLOR_BLACK);
    draw_circle(ctx, ekFILL, circle->c.x, circle->c.y, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_box(DCtx *ctx, const Box *box)
{
    draw_box2df(ctx, ekFILL, &box->box);
    draw_fill_color(ctx, kCOLOR_BLACK);
    draw_circle(ctx, ekFILL, box->center.x, box->center.y, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_obb(DCtx *ctx, const OBB *obb)
{
    draw_obb2df(ctx, ekFILL, obb->obb);
    draw_fill_color(ctx, kCOLOR_BLACK);
    draw_circle(ctx, ekFILL, obb->center.x, obb->center.y, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_tri(DCtx *ctx, const Tri *tri)
{
    V2Df center = tri2d_centroidf(&tri->tri);
    draw_tri2df(ctx, ekFILL, &tri->tri);
    draw_fill_color(ctx, kCOLOR_BLACK);
    draw_circle(ctx, ekFILL, center.x, center.y, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_poly(DCtx *ctx, const Pol *pol)
{
    V2Df center = pol2d_visual_centerf(pol->pol, .05f);
    draw_pol2df(ctx, ekFILL, pol->pol);
    draw_fill_color(ctx, kCOLOR_BLACK);
    draw_circle(ctx, ekFILL, center.x, center.y, CENTER_RADIUS);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_poly_triangles(DCtx *ctx, const Pol2Df *poly)
{
    ArrSt(Tri2Df) *triangles = pol2d_trianglesf(poly);
    bool_t ccw = pol2d_ccwf(poly);
```

```
    arrst_foreach(tri, triangles, Tri2Df)
        cassert_unref(tri2d_ccwf(tri) == ccw, ccw);
        draw_tri2df(ctx, ekSTROKE, tri);
    arrst_end();
    arrst_destroy(&triangles, NULL, Tri2Df);
}

/*-------------------------------------------------------------------------*/

static void i_draw_poly_convex_parts(DCtx *ctx, const Pol2Df *poly)
{
    ArrPt(Pol2Df) *convex_polys = pol2d_convex_partitionf(poly);
    bool_t ccw = pol2d_ccwf(poly);

    arrpt_foreach(convex, convex_polys, Pol2Df)
        cassert(pol2d_convexf(convex) == TRUE);
        cassert_unref(pol2d_ccwf(convex) == ccw, ccw);
        draw_pol2df(ctx, ekSTROKE, convex);
    arrpt_end();

    arrpt_destroy(&convex_polys, pol2d_destroyf, Pol2Df);
}

/*-------------------------------------------------------------------------*/

static void i_draw_bbox(DCtx *ctx, const Shape *shape)
{
    Box2Df bbox = kBOX2D_NULLf;
    real32_t p[2] = {2, 2};
    switch(shape->type) {
    case ekPOINT:
    {
        Cir2Df c = cir2df(shape->body.pnt.x, shape->body.pnt.y, CENTER_RADIUS);
        box2d_add_circlef(&bbox, &c);
        break;
    }

    case ekPOINT_CLOUD:
        bbox = col2dhello_cloud_box(&shape->body.cloud);
        break;

    case ekSEGMENT:
        box2d_addf(&bbox, &shape->body.seg.seg.p0);
        box2d_addf(&bbox, &shape->body.seg.seg.p1);
        break;

    case ekCIRCLE:
        box2d_add_circlef(&bbox, &shape->body.cir);
        break;
```

```
    case ekBOX:
        box2d_mergef(&bbox, &shape->body.box.box);
        break;

    case ekOBB:
    {
        const V2Df *corners = obb2d_cornersf(shape->body.obb.obb);
        box2d_addnf(&bbox, corners, 4);
        break;
    }

    case ekTRIANGLE:
    {
        const V2Df *points = (const V2Df*)&shape->body.tri.tri;
        box2d_addnf(&bbox, points, 3);
        break;
    }

    case ekCONVEX_POLY:
    case ekSIMPLE_POLY:
    {
        const V2Df *points = pol2d_pointsf(shape->body.pol.pol);
        uint32_t n = pol2d_nf(shape->body.pol.pol);
        box2d_addnf(&bbox, points, n);
        break;
    }

    cassert_default();
    }

    draw_line_color(ctx, color_rgb(0, 128, 0));
    draw_line_dash(ctx, p, 2);
    draw_box2df(ctx, ekSTROKE, &bbox);
    draw_line_dash(ctx, NULL, 0);
}

/*-----------------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);
    real32_t dash[2] = {2,2};
    draw_clear(p->ctx, color_rgb(255, 212, 255));

    arrst_foreach(shape, app->shapes, Shape)
        draw_fill_color(p->ctx, i_color(shape->collisions, shape->mouse));
        draw_line_color(p->ctx, i_color(shape->collisions, shape->mouse));

        switch(shape->type) {
        case ekPOINT:
            i_draw_point(p->ctx, &shape->body.pnt);
```

```
            break;

        case ekPOINT_CLOUD:
            i_draw_cloud(p->ctx, &shape->body.cloud);
            break;

        case ekSEGMENT:
            i_draw_segment(p->ctx, &shape->body.seg);
            break;

        case ekCIRCLE:
            i_draw_circle(p->ctx, &shape->body.cir);
            break;

        case ekBOX:
            i_draw_box(p->ctx, &shape->body.box);
            break;

        case ekOBB:
            i_draw_obb(p->ctx, &shape->body.obb);
            break;

        case ekTRIANGLE:
            i_draw_tri(p->ctx, &shape->body.tri);
            break;

        case ekCONVEX_POLY:
        case ekSIMPLE_POLY:
            i_draw_poly(p->ctx, &shape->body.pol);
            break;

        cassert_default();
        }

        if (app->selshape == shape_i)
            i_draw_bbox(p->ctx, shape);

    arrst_end()

    if (app->show_seg_pt == TRUE)
    {
        real32_t pattern[2] = {2, 2};
        draw_line_dash(p->ctx, pattern, 2);
        draw_line_color(p->ctx, kCOLOR_MAGENTA);
        arrst_foreach(dist, app->dists, Dist)
            draw_line(p->ctx, dist->p0.x, dist->p0.y, dist->p1.x, dist->p1.y);
        arrst_end();
    }

    draw_line_width(p->ctx, 1);
    draw_line_color(p->ctx, kCOLOR_BLACK);
```

```
    draw_line_dash(p->ctx, dash, 2);

    if (app->show_triangles == TRUE)
    {
        arrst_foreach(shape, app->shapes, Shape)
            if (shape->type == ekCONVEX_POLY || shape->type == ekSIMPLE_POLY)
                i_draw_poly_triangles(p->ctx, shape->body.pol.pol);
        arrst_end();
    }

    if (app->show_triangles == FALSE && app->show_convex_parts == TRUE)
    {
        arrst_foreach(shape, app->shapes, Shape)
            if (shape->type == ekSIMPLE_POLY)
                i_draw_poly_convex_parts(p->ctx, shape->body.pol.pol);
        arrst_end();
    }

    draw_line_dash(p->ctx, NULL, 2);
}

/*-----------------------------------------------------------------------------*/

static void i_OnMove(App *app, Event *e)
{
    const EvMouse *p = event_params(e, EvMouse);
    View *view = event_sender(e, View);
    col2dhello_mouse_collisions(app, p->x, p->y);
    view_update(view);
}

/*-----------------------------------------------------------------------------*/

static void i_get_shape_pos(const Shape *shape, V2Df *pos)
{
    switch(shape->type) {
    case ekPOINT:
        *pos = shape->body.pnt;
        break;

    case ekPOINT_CLOUD:
        *pos = shape->body.cloud.center;
        break;

    case ekSEGMENT:
        *pos = shape->body.seg.center;
        break;

    case ekCIRCLE:
        *pos = shape->body.cir.c;
        break;
```

```
    case ekBOX:
        *pos = shape->body.box.center;
        break;

    case ekOBB:
        *pos = shape->body.obb.center;
        break;

    case ekTRIANGLE:
        *pos = shape->body.tri.center;
        *pos = shape->body.tri.center;
        break;

    case ekCONVEX_POLY:
    case ekSIMPLE_POLY:
        *pos = shape->body.pol.center;
        break;

    cassert_default();
    }
}

/*---------------------------------------------------------------------------*/

static void i_set_shape_pos(Shape *shape, const V2Df pos)
{
    switch(shape->type) {
    case ekPOINT:
        shape->body.pnt = pos;
        break;

    case ekPOINT_CLOUD:
        shape->body.cloud.center = pos;
        break;

    case ekSEGMENT:
        shape->body.seg.center = pos;
        col2dhello_update_seg(&shape->body.seg);
        break;

    case ekCIRCLE:
        shape->body.cir.c = pos;
        break;

    case ekBOX:
        shape->body.box.center = pos;
        col2dhello_update_box(&shape->body.box);
        break;

    case ekOBB:
```

```
        shape->body.obb.center = pos;
        col2dhello_update_obb(&shape->body.obb);
        break;

    case ekTRIANGLE:
        shape->body.tri.center = pos;
        col2dhello_update_tri(&shape->body.tri);
        break;

    case ekCONVEX_POLY:
    case ekSIMPLE_POLY:
        shape->body.pol.center = pos;
        col2dhello_update_pol(&shape->body.pol);
        break;

    cassert_default();
    }
}

/*-----------------------------------------------------------------------------*/

static void i_OnDown(App *app, Event *e)
{
    uint32_t selshape = UINT32_MAX;
    arrst_foreach(shape, app->shapes, Shape)
        if (shape->mouse == TRUE)
        {
            selshape = shape_i;
            break;
        }
    arrst_end();

    if (selshape != app->selshape)
    {
        View *view = event_sender(e, View);
        app->selshape = selshape;
        col2dhello_dbind_shape(app);
        view_update(view);
    }

    if (app->selshape != UINT32_MAX)
    {
        const EvMouse *p = event_params(e, EvMouse);
        const Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
        app->mouse_pos.x = p->x;
        app->mouse_pos.y = p->y;
        i_get_shape_pos(shape, &app->obj_pos);
    }
}

/*-----------------------------------------------------------------------------*/
```

```c
static void i_OnDrag(App *app, Event *e)
{
    if (app->selshape != UINT32_MAX)
    {
        const EvMouse *p = event_params(e, EvMouse);
        Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
        V2Df move = v2df(app->obj_pos.x + (p->x - app->mouse_pos.x), app->
            ↪ obj_pos.y + (p->y - app->mouse_pos.y));
        i_set_shape_pos(shape, move);
        col2dhello_collisions(app);
        view_update(app->view);
    }
}

/*-----------------------------------------------------------------------------*/

static Layout *i_layout(App *app)
{
    Layout *layout1 = layout_create(2, 1);
    Layout *layout2 = i_left_layout(app);
    View *view = view_create();
    view_size(view, s2df(640, 580));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    view_OnMove(view, listener(app, i_OnMove, App));
    view_OnDown(view, listener(app, i_OnDown, App));
    view_OnDrag(view, listener(app, i_OnDrag, App));
    layout_layout(layout1, layout2, 0, 0);
    layout_view(layout1, view, 1, 0);
    layout_valign(layout1, 0, 0, ekTOP);
    layout_hexpand(layout1, 1);
    app->view = view;
    return layout1;
}

/*-----------------------------------------------------------------------------*/

Window *col2dhello_window(App *app)
{
    Panel *panel = panel_create();
    Layout *layout = i_layout(app);
    Window *window = window_create(ekWINDOW_STDRES);
    panel_layout(panel, layout);
    window_panel(window, panel);
    return window;
}

/*-----------------------------------------------------------------------------*/

void col2dhello_dbind_shape(App *app)
{
```

```
    if (app->selshape != UINT32_MAX)
    {
        Shape *shape = arrst_get(app->shapes, app->selshape, Shape);
        switch(shape->type) {
        case ekPOINT:
            panel_visible_layout(app->obj_panel, 1);
            app->sel_area = 0;
            break;

        case ekPOINT_CLOUD:
            layout_dbind_obj(app->cld_layout, &shape->body.cloud, Cloud);
            panel_visible_layout(app->obj_panel, 2);
            app->sel_area = 0;
            break;

        case ekSEGMENT:
            layout_dbind_obj(app->seg_layout, &shape->body.seg, Seg);
            panel_visible_layout(app->obj_panel, 3);
            app->sel_area = 0;
            break;

        case ekCIRCLE:
            layout_dbind_obj(app->cir_layout, &shape->body.cir, Cir2Df);
            panel_visible_layout(app->obj_panel, 4);
            app->sel_area = cir2d_areaf(&shape->body.cir);
            break;

        case ekBOX:
            layout_dbind_obj(app->box_layout, &shape->body.box, Box);
            panel_visible_layout(app->obj_panel, 5);
            break;

        case ekOBB:
            layout_dbind_obj(app->obb_layout, &shape->body.obb, OBB);
            panel_visible_layout(app->obj_panel, 6);
            break;

        case ekTRIANGLE:
            layout_dbind_obj(app->tri_layout, &shape->body.tri, Tri);
            panel_visible_layout(app->obj_panel, 7);
            break;

        case ekCONVEX_POLY:
        case ekSIMPLE_POLY:
            layout_dbind_obj(app->pol_layout, &shape->body.pol, Pol);
            panel_visible_layout(app->obj_panel, 8);
            break;

        cassert_default();
        }
    }
```

```
    else
    {
        layout_dbind_obj(app->cld_layout, NULL, Cloud);
        layout_dbind_obj(app->seg_layout, NULL, Seg);
        layout_dbind_obj(app->cir_layout, NULL, Cir2Df);
        layout_dbind_obj(app->box_layout, NULL, Box);
        layout_dbind_obj(app->obb_layout, NULL, OBB);
        layout_dbind_obj(app->tri_layout, NULL, Tri);
        layout_dbind_obj(app->pol_layout, NULL, Pol);
        panel_visible_layout(app->obj_panel, 0);
    }

    col2dhello_update_gui(app);
}
```

Capítulo 29

Dibujando en una imagen

En este ejemplo vemos como generar gráficos vectoriales en dos contextos diferentes utilizando la misma rutina de dibujo (Figura 29.1). En la parte izquierda volcamos directamente en la ventana a través de un control `View`. En la parte derecha generamos una imagen utilizando diferentes resoluciones. Para visualizarla utilizamos un control `ImageView` configurado para estirar la imagen en caso que sea de menor tamaño que el propio control, lo que deja patente la pérdida de calidad. El **código fuente** está en la caperta `/src/howto/drawimg` de la distribución del SDK.

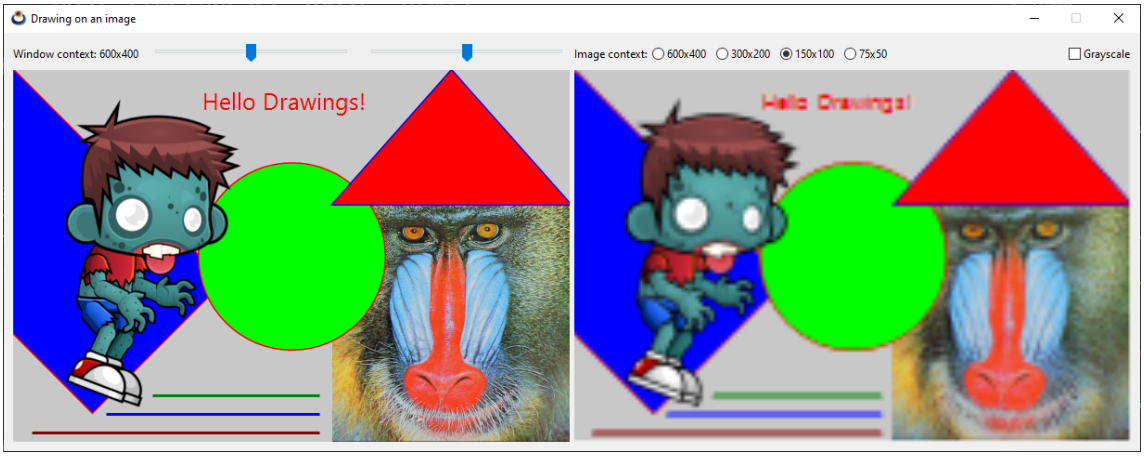


Figura 29.1: Contextos 2D: Ventana (izquierda), Imagen (derecha).

Listado 29.1: demo/drawimg/drawimg.c

```
/* Drawing on an image */

#include "res_drawimg.h"
#include <nappgui.h>
```

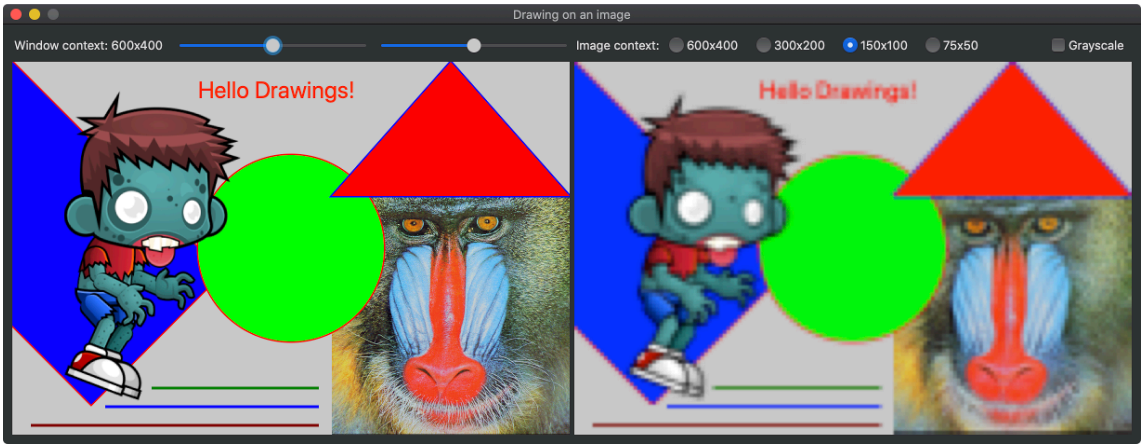


Figura 29.2: Versión macOS.

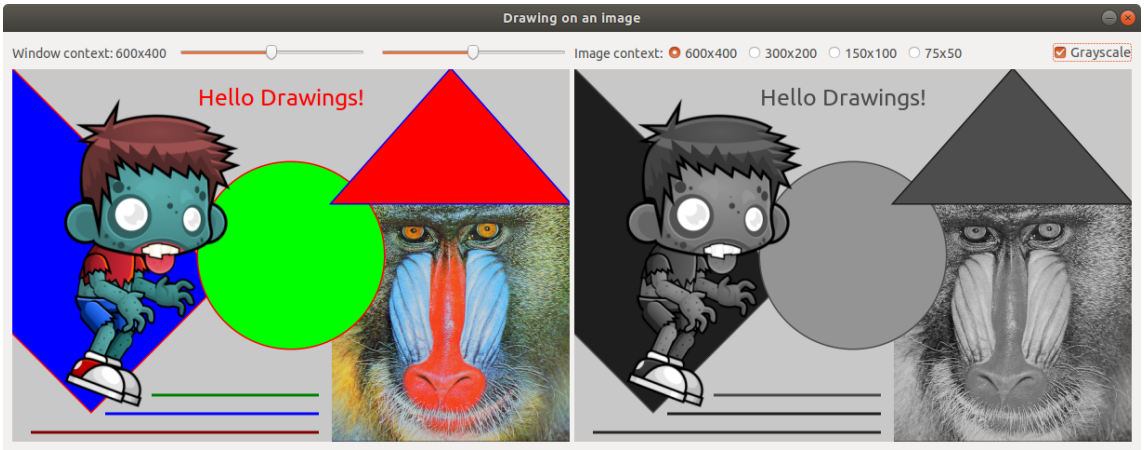


Figura 29.3: Versión Linux.

```
typedef struct _app_t App;

struct _app_t
{
    Window *window;
    Window *expwin;
    Font *font;
    View *view;
    ImageView *iview;
    uint32_t res;
    real32_t angle;
    real32_t scale;
    String *exp_path;
    codec_t exp_codec;
```

```
    uint32_t exp_bpp;
    bool_t exp_alpha;
};

static uint32_t i_WIDTH[4] = {600, 300, 150, 75};
static uint32_t i_HEIGHT[4] = {400, 200, 100, 50};
static real32_t i_SCALE[4] = {1, .5f, .25f, .125f};

/*---------------------------------------------------------------------------*/

static void i_draw(DCtx *ctx, const T2Df *t2d_global, const Font *font)
{
    T2Df t2d_object;
    V2Df triangle[] = { {472,0}, {600,144}, {344,144} };
    const Image *image1 = gui_image(MONKEY_GIF);
    const Image *image2 = gui_image(ZOMBIE_PNG);
    t2d_scalef(&t2d_object, t2d_global, .5f, .5f);
    draw_matrixf(ctx, &t2d_object);
    draw_image(ctx, image1, 688, 288);
    draw_line_color(ctx, color_rgb(255, 0, 0));
    draw_line_width(ctx, 3);
    draw_fill_color(ctx, color_rgb(0, 0, 255));
    t2d_rotatef(&t2d_object, t2d_global, kBMATH_PIf / 4);
    draw_matrixf(ctx, &t2d_object);
    draw_rect(ctx, ekSKFILL, 0, 0, 320, 200);
    draw_fill_color(ctx, color_rgb(0, 255, 0));
    draw_matrixf(ctx, t2d_global);
    draw_circle(ctx, ekSKFILL, 300, 200, 100);
    draw_line_color(ctx, color_rgb(0, 0, 255));
    draw_fill_color(ctx, color_rgb(255, 0, 0));
    draw_polygon(ctx, ekSKFILL, triangle, 3);
    t2d_scalef(&t2d_object, t2d_global, .7f, .7f);
    draw_matrixf(ctx, &t2d_object);
    draw_image(ctx, image2, 0, 0);
    draw_font(ctx, font);
    draw_matrixf(ctx, t2d_global);
    draw_text_color(ctx, color_rgb(255, 0, 0));
    draw_text(ctx, "Hello Drawings!", 200, 15);
    draw_line_color(ctx, color_rgb(0, 128, 0));
    draw_line(ctx, 150, 350, 330, 350);
    draw_line_color(ctx, color_rgb(0, 0, 255));
    draw_line(ctx, 100, 370, 330, 370);
    draw_line_color(ctx, color_rgb(128, 0, 0));
    draw_line(ctx, 20, 390, 330, 390);
}

/*---------------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    T2Df t2d;
```

```
    const EvDraw *p = event_params(e, EvDraw);
    t2d_rotatef(&t2d, kT2D_IDENTf, app->angle);
    t2d_scalef(&t2d, &t2d, app->scale, 1);
    draw_clear(p->ctx, color_rgb(200, 200, 200));
    i_draw(p->ctx, &t2d, app->font);
}

/*-----------------------------------------------------------------------------*/

static void i_draw_img(App *app)
{
    T2Df t2d;
    DCtx *ctx = dctx_bitmap(i_WIDTH[app->res], i_HEIGHT[app->res], ekRGB24);
    Image *image;
    t2d_scalef(&t2d, kT2D_IDENTf, i_SCALE[app->res], i_SCALE[app->res]);
    draw_clear(ctx, color_rgb(200, 200, 200));
    i_draw(ctx, &t2d, app->font);
    image = dctx_image(&ctx);
    imageview_image(app->iview, image);
    image_destroy(&image);
}

/*-----------------------------------------------------------------------------*/

static void i_OnResolution(App *app, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    app->res = p->index;
    i_draw_img(app);
}

/*-----------------------------------------------------------------------------*/

static Layout *i_filename_layout(void)
{
    Layout *layout = layout_create(2, 1);
    Edit *edit = edit_create();
    Button *button = button_push();
    button_text(button, "Open");
    layout_edit(layout, edit, 0, 0);
    layout_button(layout, button, 1, 0);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static Layout *i_bpp_layout(void)
{
    Layout *layout = layout_create(1, 5);
    Button *button1 = button_radio();
    Button *button2 = button_radio();
```

```
    Button *button3 = button_radio();
    Button *button4 = button_radio();
    Button *button5 = button_radio();
    button_text(button1, "1 bpp (2 colors)");
    button_text(button2, "2 bpp (4 colors)");
    button_text(button3, "4 bpp (16 colors)");
    button_text(button4, "8 bpp (32 colors)");
    button_text(button5, "RGB (True color)");
    layout_button(layout, button1, 0, 0);
    layout_button(layout, button2, 0, 1);
    layout_button(layout, button3, 0, 2);
    layout_button(layout, button4, 0, 3);
    layout_button(layout, button5, 0, 4);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static void i_OnOk(App *app, Event *e)
{
    window_stop_modal(app->expwin, 1);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static void i_OnCancel(App *app, Event *e)
{
    window_stop_modal(app->expwin, 0);
    unref(e);
}

/*-----------------------------------------------------------------------------*/

static Window *i_export_window(App *app)
{
    Window *window = window_create(ekWINDOW_TITLE | ekWINDOW_CLOSE);
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(3, 4);
    Layout *layout2 = i_filename_layout();
    Layout *layout3 = i_bpp_layout();
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    PopUp *popup = popup_create();
    Button *button1 = button_check();
    Button *button2 = button_push();
    Button *button3 = button_push();
    label_text(label1, "File name:");
    label_text(label2, "Format:");
```

```
    label_text(label3, "Pixel Depth (bpp):");
    label_text(label4, "Transparent background:");
    button_text(button2, "Ok");
    button_text(button3, "Cancel");
    button_OnClick(button2, listener(app, i_OnOk, App));
    button_OnClick(button3, listener(app, i_OnCancel, App));
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 0, 1);
    layout_label(layout1, label3, 0, 2);
    layout_label(layout1, label4, 0, 3);
    layout_layout(layout1, layout2, 1, 0);
    layout_popup(layout1, popup, 1, 1);
    layout_layout(layout1, layout3, 1, 2);
    layout_button(layout1, button1, 1, 3);
    layout_button(layout1, button2, 2, 0);
    layout_button(layout1, button3, 2, 1);
    panel_layout(panel, layout1);
    window_panel(window, panel);
    window_title(window, "Image export");
    return window;
}

/*-------------------------------------------------------------------------*/

static void i_export_png(void)
{
    const uint32_t w = 640, h = 400;
    uint32_t i, j, wi = w / 4;
    Palette *palette = palette_create(4);
    Pixbuf *pixbuf = pixbuf_create(w, h, ekINDEX2);
    color_t *c = palette_colors(palette);
    Image *image = NULL;
    c[0] = color_rgba(255, 0, 0, 255);
    c[1] = color_rgba(0, 255, 0, 170);
    c[2] = color_rgba(0, 0, 255, 85);
    c[3] = color_rgba(255, 255, 255, 1);
    for (i = 0; i < w; ++i)
    {
        uint32_t idx = 3;
        if (i < wi)
            idx = 0;
        else if (i < 2 * wi)
            idx = 1;
        else if (i < 3 * wi)
            idx = 2;

        for (j = 0; j < h; ++j)
            pixbuf_set(pixbuf, i, j, idx);
    }

    image = image_from_pixbuf(pixbuf, palette);
```

```
    image_codec(image, ekGIF);
    image_to_file(image, "/home/fran/Desktop/export.gif", NULL);
    pixbuf_destroy(&pixbuf);
    palette_destroy(&palette);
    image_destroy(&image);

    {
        Image *img = image_from_file("/home/fran/Desktop/country.jpg", NULL);
        image_codec(img, ekGIF);
        image_to_file(img, "/home/fran/Desktop/country.gif", NULL);
        image_destroy(&img);
    }
}

/*-----------------------------------------------------------------------------*/

static void i_OnExport(App *app, Event *e)
{
    V2Df p0, p1;
    S2Df s0, s1;
    uint32_t res = 0;
        unref(e);
    app->expwin = i_export_window(app);
    p0 = window_get_origin(app->window);
    s0 = window_get_size(app->window);
    s1 = window_get_size(app->expwin);
    p1 = v2df(p0.x + (s0.width - s1.width) / 2, p0.y + (s0.height - s1.height)
        ↪ / 2);
    window_origin(app->expwin, p1);
    res = window_modal(app->expwin, app->window);

    if (res == 1)
    {
        i_export_png();
    }

    window_destroy(&app->expwin);
}

/*-----------------------------------------------------------------------------*/

static Layout *i_img_layout(App *app)
{
    Layout *layout = layout_create(7, 1);
    Label *label = label_create();
    Button *button1 = button_radio();
    Button *button2 = button_radio();
    Button *button3 = button_radio();
    Button *button4 = button_radio();
    Button *button5 = button_push();
    label_text(label, "Image context:");
```

```
    button_text(button1, "600x400");
    button_text(button2, "300x200");
    button_text(button3, "150x100");
    button_text(button4, "75x50");
    button_text(button5, "Export...");
    button_state(button1, ekGUI_ON);
    button_OnClick(button1, listener(app, i_OnResolution, App));
    button_OnClick(button5, listener(app, i_OnExport, App));
    layout_label(layout, label, 0, 0);
    layout_button(layout, button1, 1, 0);
    layout_button(layout, button2, 2, 0);
    layout_button(layout, button3, 3, 0);
    layout_button(layout, button4, 4, 0);
    layout_button(layout, button5, 6, 0);
    layout_hmargin(layout, 0, 5);
    layout_hmargin(layout, 1, 10);
    layout_hmargin(layout, 2, 10);
    layout_hmargin(layout, 3, 10);
    layout_hexpand(layout, 5);
    return layout;
}

/*-----------------------------------------------------------------------------*/

static void i_OnAngle(App *app, Event *e)
{
    const EvSlider *p = event_params(e, EvSlider);
    app->angle = (p->pos - .5f) * kBMATH_PIf;
    view_update(app->view);
}

/*-----------------------------------------------------------------------------*/

static void i_OnScale(App *app, Event *e)
{
    const EvSlider *p = event_params(e, EvSlider);
    app->scale = p->pos + .5f;
    view_update(app->view);
}

/*-----------------------------------------------------------------------------*/

static Layout *i_win_layout(App *app)
{
    Layout *layout = layout_create(5, 1);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Slider *slider1 = slider_create();
    Slider *slider2 = slider_create();
    label_text(label1, "Window context: 600x400");
```

```
    label_text(label2, "Angle:");
    label_text(label3, "Scale:");
    slider_value(slider1, .5f);
    slider_value(slider2, .5f);
    slider_OnMoved(slider1, listener(app, i_OnAngle, App));
    slider_OnMoved(slider2, listener(app, i_OnScale, App));
    layout_label(layout, label1, 0, 0);
    layout_label(layout, label2, 1, 0);
    layout_label(layout, label3, 3, 0);
    layout_slider(layout, slider1, 2, 0);
    layout_slider(layout, slider2, 4, 0);
    layout_hmargin(layout, 0, 10);
    layout_hmargin(layout, 2, 10);
    layout_hexpand2(layout, 2, 4, .5f);
    return layout;
}

/*---------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(2, 2);
    Layout *layout2 = i_win_layout(app);
    Layout *layout3 = i_img_layout(app);
    View *view = view_create();
    ImageView *iview = imageview_create();
    view_size(view, s2df(600, 400));
    imageview_size(iview, s2df(600, 400));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    imageview_scale(iview, ekGUI_SCALE_ASPECT);
    layout_layout(layout1, layout2, 0, 0);
    layout_view(layout1, view, 0, 1);
    layout_imageview(layout1, iview, 1, 1);
    layout_layout(layout1, layout3, 1, 0);
    layout_margin(layout1, 10);
    layout_hmargin(layout1, 0, 5);
    layout_vmargin(layout1, 0, 5);
    panel_layout(panel, layout1);
    app->view = view;
    app->iview = iview;
    return panel;
}

/*---------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
```

```
}

/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    gui_respack(res_drawimg_respack);
    gui_language("");
    app->window = window_create(ekWINDOW_STD);
    app->font = font_system(25.f, 0);
    app->res = 0;
    app->angle = 0;
    app->scale = 1;
    i_draw_img(app);
    window_panel(app->window, panel);
    window_title(app->window, "Drawing on an image");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    font_destroy(&(*app)->font);
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 30

Dibujos con scroll

La siguiente aplicación muestra como gestionar un área de dibujo muy grande, de la que únicamente una pequeña porción es visible. Representaremos una rejilla de 2000x2000 celdas, utilizando un control `View` con barras de scroll. Los objetivos que perseguimos con este ejemplo son:

- Optimizar el evento `OnDraw` para dibujar únicamente el área visible, evitando lanzar comandos innecesarios.
- Dimensionar las barras de scroll con `view_content_size`.
- Desplazar el área visible mediante `view_scroll_x`, `view_scroll_y`.
- Obtener el área visible con `view_viewport`.
- Uso del ratón: Poder hacer click sobre una celda o resaltarla cuando el cursor está sobre ella.
- Uso del teclado: Permitir a la vista capturar el foco y mover la celda activa con las teclas `[Left]`, `[Right]`, `[Up]` y `[Down]`. La navegación con el teclado exige que dicha celda siempre sea visible.

Listado 30.1: demo/drawbig/drawbig.c

```
/* Drawing a big area with scrollbars */

#include <nappgui.h>

typedef struct _app_t App;

struct _app_t
{
    Window *window;
    View *view;
    Label *label;
    uint32_t col_id;
```

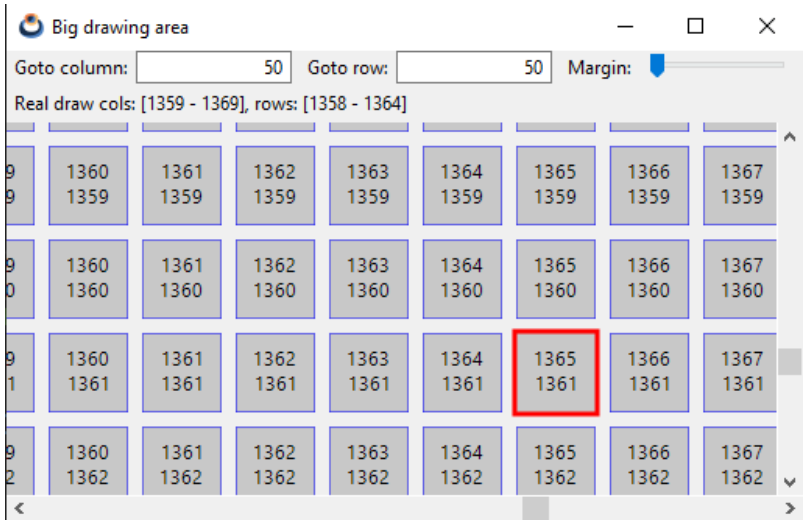


Figura 30.1: Versión Windows.

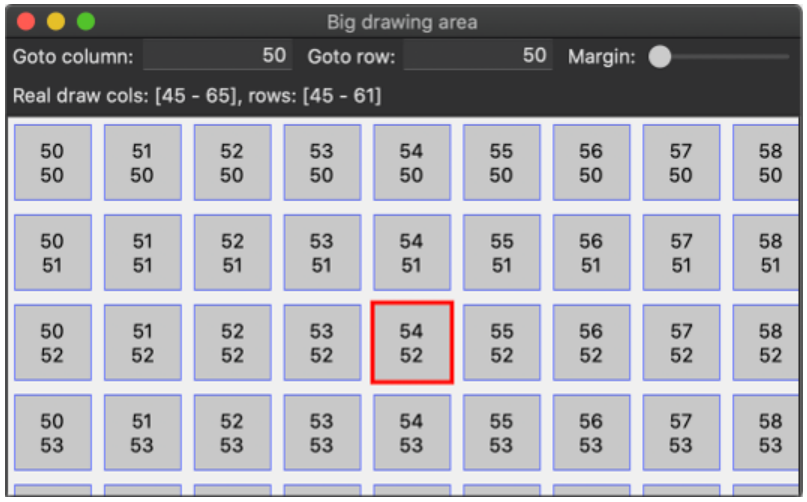


Figura 30.2: Versión macOS.

```
    uint32_t row_id;
    uint32_t margin;
    uint32_t mouse_cell_x;
    uint32_t mouse_cell_y;
    uint32_t sel_cell_x;
    uint32_t sel_cell_y;
    bool_t focus;
};

static const uint32_t i_NUM_COLS = 2000;
```

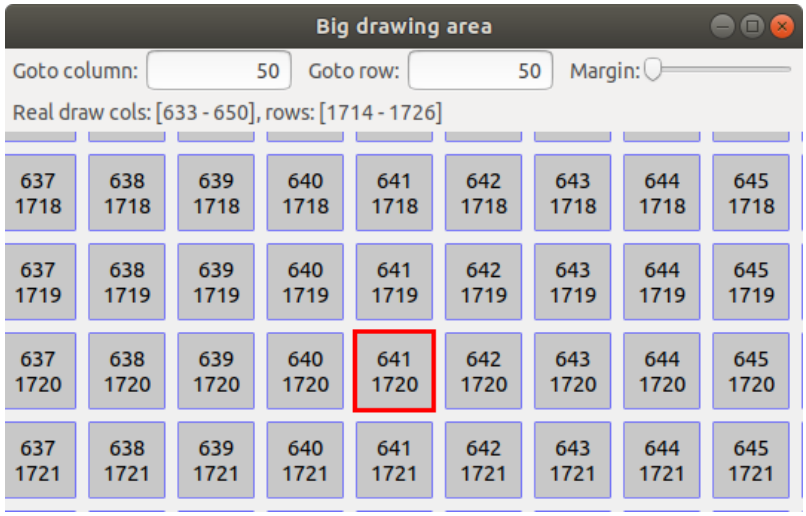


Figura 30.3: Versión Linux.

```
static const uint32_t i_NUM_ROWS = 2000;
static const real32_t i_CELL_SIZE = 50;

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_dbind(void)
{
    dbind(App, uint32_t, col_id);
    dbind(App, uint32_t, row_id);
    dbind(App, uint32_t, margin);
    dbind_range(App, uint32_t, col_id, 0, i_NUM_COLS - 1);
    dbind_range(App, uint32_t, row_id, 0, i_NUM_ROWS - 1);
    dbind_range(App, uint32_t, margin, 10, 50);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_content_size(App *app)
{
    real32_t width = i_NUM_COLS * i_CELL_SIZE + (i_NUM_COLS + 1) * app->
        ↪ margin;
    real32_t height = i_NUM_ROWS * i_CELL_SIZE + (i_NUM_ROWS + 1) * app->
        ↪ margin;
    view_content_size(app->view, s2df((real32_t)width, (real32_t)height),
        ↪ s2df(10, 10));
}
```

```
/*
    ↪ ---------------------------------------------------------------------------
    ↪ */

static void i_scroll_to_cell(App *app)
{
    real32_t xpos = app->col_id * i_CELL_SIZE + (app->col_id + 1) * app->
        ↪ margin;
    real32_t ypos = app->row_id * i_CELL_SIZE + (app->row_id + 1) * app->
        ↪ margin;
    xpos -= 5;
    ypos -= 5;
    view_scroll_x(app->view, xpos);
    view_scroll_y(app->view, ypos);
}

/*
    ↪ ---------------------------------------------------------------------------
    ↪ */

static void i_draw_clipped(App *app, DCtx *ctx, const real32_t x, const
    ↪ real32_t y, const real32_t width, const real32_t height)
{
    register uint32_t sti, edi;
    register uint32_t stj, edj;
    real32_t cellsize = i_CELL_SIZE + (real32_t)app->margin;
    real32_t hcell = i_CELL_SIZE / 2;
    register real32_t posx = 0;
    register real32_t posy = 0;
    register uint32_t i, j;

    /* Calculate the visible cols */
    sti = (uint32_t)bmath_floorf(x / cellsize);
    edi = sti + (uint32_t)bmath_ceilf(width / cellsize) + 1;
    if (edi > i_NUM_COLS)
        edi = i_NUM_COLS;

    /* Calculate the visible rows */
    stj = (uint32_t)bmath_floorf(y / cellsize);
    edj = stj + (uint32_t)bmath_ceilf(height / cellsize) + 1;
    if (edj > i_NUM_ROWS)
        edj = i_NUM_ROWS;

    posy = (real32_t)app->margin + stj * cellsize;

    {
        char_t text[256];
        bstd_sprintf(text, sizeof(text), "Real draw cols: [%d - %d], rows:
            ↪  [%d - %d]", sti, edi, stj, edj);
        label_text(app->label, text);
```

```
    }

    draw_fill_color(ctx, color_gray(240));
    draw_rect(ctx, ekFILL, x, y, width, height);
    draw_fill_color(ctx, color_gray(200));
    draw_line_color(ctx, kCOLOR_BLUE);
    draw_line_width(ctx, 1);
    draw_text_align(ctx, ekCENTER, ekCENTER);
    draw_text_halign(ctx, ekCENTER);

    for (j = stj; j < edj; ++j)
    {
        posx = (real32_t)app->margin + sti * cellsize;
        for (i = sti; i < edi; ++i)
        {
            char_t text[128];
            bool_t special_cell = FALSE;

            bstd_sprintf(text, sizeof(text), "%d\n%d", i, j);

            if (app->sel_cell_x == i && app->sel_cell_y == j)
            {
                draw_line_width(ctx, 6);
                if (app->focus == TRUE)
                    draw_line_color(ctx, kCOLOR_RED);
                else
                    draw_line_color(ctx, color_gray(100));

                special_cell = TRUE;
            }
            else if (app->mouse_cell_x == i && app->mouse_cell_y == j)
            {
                draw_line_width(ctx, 3);
                draw_line_color(ctx, kCOLOR_BLUE);
                special_cell = TRUE;
            }

            draw_rect(ctx, ekSKFILL, posx, posy, i_CELL_SIZE, i_CELL_SIZE)
                ↪ ;
            draw_text(ctx, text, posx + hcell, posy + hcell);

            if (special_cell == TRUE)
            {
                draw_line_width(ctx, 1);
                draw_line_color(ctx, kCOLOR_BLUE);
            }

            posx += cellsize;
        }

        posy += cellsize;
```

```
    }
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);
    i_draw_clipped(app, p->ctx, p->x, p->y, p->width, p->height);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_mouse_cell(App *app, const real32_t x, const real32_t y,
    ↪ const uint32_t action)
{
    real32_t cellsize = i_CELL_SIZE + (real32_t)app->margin;
    uint32_t mx = (uint32_t)bmath_floorf(x / cellsize);
    uint32_t my = (uint32_t)bmath_floorf(y / cellsize);
    real32_t xmin = mx * cellsize + (real32_t)app->margin;
    real32_t xmax = xmin + i_CELL_SIZE;
    real32_t ymin = my * cellsize + (real32_t)app->margin;
    real32_t ymax = ymin + i_CELL_SIZE;

    if (x >= xmin && x <= xmax && y >= ymin && y <= ymax)
    {
        if (action == 0)
        {
            app->mouse_cell_x = mx;
            app->mouse_cell_y = my;
        }
        else
        {
            app->sel_cell_x = mx;
            app->sel_cell_y = my;
        }
    }
    else
    {
        app->mouse_cell_x = UINT32_MAX;
        app->mouse_cell_y = UINT32_MAX;
    }

    view_update(app->view);
}

/*
```

```
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnMove(App *app, Event *e)
{
    const EvMouse *p = event_params(e, EvMouse);
    i_mouse_cell(app, p->x, p->y, 0);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnUp(App *app, Event *e)
{
    const EvMouse *p = event_params(e, EvMouse);
    i_mouse_cell(app, p->x, p->y, 0);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnDown(App *app, Event *e)
{
    const EvMouse *p = event_params(e, EvMouse);
    i_mouse_cell(app, p->x, p->y, 1);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnFocus(App *app, Event *e)
{
    const bool_t *p = event_params(e, bool_t);
    app->focus = *p;
    view_update(app->view);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnKeyDown(App *app, Event *e)
{
    const EvKey *p = event_params(e, EvKey);
    View *view = event_sender(e, View);
    real32_t margin = (real32_t)app->margin;
    real32_t cellsize = i_CELL_SIZE + margin;
    V2Df scroll;
```

```
    S2Df size;

    view_viewport(view, &scroll, &size);

    if (p->key == ekKEY_DOWN && app->sel_cell_y < i_NUM_ROWS - 1)
    {
        real32_t ymin = (app->sel_cell_y + 1) * cellsize + margin;
        ymin += i_CELL_SIZE;

        if (scroll.y + size.height <= ymin)
        {
            view_scroll_y(view, ymin - size.height + margin);
            app->mouse_cell_x = UINT32_MAX;
            app->mouse_cell_y = UINT32_MAX;
        }

        app->sel_cell_y += 1;
        view_update(app->view);
    }

    if (p->key == ekKEY_UP && app->sel_cell_y > 0)
    {
        real32_t ymin = (app->sel_cell_y - 1) * cellsize + (real32_t)app->
            ↪ margin;

        if (scroll.y >= ymin)
        {
            view_scroll_y(view, ymin - margin);
            app->mouse_cell_x = UINT32_MAX;
            app->mouse_cell_y = UINT32_MAX;
        }

        app->sel_cell_y -= 1;
        view_update(app->view);
    }

    if (p->key == ekKEY_RIGHT && app->sel_cell_x < i_NUM_COLS - 1)
    {
        real32_t xmin = (app->sel_cell_x + 1) * cellsize + margin;
        xmin += i_CELL_SIZE;

        if (scroll.x + size.width <= xmin)
        {
            view_scroll_x(view, xmin - size.width + margin);
            app->mouse_cell_x = UINT32_MAX;
            app->mouse_cell_y = UINT32_MAX;
        }

        app->sel_cell_x += 1;
        view_update(app->view);
    }
```

```
    if (p->key == ekKEY_LEFT && app->sel_cell_x > 0)
    {
        real32_t xmin = (app->sel_cell_x - 1) * cellsize + (real32_t)app->
            ↪ margin;

        if (scroll.x >= xmin)
        {
            view_scroll_x(view, xmin - margin);
            app->mouse_cell_x = UINT32_MAX;
            app->mouse_cell_y = UINT32_MAX;
        }

        app->sel_cell_x -= 1;
        view_update(app->view);
    }
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnDataChange(App *app, Event *e)
{
    unref(e);
    i_scroll_to_cell(app);
    view_update(app->view);
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(6, 1);
    Layout *layout2 = layout_create(1, 3);
    Label *label1 = label_create();
    Label *label2 = label_create();
    Label *label3 = label_create();
    Label *label4 = label_create();
    Edit *edit1 = edit_create();
    Edit *edit2 = edit_create();
    Slider *slider = slider_create();
    View *view = view_scroll();
    label_text(label1, "Goto column:");
    label_text(label2, "Goto row:");
    label_text(label3, "Margin:");
    edit_align(edit1, ekRIGHT);
    edit_align(edit2, ekRIGHT);
```

```
    view_size(view, s2df(256, 256));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    view_OnMove(view, listener(app, i_OnMove, App));
    view_OnUp(view, listener(app, i_OnUp, App));
    view_OnDown(view, listener(app, i_OnDown, App));
    view_OnFocus(view, listener(app, i_OnFocus, App));
    view_OnKeyDown(view, listener(app, i_OnKeyDown, App));
    layout_label(layout1, label1, 0, 0);
    layout_label(layout1, label2, 2, 0);
    layout_label(layout1, label3, 4, 0);
    layout_edit(layout1, edit1, 1, 0);
    layout_edit(layout1, edit2, 3, 0);
    layout_slider(layout1, slider, 5, 0);
    layout_layout(layout2, layout1, 0, 0);
    layout_label(layout2, label4, 0, 1);
    layout_view(layout2, view, 0, 2);
    layout_tabstop(layout2, 0, 2, TRUE);
    layout_margin2(layout1, 0, 5);
    layout_hmargin(layout1, 0, 5);
    layout_hmargin(layout1, 1, 10);
    layout_hmargin(layout1, 2, 5);
    layout_hmargin(layout1, 3, 10);
    layout_hmargin(layout1, 4, 5);
    layout_vmargin(layout2, 0, 5);
    layout_vmargin(layout2, 1, 5);
    layout_halign(layout2, 0, 0, ekLEFT);
    layout_halign(layout2, 0, 1, ekJUSTIFY);
    layout_vexpand(layout2, 2);
    cell_padding2(layout_cell(layout2, 0, 1), 0, 5);
    cell_dbind(layout_cell(layout1, 1, 0), App, uint32_t, col_id);
    cell_dbind(layout_cell(layout1, 3, 0), App, uint32_t, row_id);
    cell_dbind(layout_cell(layout1, 5, 0), App, uint32_t, margin);
    layout_dbind(layout2, listener(app, i_OnDataChange, App), App);
    layout_dbind_obj(layout2, app, App);
    panel_layout(panel, layout2);
    app->view = view;
    app->label = label4;
    return panel;
}

/*
    ↪ ------------------------------------------------------------------------
    ↪ */

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}
```

```
/*
    ↪ ---------------------------------------------------------------------------
    ↪ */

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = NULL;
    i_dbind();
    app->col_id = 50;
    app->row_id = 50;
    app->margin = 10;
    app->mouse_cell_x = UINT32_MAX;
    app->mouse_cell_y = UINT32_MAX;
    app->sel_cell_x = app->col_id;
    app->sel_cell_y = app->row_id;
    app->focus = FALSE;
    panel = i_panel(app);
    app->window = window_create(ekWINDOW_STDRES);
    i_content_size(app);
    window_panel(app->window, panel);
    window_title(app->window, "Big drawing area");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    i_scroll_to_cell(app);
    return app;
}

/*
    ↪ ---------------------------------------------------------------------------
    ↪ */

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    heap_delete(app, App);
}

/*
    ↪ ---------------------------------------------------------------------------
    ↪ */

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 31

Imágenes desde URLs

En esta demo construimos un sencillo visor de imágenes Web. El programa permite descargarlas y visualizarlas mediante una lista. El **código fuente** está en la caperta `/src /howto/urlimg` de la distribución del SDK.

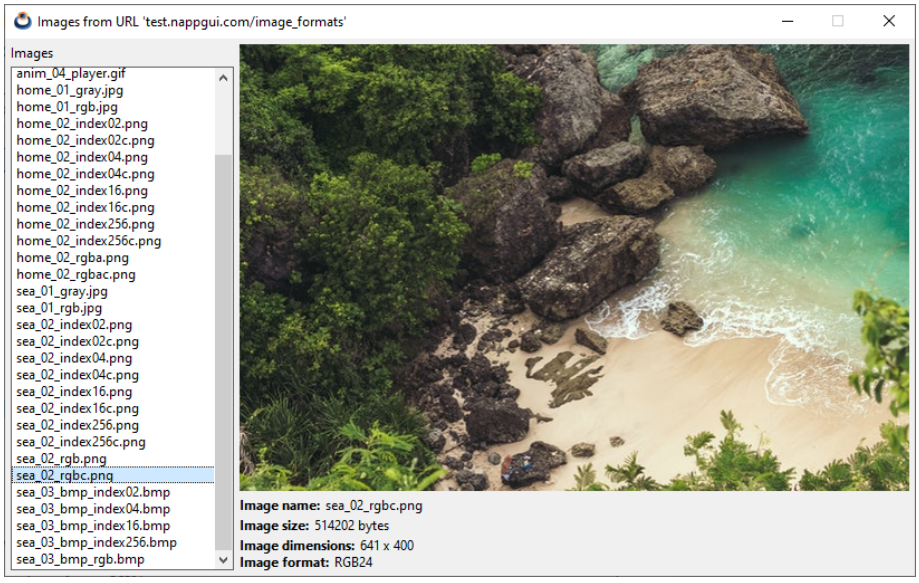


Figura 31.1: Versión Windows

Listado 31.1: demo/urlimg/urlimg.c

```
/* Images from URL */

#include <inet/inet.h>
#include <inet/httpreq.h>
#include <nappgui.h>
```

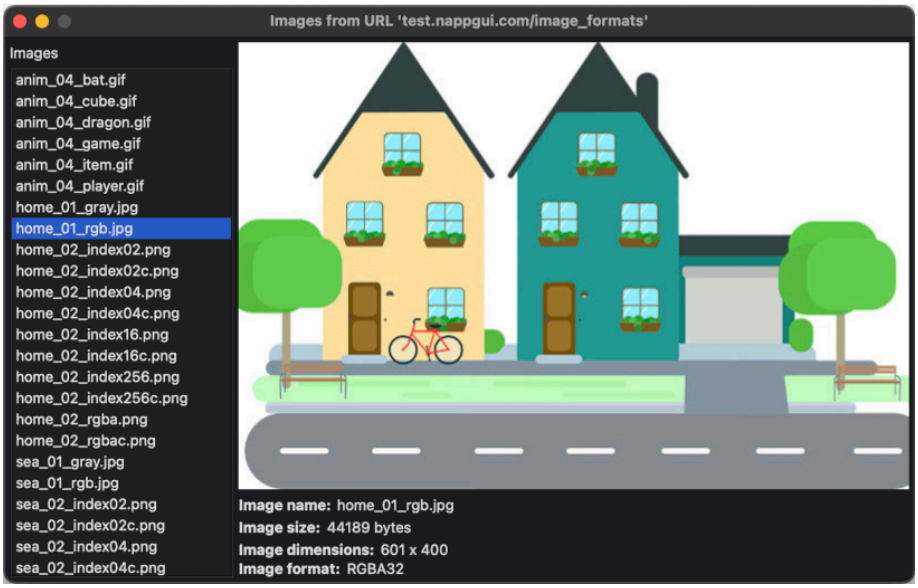


Figura 31.2: Versión macOS

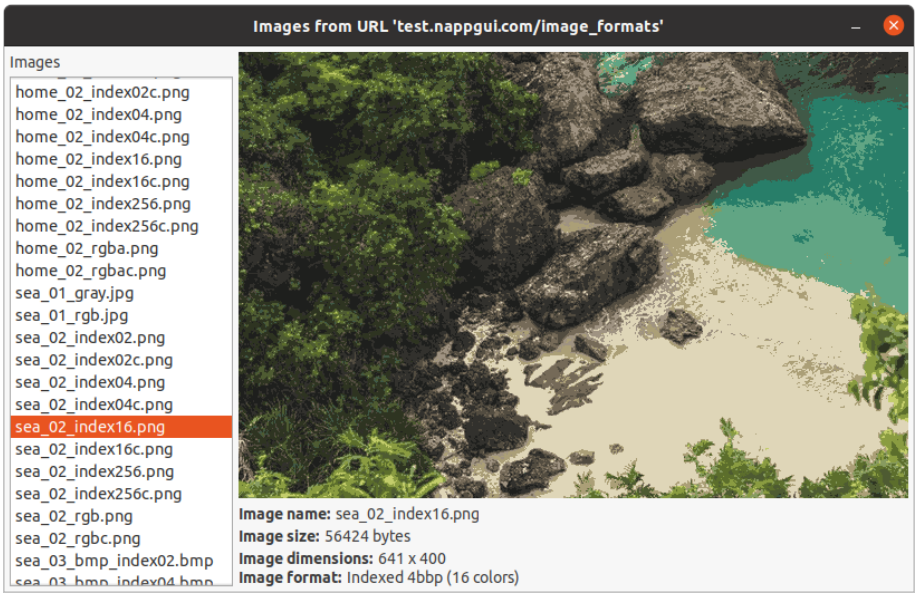


Figura 31.3: Versión Linux

```
typedef struct _app_t App;

struct _app_t
{
    Window *window;
```

```
    ImageView *view;
    uint32_t selected;
    Label *imgname;
    Label *imgsize;
    Label *imgres;
    Label *imgformat;
};

static const char_t *i_FILES[] = {
                              "anim_04_bat.gif",
                              "anim_04_cube.gif",
                              "anim_04_dragon.gif",
                              "anim_04_game.gif",
                              "anim_04_item.gif",
                              "anim_04_player.gif",
                              "static_05_cube.gif",
                              "home_01_gray.jpg",
                              "home_01_rgb.jpg",
                              "home_02_index02.png",
                              "home_02_index02c.png",
                              "home_02_index04.png",
                              "home_02_index04c.png",
                              "home_02_index16.png",
                              "home_02_index16c.png",
                              "home_02_index256.png",
                              "home_02_index256c.png",
                              "home_02_rgba.png",
                              "home_02_rgbac.png",
                              "sea_01_gray.jpg",
                              "sea_01_rgb.jpg",
                              "sea_02_index02.png",
                              "sea_02_index02c.png",
                              "sea_02_index04.png",
                              "sea_02_index04c.png",
                              "sea_02_index16.png",
                              "sea_02_index16c.png",
                              "sea_02_index256.png",
                              "sea_02_index256c.png",
                              "sea_02_rgb.png",
                              "sea_02_rgbc.png",
                              "sea_03_bmp_index02.bmp",
                              "sea_03_bmp_index04.bmp",
                              "sea_03_bmp_index16.bmp",
                              "sea_03_bmp_index256.bmp",
                              "sea_03_bmp_rgb.bmp" };

/*---------------------------------------------------------------------------*/

static __INLINE String *i_pixformat(const pixformat_t format, const uint32_t
    ↪ ncolors)
{
```

```
    switch (format) {
    case ekINDEX1:
        return str_printf("Indexed 1bbp (%d colors)", ncolors);
    case ekINDEX2:
        return str_printf("Indexed 2bbp (%d colors)", ncolors);
    case ekINDEX4:
        return str_printf("Indexed 4bbp (%d colors)", ncolors);
    case ekINDEX8:
        return str_printf("Indexed 8bbp (%d colors)", ncolors);
    case ekGRAY8:
        return str_c("Gray8");
    case ekRGB24:
        return str_c("RGB24");
    case ekRGBA32:
        return str_c("RGBA32");
    case ekFIMAGE:
                break;
    }
    return str_c("Unknown");
}

/*---------------------------------------------------------------------------*/

static void i_download(App *app)
{
    String *url = str_printf("http://test.nappgui.com/image_formats/%s",
        ↪ i_FILES[app->selected]);
    Stream *stm = http_dget(tc(url), NULL, NULL);
    if (stm != NULL)
    {
        uint32_t ncolors = 0;
        uint64_t start = stm_bytes_readed(stm);
        Image *image = image_read(stm);
        uint64_t end = stm_bytes_readed(stm);
        uint32_t width = image_width(image);
        uint32_t height = image_width(image);
        pixformat_t format = image_format(image);
        String *ssize = str_printf("%d bytes", (uint32_t)(end - start));
        String *sres = NULL;
        String *sformat = NULL;

        /* Full check of read/write pixels
        We create again the same image, based on pixel info */
        if (image_get_codec(image) != ekGIF)
        {
            Pixbuf *pixels = image_pixels(image, ekFIMAGE);
            Image *nimage = image_from_pixbuf(pixels, NULL);
            cassert(format == pixbuf_format(pixels));
            pixbuf_destroy(&pixels);
            image_destroy(&image);
            image = nimage;
```

```
        }

        imageview_image(app->view, image);
        sres = str_printf("%d x %d", width, height);
        sformat = i_pixformat(format, ncolors);
        label_text(app->imgname, i_FILES[app->selected]);
        label_text(app->imgsize, tc(ssize));
        label_text(app->imgres, tc(sres));
        label_text(app->imgformat, tc(sformat));
        stm_close(&stm);
        image_destroy(&image);
        str_destroy(&ssize);
        str_destroy(&sres);
        str_destroy(&sformat);
    }

    str_destroy(&url);
}

/*---------------------------------------------------------------------------*/

static Layout* i_label(const char_t *title, Label **info)
{
    Layout *layout = layout_create(2, 1);
    Label *label = label_create();
    Font *font = font_system(font_regular_size(), ekFBOLD);
    *info = label_create();
    label_text(label, title);
    label_font(label, font);
    layout_label(layout, label, 0, 0);
    layout_label(layout, *info, 1, 0);
    layout_halign(layout, 1, 0, ekJUSTIFY);
    layout_hmargin(layout, 0, 5);
    layout_hexpand(layout, 1);
    font_destroy(&font);
    return layout;
}

/*---------------------------------------------------------------------------*/

static void i_add_files(ListBox *listbox)
{
    register uint32_t i, n = sizeof(i_FILES) / sizeof(char_t*);
    for (i = 0; i < n; ++i)
        listbox_add_elem(listbox, i_FILES[i], NULL);
    listbox_select(listbox, 0, TRUE);
}

/*---------------------------------------------------------------------------*/

static void i_OnSelect(App *app, Event *e)
```

```
{
    const EvButton *p = event_params(e, EvButton);
    app->selected = p->index;
    i_download(app);
}

/*-------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout1 = layout_create(2, 1);
    Layout *layout2 = layout_create(1, 2);
    Layout *layout3 = layout_create(1, 5);
    Label *label = label_create();
    ListBox *listbox = listbox_create();
    ImageView *view = imageview_create();
    app->view = view;
    label_text(label, "Images");
    i_add_files(listbox);
    listbox_OnSelect(listbox, listener(app, i_OnSelect, App));
    imageview_size(view, s2df(600, 400));
    layout_label(layout2, label, 0, 0);
    layout_listbox(layout2, listbox, 0, 1);
    layout_imageview(layout3, view, 0, 0);
    layout_layout(layout3, i_label("Image name:", &app->imgname), 0, 1);
    layout_layout(layout3, i_label("Image size:", &app->imgsize), 0, 2);
    layout_layout(layout3, i_label("Image dimensions:", &app->imgres), 0, 3);
    layout_layout(layout3, i_label("Pixel format:", &app->imgformat), 0, 4);
    layout_layout(layout1, layout2, 0, 0);
    layout_layout(layout1, layout3, 1, 0);
    layout_margin(layout1, 5);
    layout_hmargin(layout1, 0, 5);
    layout_vmargin(layout2, 0, 5);
    layout_vmargin(layout3, 0, 5);
    layout_vmargin(layout3, 1, 3);
    layout_vmargin(layout3, 2, 3);
    layout_hsize(layout1, 0, 200);
    layout_vexpand(layout2, 1);
    panel_layout(panel, layout1);
    return panel;
}

/*-------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
{
    osapp_finish();
    unref(app);
    unref(e);
}
```

```
/*---------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->window = window_create(ekWINDOW_STD);
    app->selected = 0;
    inet_start();
    i_download(app);
    window_panel(app->window, panel);
    window_title(app->window, "Images from URL 'http://test.nappgui.com/
        ↪ image_formats'");
    window_origin(app->window, v2df(500, 200));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*---------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    window_destroy(&(*app)->window);
    inet_finish();
    heap_delete(app, App);
}

/*---------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 32

Tabla de colores

La elección de colores RGB arbitrarios para utilizalos en interfaces gráficas no siempre será coherente con el tema de escritorio de la plataforma de destino. En *"Colores"* (Página 283) se definen una serie de colores "de sistema" y la posibilidad de crear versiones alternativas para temas claros u oscuros. En esta demo muestra este repertorio en función de la plataforma donde corre el programa. El **código fuente** está en la caperta `/src/howto/colorview` de la distribución del SDK.

Figura 32.1: Tabla de colores.

Listado 32.1: demo/colorview/colorview.c

```
/* Color View */

#include <nappgui.h>

typedef struct _viewitem_t ViewItem;
typedef struct _app_t App;
```

```
struct _viewitem_t
{
    const char_t *name;
    color_t color;
};

struct _app_t
{
    Window *window;
    View *view;
    ArrSt(ViewItem) *items;
    uint32_t num_cols;
    Font *font;
};

DeclSt(ViewItem);
static const real32_t i_ITEM_WIDTH = 64;
static const real32_t i_VER_MARGIN = 10;
static const real32_t i_HOR_MARGIN = 15;

/*-----------------------------------------------------------------------------*/

static void i_add(ArrSt(ViewItem) *items, const char_t *name, const color_t
    ↪ color)
{
    ViewItem *item = arrst_new(items, ViewItem);
    item->name = name;
    item->color = color;
}

/*-----------------------------------------------------------------------------*/

static ArrSt(ViewItem)* i_colors(void)
{
    ArrSt(ViewItem) *items = arrst_create(ViewItem);
    i_add(items, "Label", gui_label_color());
    i_add(items, "View", gui_view_color());
    i_add(items, "Line", gui_line_color());
    i_add(items, "Border", gui_border_color());
    i_add(items, "Link", gui_link_color());
    i_add(items, "Alt1", gui_alt_color(color_rgb(192, 255, 255), color_rgb(48,
        ↪ 112, 112)));
    i_add(items, "Alt2", gui_alt_color(color_rgb(255, 192, 255), color_rgb(128,
        ↪  48, 112)));
    i_add(items, "Alt3", gui_alt_color(color_rgb(255, 255, 192), color_rgb(112,
        ↪  112, 48)));
    i_add(items, "Black", kCOLOR_BLACK);
    i_add(items, "White", kCOLOR_WHITE);
    i_add(items, "Red", kCOLOR_RED);
    i_add(items, "Green", kCOLOR_GREEN);
    i_add(items, "Blue", kCOLOR_BLUE);
```

```
    i_add(items, "Yellow", kCOLOR_YELLOW);
    i_add(items, "Cyan", kCOLOR_CYAN);
    i_add(items, "Magenta", kCOLOR_MAGENTA);
    i_add(items, "Silver", color_rgb(192, 192, 192));
    i_add(items, "Gray", color_rgb(128, 128, 128));
    i_add(items, "Maroon", color_rgb(128, 0, 0));
    i_add(items, "Olive", color_rgb(128, 128, 0));
    i_add(items, "DGreen", color_rgb(0, 128, 0));
    i_add(items, "Teal", color_rgb(0, 128, 128));
    i_add(items, "Navy", color_rgb(0, 0, 128));
    i_add(items, "Purple", color_rgb(128, 0, 128));
    return items;
}

/*-----------------------------------------------------------------------*/

static void i_draw(DCtx *ctx, real32_t x, real32_t y, real32_t width, real32_t
    ↪ height, const ViewItem *item)
{
    real32_t cx1 = x + width / 2;
    real32_t cx2 = x + (width - i_ITEM_WIDTH) / 2;
    real32_t cy = y + height - i_ITEM_WIDTH;
    draw_fill_color(ctx, item->color);
    draw_rect(ctx, ekFILL, cx2, cy, i_ITEM_WIDTH, i_ITEM_WIDTH);
    draw_text_color(ctx, gui_label_color());
    draw_text(ctx, item->name, cx1, cy);
}

/*-----------------------------------------------------------------------*/

static void i_OnDraw(App *app, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);
    real32_t cwidth = (p->width - 2 * i_HOR_MARGIN) / app->num_cols;
    real32_t cheight = i_ITEM_WIDTH + font_height(app->font);

    draw_font(p->ctx, app->font);
    draw_text_align(p->ctx, ekCENTER, ekBOTTOM);

    arrst_foreach(item, app->items, ViewItem)
        uint32_t row = item_i / app->num_cols;
        uint32_t col = item_i  % app->num_cols;
        real32_t x = i_HOR_MARGIN + col * cwidth;
        real32_t y = row * cheight + (row + 1) * i_VER_MARGIN;
        i_draw(p->ctx, x, y, cwidth, cheight, item);
    arrst_end();
}

/*-----------------------------------------------------------------------*/

static void i_OnSize(App *app, Event *e)
```

```
{
    const EvSize *p = event_params(e, EvSize);
    View *view = event_sender(e, View);
    real32_t minwidth = i_ITEM_WIDTH + 2 * i_HOR_MARGIN;
    real32_t cwidth = 0, cheight = 0;

    cwidth = p->width;

    if (cwidth < minwidth)
    {
        cwidth = minwidth;
        app->num_cols = 1;
    }
    else
    {
        uint32_t n, num_rows;
        app->num_cols = (uint32_t)((cwidth - i_HOR_MARGIN) / (i_ITEM_WIDTH +
            ↪ i_HOR_MARGIN));
        n = arrst_size(app->items, ViewItem);
        num_rows = (n / app->num_cols);
        if ((n % app->num_cols) > 0)
            num_rows += 1;

        cheight = num_rows * (i_ITEM_WIDTH + font_height(app->font) +
            ↪ i_VER_MARGIN) + i_VER_MARGIN;
        if (cheight < p->height)
            cheight = p->height;
    }

    view_content_size(view, s2df(cwidth, cheight), s2df(1, 1));
    view_update(view);
}

/*---------------------------------------------------------------------------*/

static Panel *i_panel(App *app)
{
    Panel *panel = panel_create();
    Layout *layout = layout_create(1, 1);
    View *view = view_scroll();
    view_size(view, s2df(300, 200));
    view_OnDraw(view, listener(app, i_OnDraw, App));
    view_OnSize(view, listener(app, i_OnSize, App));
    layout_view(layout, view, 0, 0);
    panel_layout(panel, layout);
    return panel;
}

/*---------------------------------------------------------------------------*/

static void i_OnClose(App *app, Event *e)
```

```
{
    osapp_finish();
    unref(app);
    unref(e);
}

/*-------------------------------------------------------------------------*/

static App *i_create(void)
{
    App *app = heap_new0(App);
    Panel *panel = i_panel(app);
    app->items = i_colors();
    app->font = font_system(font_regular_size(), 0);
    app->window = window_create(ekWINDOW_STDRES);
    window_panel(app->window, panel);
    window_title(app->window, "Color View");
    window_origin(app->window, v2df(500, 200));
    window_size(app->window, s2df(500, 300));
    window_OnClose(app->window, listener(app, i_OnClose, App));
    window_show(app->window);
    return app;
}

/*-------------------------------------------------------------------------*/

static void i_destroy(App **app)
{
    arrst_destroy(&(*app)->items, NULL, ViewItem);
    window_destroy(&(*app)->window);
    font_destroy((&(*app)->font));
    heap_delete(app, App);
}

/*-------------------------------------------------------------------------*/

#include "osmain.h"
osmain(i_create, i_destroy, "", App)
```

Capítulo 33

Lectura/Escritura de Json

Listado 33.1: demo/htjson/htjson.c

```
/* JSON parsing examples */

#include "res_htjson.h"
#include <draw2d/draw2dall.h>
#include <inet/json.h>

/*---------------------------------------------------------------------------*/

/* C structs that map a Json object */
typedef struct _product_t Product;
typedef struct _products_t Products;

struct _product_t
{
    String *description;
    real32_t price;
};

struct _products_t
{
    uint32_t size;
    ArrSt(Product) *data;
};

DeclSt(Product);

/*---------------------------------------------------------------------------*/

static Stream* i_stm_from_json(const char_t* json_data)
{
    return stm_from_block((const byte_t*)json_data, str_len_c(json_data));
}
```

```
/*--------------------------------------------------------------------------*/

int main(int argc, char *argv[])
{
    unref(argc);
    unref(argv);
    draw2d_start();

    /* Parsing a Json boolean */
    {
        Stream *stm = i_stm_from_json("true");
        bool_t *json = json_read(stm, NULL, bool_t);
        bstd_printf("bool_t from Json: %d\n", *json);
        json_destroy(&json, bool_t);
        stm_close(&stm);
    }

    /* Parsing a Json unsigned int */
    {
        Stream *stm = i_stm_from_json("6654");
        uint16_t *json = json_read(stm, NULL, uint16_t);
        bstd_printf("uint16_t from Json: %d\n", *json);
        json_destroy(&json, uint16_t);
        stm_close(&stm);
    }

    /* Parsing a Json signed int */
    {
        Stream *stm = i_stm_from_json("-567");
        int16_t *json = json_read(stm, NULL, int16_t);
        bstd_printf("int16_t from Json: %d\n", *json);
        json_destroy(&json, int16_t);
        stm_close(&stm);
    }

    /* Parsing a Json real */
    {
        Stream *stm = i_stm_from_json("456.45");
        real32_t *json = json_read(stm, NULL, real32_t);
        bstd_printf("real32_t from Json: %.3f\n", *json);
        json_destroy(&json, real32_t);
        stm_close(&stm);
    }

    /* Parsing a Json string */
    {
        Stream *stm = i_stm_from_json("\"Hello World\"");
        String *json = json_read(stm, NULL, String);
        bstd_printf("String from Json: %s\n", tc(json));
        json_destroy(&json, String);
        stm_close(&stm);
```

```
}

/* Parsing a Json b64 encoded image */
{
    uint32_t size;
    ResPack *pack = res_htjson_respack("");
    const byte_t *data = respack_file(pack, JSON_B64_IMAGE_TXT, &size);
    Stream *stm = stm_from_block(data, size);
    Image *json = json_read(stm, NULL, Image);
    uint32_t width = image_width(json);
    uint32_t height = image_height(json);
    bstd_printf("Image from Json: width: %d height: %d\n", width, height);
    json_destroy(&json, Image);
    stm_close(&stm);
    respack_destroy(&pack);
}

/* Parsing a Json int array */
{
    Stream *stm = i_stm_from_json("[ -321, 12, -8943, 228, -220, 347 ]");
    ArrSt(int16_t) *json = json_read(stm, NULL, ArrSt(int16_t));
    bstd_printf("ArrSt(int16_t) from Json: ");
    arrst_foreach(id, json, int16_t)
        bstd_printf("%d ", *id);
    arrst_end()
    bstd_printf("\n");
    json_destroy(&json, ArrSt(int16_t));
    stm_close(&stm);
}

/* Parsing a Json String array */
{
    Stream *stm = i_stm_from_json("[ \"Red\", \"Green\", \"Blue\", \"Yellow
        ↪ \", \"Orange\" ]");
    ArrPt(String) *json = json_read(stm, NULL, ArrPt(String));
    bstd_printf("ArrPt(String) from Json: ");
    arrpt_foreach(str, json, String)
        bstd_printf("%s ", tc(str));
    arrpt_end()
    bstd_printf("\n");
    json_destroy(&json, ArrPt(String));
    stm_close(&stm);
}

/* Data binding (only once time in application) */
/* This allows the Json parser to know the structure of the objects */
dbind(Product, String*, description);
dbind(Product, real32_t, price);
dbind(Products, uint32_t, size);
dbind(Products, ArrSt(Product)*, data);
```

```
/* Parsing a Json object */
{
    static const char_t *JSON_OBJECT = "\
    {\
        \"size\" : 3,\
        \"data\" : [\
            {\
                \"description\" : \"Intel i7-7700K\",\
                \"price\" : 329.99\
            },\
            {\
                \"description\" : \"Ryzen-5-1600\",\
                \"price\" : 194.99\
            },\
            {\
                \"description\" : \"GTX-1060\",\
                \"price\" : 449.99\
            }\
        ]\
    }";

    Stream *stm = i_stm_from_json(JSON_OBJECT);
    Products *json = json_read(stm, NULL, Products);
    bstd_printf("Products object from Json: size %d\n", json->size);
    arrst_foreach(elem, json->data, Product)
        bstd_printf("    Product: %s Price %.2f\n", tc(elem->description),
            ↪ elem->price);
    arrst_end()
    bstd_printf("\n");
    json_destroy(&json, Products);
    stm_close(&stm);
}

/* Writting data/objects to JSon */
{
    Stream *stm = stm_memory(1024);

    /* Write boolean as Json */
    {
        bool_t data_bool = TRUE;
        stm_writef(stm, "Json from bool_t: ");
        json_write(stm, &data_bool, NULL, bool_t);
        stm_writef(stm, "\n");
    }

    /* Write unsigned integer as Json */
    {
        uint16_t data_uint = 6654;
        stm_writef(stm, "Json from uint16_t: ");
        json_write(stm, &data_uint, NULL, uint16_t);
        stm_writef(stm, "\n");
```

```
    }

    /* Write integer as Json */
    {
        int16_t data_int = -567;
        stm_writef(stm, "Json from int16_t: ");
        json_write(stm, &data_int, NULL, int16_t);
        stm_writef(stm, "\n");
    }

    /* Write real32_t as Json */
    {
        real32_t data_real = 456.45f;
        stm_writef(stm, "Json from real32_t: ");
        json_write(stm, &data_real, NULL, real32_t);
        stm_writef(stm, "\n");
    }

    /* Write String as Json */
    {
        String *data_str = str_c("Hello World");
        stm_writef(stm, "Json from String: ");
        json_write(stm, data_str, NULL, String);
        stm_writef(stm, "\n");
        str_destroy(&data_str);
    }

    /* Write Image as Json (string b64) */
    {
        Pixbuf *pixbuf = pixbuf_create(2, 2, ekGRAY8);
        Image *data_image = NULL;
        bmem_set1(pixbuf_data(pixbuf), 2 * 2, 128);
        data_image = image_from_pixbuf(pixbuf, NULL);
        stm_writef(stm, "Json from Image: ");
        json_write(stm, data_image, NULL, Image);
        stm_writef(stm, "\n");
        pixbuf_destroy(&pixbuf);
        image_destroy(&data_image);
    }

    /* Write int array as Json */
    {
        ArrSt(int16_t) *array = arrst_create(int16_t);
        arrst_append(array, -321, int16_t);
        arrst_append(array, 12, int16_t);
        arrst_append(array, -8943, int16_t);
        arrst_append(array, 228, int16_t);
        arrst_append(array, -220, int16_t);
        arrst_append(array, 347, int16_t);
        stm_writef(stm, "Json from int array: ");
        json_write(stm, array, NULL, ArrSt(int16_t));
```

```
        stm_writef(stm, "\n");
        arrst_destroy(&array, NULL, int16_t);
    }

    /* Write string array as Json */
    {
        ArrPt(String) *array = arrpt_create(String);
        arrpt_append(array, str_c("Red"), String);
        arrpt_append(array, str_c("Green"), String);
        arrpt_append(array, str_c("Blue"), String);
        arrpt_append(array, str_c("Yellow"), String);
        arrpt_append(array, str_c("Orange"), String);
        stm_writef(stm, "Json from string array: ");
        json_write(stm, array, NULL, ArrPt(String));
        stm_writef(stm, "\n");
        arrpt_destroy(&array, str_destroy, String);
    }

    /* Write object as Json */
    {
        Products *products = heap_new(Products);
        products->size = 3;
        products->data = arrst_create(Product);

        {
            Product *product = arrst_new(products->data, Product);
            product->description = str_c("Intel i7-7700K");
            product->price = 329.99f;
        }

        {
            Product *product = arrst_new(products->data, Product);
            product->description = str_c("Ryzen-5-1600");
            product->price = 194.99f;
        }

        {
            Product *product = arrst_new(products->data, Product);
            product->description = str_c("GTX-1060");
            product->price = 449.99f;
        }

        stm_writef(stm, "Json from object: ");
        json_write(stm, products, NULL, Products);
        stm_writef(stm, "\n");
        dbind_destroy(&products, Products);
    }

    {
        String *str = stm_str(stm);
        bstd_printf("%s\n", tc(str));
```

```
            str_destroy(&str);
        }

        stm_close(&stm);
    }

    draw2d_finish();
    return 0;
}
```

Salida del programa.

```
bool_t from Json: 1
uint16_t from Json: 6654
int16_t from Json: -567
real32_t from Json: 456.450
String from Json: Hello World
Image from Json: width: 269 height: 400
ArrSt(int16_t) from Json: -321 12 -8943 228 -220 347
ArrPt(String) from Json: Red Green Blue Yellow Orange
Products object from Json: size 3
    Product: Intel i7-7700K Price 329.99
    Product: Ryzen-5-1600 Price 194.99
    Product: GTX-1060 Price 449.99

Json from bool_t: true
Json from uint16_t: 6654
Json from int16_t: -567
Json from real32_t: 456.450012
Json from String: "Hello World"
Json from Image: "iVBORw0KGgoAAAANSUhEUgAAAAI..."
Json from int array: [ -321, 12, -8943, 228, -220, 347 ]
Json from string array: [ "Red", "Green", "Blue", "Yellow", "Orange" ]
Json from object: {"size" : 3, "data" : [ {"description" : "Intel i7-7700K", "
    ↪ price" : 329.989990 }, {"description" : "Ryzen-5-1600", "price" :
    ↪ 194.990005 }, {"description" : "GTX-1060", "price" : 449.989990 } ] }
```

Capítulo 34

Alternativa a STL

La *Standard Template Library* de C++ provee de contenedores y algoritmos genéricos como parte del lenguaje. El problema es que no se pueden utilizar desde código C "puro", por lo que NAppGUI proporciona una implementación de Arrays y Set al menos tan eficiente como las de las STL.

Resultado en un i7-4970k Win10 x64

```
NAppGUI Containers vs STL.
- Created 2000000 elements of 328 bytes
- Starting...
- Add to ArrSt(Product) and sort: 2.160294
- Add to vector<Product> and sort: 2.499203
- Add to ArrPt(Product) and sort: 0.697777
- Add to vector<Product*> and sort: 0.541828
- Add to SetSt(Product): 2.386245
- Add to set<Product>: 2.533197
- Add to SetPt(Product): 2.861091
- Add to set<Product*>: 2.919082
```

Listado 34.1: demo/stlcmp/stlcmp.cpp

```
/* NAppGUI containers VS STL */

#include <core/coreall.h>
#include <core/arrst.hpp>
#include <core/arrpt.hpp>
#include <core/setst.hpp>
#include <core/setpt.hpp>
#include <sewer/nowarn.hxx>
#include <vector>
#include <set>
#include <algorithm>
#include <sewer/warn.hxx>

using namespace std;
```

```
struct Product
{
    uint32_t id;
    char_t code[64];
    char_t description[256];
    real32_t price;
};

DeclSt(Product);
DeclPt(Product);

/*---------------------------------------------------------------------------*/

static void i_init(Product *product, uint32_t id, real32_t price)
{
    cassert_no_null(product);
    product->id = id;
    bstd_sprintf(product->code, 64, "Code-[%d]", id);
    bstd_sprintf(product->description, 256, "Description-[%d]", id);
    product->price = price;
}

/*---------------------------------------------------------------------------*/

static Product *i_create(uint32_t id, real32_t price)
{
    Product *product = heap_new(Product);
    i_init(product, id, price);
    return product;
}

/*---------------------------------------------------------------------------*/

static int i_compare(const Product *p1, const Product *p2)
{
    return (int)p1->id - (int)p2->id;
}

/*---------------------------------------------------------------------------*/

struct i_stl_compare
{
    inline bool operator()(const Product &lhs, const Product &rhs) const
    { return lhs.id < rhs.id; }

    inline bool operator()(const Product* lhs, const Product* rhs) const
    { return lhs->id < rhs->id; }

};
```

```
/*---------------------------------------------------------------------------*/

// All stl destructors should be called before 'core_finish',
// because this function makes a Debug memory dump.
static void i_core_finish(void)
{
    core_finish();
}

/*---------------------------------------------------------------------------*/

int main(int argc, char *argv[])
{
    bool_t err;
    uint32_t n;
    uint32_t *ids;
    Product *products;
    Product **pproducts;
    ArrSt(Product) *arrst;
    ArrPt(Product) *arrpt;
    SetSt(Product) *setst;
    SetPt(Product) *setpt;
    vector<Product> stl_arrst;
    vector<Product*> stl_arrpt;
    set<Product,i_stl_compare> stl_setst;
    set<Product*,i_stl_compare> stl_setpt;
    Clock *clock;
    real64_t t;

    core_start();
    atexit(i_core_finish);

    if (argc == 2)
    {
        n = str_to_u32(argv[1], 10, &err);
        if (err == TRUE)
        {
            log_printf("Use: stlcmp [size].");
            return 0;
        }
    }
    else
    {
        n = 2000000;
    }

    bstd_printf("NAppGUI Containers vs STL.\n");

    // Create the elements. This time is out of the test
    // The elements will be shuffled randomly
    ids = heap_new_n(n, uint32_t);
```

```
    for (uint32_t i = 0; i < n; ++i)
        ids[i] = i;
    bmath_rand_seed(526);
    bmem_shuffle_n(ids, n, uint32_t);

    products = heap_new_n(n, Product);
    pproducts = heap_new_n(n, Product*);
    for (uint32_t i = 0; i < n; ++i)
    {
        i_init(&products[i], ids[i], 100.f + i);
        pproducts[i] = i_create(ids[i], 100.f + i);
    }

    arrst = arrst_create(Product);
    arrpt = arrpt_create(Product);
    setst = setst_create(i_compare, Product);
    setpt = setpt_create(i_compare, Product);

    clock = clock_create(0.);
    bstd_printf("- Created %d elements of %lu bytes\n", n, sizeof(Product));
    bstd_printf("- Starting...\n");

    // NAppGUI struct array
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
    {
        Product *p = arrst_new(arrst, Product);
        *p = products[i];
    }
    arrst_sort(arrst, i_compare, Product);
    t = clock_elapsed(clock);
    bstd_printf("- Add to ArrSt(Product) and sort: %.6f\n", t);

    // STL struct array
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
        stl_arrst.push_back(products[i]);
    sort(stl_arrst.begin(), stl_arrst.end(), i_stl_compare());
    t = clock_elapsed(clock);
    bstd_printf("- Add to vector<Product> and sort: %.6f\n", t);

    // NAppGUI pointer array
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
        arrpt_append(arrpt, pproducts[i], Product);
    arrpt_sort(arrpt, i_compare, Product);
    t = clock_elapsed(clock);
    bstd_printf("- Add to ArrPt(Product) and sort: %.6f\n", t);

    // STL pointer array
    clock_reset(clock);
```

```
    for (uint32_t i = 0; i < n; ++i)
        stl_arrpt.push_back(pproducts[i]);
    sort(stl_arrpt.begin(), stl_arrpt.end(), i_stl_compare());
    t = clock_elapsed(clock);
    bstd_printf("- Add to vector<Product*> and sort: %.6f\n", t);

    // NAppGUI struct set
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
    {
        // TODO: review 'setst_insert'. The copy makes the insertion slower
        Product *product = setst_insert(setst, &products[i], Product);
        *product = products[i];
    }
    t = clock_elapsed(clock);
    bstd_printf("- Add to SetSt(Product): %.6f\n", t);

    // STL struct set
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
        stl_setst.insert(products[i]);
    t = clock_elapsed(clock);
    bstd_printf("- Add to set<Product>: %.6f\n", t);

    // NAppGUI pointer set
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
        setpt_insert(setpt, pproducts[i], Product);
    t = clock_elapsed(clock);
    bstd_printf("- Add to SetPt(Product): %.6f\n", t);

    // STL pointer set
    clock_reset(clock);
    for (uint32_t i = 0; i < n; ++i)
        stl_setpt.insert(pproducts[i]);
    t = clock_elapsed(clock);
    bstd_printf("- Add to set<Product*>: %.6f\n", t);

    // Verify the sorting correctness
    clock_reset(clock);
    arrst_foreach(product, arrst, Product)
        if (product->id != product_i)
            bstd_printf("- Sorting error!!!!!\n");
    arrst_end();
    t = clock_elapsed(clock);
    bstd_printf("- Loop ArrSt(Product): %.6f\n", t);

    clock_reset(clock);
    for (size_t i = 0; i < stl_arrst.size(); ++i)
    {
        if (i != stl_arrst[i].id)
```

```
            bstd_printf("- Sorting error!!!!!\n");
    }
    t = clock_elapsed(clock);
    bstd_printf("- Loop vector<Product>: %.6f\n", t);

    clock_reset(clock);
    arrpt_foreach(product, arrpt, Product)
        if (product->id != product_i)
            bstd_printf("- Sorting error!!!!!\n");
    arrpt_end();
    t = clock_elapsed(clock);
    bstd_printf("- Loop ArrPt(Product): %.6f\n", t);

    clock_reset(clock);
    for (size_t i = 0; i < stl_arrpt.size(); ++i)
    {
        if (i != stl_arrpt[i]->id)
            bstd_printf("- Sorting error!!!!!\n");
    }
    t = clock_elapsed(clock);
    bstd_printf("- Loop vector<Product*>: %.6f\n", t);

    clock_reset(clock);
    setst_foreach(product, setst, Product)
        if (product->id != product_i)
            bstd_printf("- Sorting error!!!!!\n");
    setst_fornext(product, setst, Product);
    t = clock_elapsed(clock);
    bstd_printf("- Loop SetSt<Product>: %.6f\n", t);

    uint32_t ic = 0;
    clock_reset(clock);
    for (set<Product,i_stl_compare>::iterator i = stl_setst.begin(); i !=
        ↪ stl_setst.end(); ++i)
    {
        if (i->id != ic++)
            bstd_printf("- Sorting error!!!!!\n");
    }
    t = clock_elapsed(clock);
    bstd_printf("- Loop set<Product>: %.6f\n", t);

    clock_reset(clock);
    setpt_foreach(product, setpt, Product)
        if (product->id != product_i)
            bstd_printf("- Sorting error!!!!!\n");
    setpt_fornext(product, setpt, Product);
    t = clock_elapsed(clock);
    bstd_printf("- Loop SetPt<Product>: %.6f\n", t);

    ic = 0;
    clock_reset(clock);
```

```
    for (set<Product*,i_stl_compare>::iterator i = stl_setpt.begin(); i !=
        ↪ stl_setpt.end(); ++i)
    {
        if ((*i)->id != ic++)
            bstd_printf("- Sorting error!!!!!\n");
    }
    t = clock_elapsed(clock);
    bstd_printf("- Loop set<Product*>: %.6f\n", t);

    clock_destroy(&clock);
    arrst_destroy(&arrst, NULL, Product);
    arrpt_destroy(&arrpt, NULL, Product);
    setst_destroy(&setst, NULL, Product);
    setpt_destroy(&setpt, NULL, Product);

    for (uint32_t i = 0; i < n; ++i)
        heap_delete(&pproducts[i], Product);

    heap_delete_n(&products, n, Product);
    heap_delete_n(&pproducts, n, Product*);
    heap_delete_n(&ids, n, uint32_t);

    return 0;
}
```

Parte 4

Referencia de la librería

Capítulo 35

Librería Sewer

35.1. Tipos y Constantes

int8_t

Entero con signo de 8 bits. Puede representar un valor entre `INT8_MIN` y `INT8_MAX`.

int16_t

Entero con signo de 16 bits. Puede representar un valor entre `INT16_MIN` y `INT16_MAX`.

int32_t

Entero con signo de 32 bits. Puede representar un valor entre `INT32_MIN` y `INT32_MAX`.

int64_t

Entero con signo de 64 bits. Puede representar un valor entre `INT64_MIN` y `INT64_MAX`.

uint8_t

Entero sin signo de 8 bits. Puede representar un valor entre 0 y `UINT8_MAX`.

uint16_t

Entero sin signo de 16 bits. Puede representar un valor entre 0 y `UINT16_MAX`.

uint32_t

Entero sin signo de 32 bits. Puede representar un valor entre 0 y `UINT32_MAX`.

uint64_t

Entero sin signo de 64 bits. Puede representar un valor entre 0 y `UINT64_MAX`.

char_t

Tipo carácter de 8 bits (Unicode). Un solo carácter puede necesitar 1, 2, 3 o 4 elementos (bytes), dependiendo de *"Codificaciones UTFCodificaciones UTF"* (Página 159).

byte_t

Tipo de 8 bits para almacenar bloques de memoria genéricos.

bool_t

Booleano de 8 bits. Solo dos valores son correctos `TRUE` (1) y `FALSE` (0).

real

Número de punto flotante de 32 o 64 bits.

real32_t

Número de punto flotante de 32 bits. El tipo `float` de C.

real64_t

Número de punto flotante de 64 bits. El tipo `double` de C.

TRUE

Cierto, verdadero.

```
const bool_t TRUE = 1;
```

FALSE

Falso.

```
const bool_t FALSE = 0;
```

NULL

Puntero nulo.

```
const void* NULL = 0;
```

INT8_MIN

-128.

```
const int8_t INT8_MIN = 0x80;
```

INT8_MAX

127.

```
const int8_t INT8_MAX = 0x7F;
```

INT16_MIN

-32.768.

```
const int16_t INT16_MIN = 0x8000;
```

INT16_MAX

32.767.

```
const int16_t INT16_MAX = 0x7FFF;
```

INT32_MIN

-2.147.483.648.

```
const int32_t INT32_MIN = 0x80000000;
```

INT32_MAX

2.147.483.647.

```
const int32_t INT32_MAX = 0x7FFFFFFF;
```

INT64_MIN

-9.223.372.036.854.775.808.

```
const int64_t INT64_MIN = 0x8000000000000000;
```

INT64_MAX

9.223.372.036.854.775.807.

```
const int64_t INT64_MAX = 0x7FFFFFFFFFFFFFFF;
```

UINT8_MAX

255.

```
const uint8_t UINT8_MAX = 0xFF;
```

UINT16_MAX

65.535.

```
const uint16_t UINT16_MAX = 0xFFFF;
```

UINT32_MAX

4.294.967.295.

```
const uint32_t UINT32_MAX = 0xFFFFFFFF;
```

UINT64_MAX

18.446.744.073.709.551.615.

```
const uint64_t UINT64_MAX = 0xFFFFFFFFFFFFFFFF;
```

kE

El número de Euler.

```
const real32_t kBMATH_Ef = 2.718281828459045f;
const real64_t kBMATH_Ed = 2.718281828459045;
const real BMath::kE;
```

kLN2

El logaritmo natural de 2.

```
const real32_t kBMATH_LN2f = 0.6931471805599453f;
const real64_t kBMATH_LN2d = 0.6931471805599453;
const real BMath::kLN2;
```

kLN10

El logaritmo natural de 10.

```
const real32_t kBMATH_LN10f = 2.302585092994046f;
const real64_t kBMATH_LN10d = 2.302585092994046;
const real BMath::kLN10;
```

kPI

El número Pi.

```
const real32_t kBMATH_PIf = 3.141592653589793f;
const real64_t kBMATH_PId = 3.141592653589793;
const real BMath::kPI;
```

kSQRT2

Raíz cuadrada de 2.

```
const real32_t kBMATH_SQRT2f = 1.414213562373095f;
const real64_t kBMATH_SQRT2d = 1.414213562373095;
const real BMath::kSQRT2;
```

kSQRT3

Raíz cuadrada de 3.

```
const real32_t kBMATH_SQRT3f = 1.732050807568878f;
const real64_t kBMATH_SQRT3d = 1.732050807568878;
const real BMath::kSQRT3;
```

kDEG2RAD

Conversión de un grado a radianes.

```
const real32_t kBMATH_DEG2RADf = 0.017453292519943f;
const real64_t kBMATH_DEG2RADd = 0.017453292519943;
const real BMath::kDEG2RAD;
```

kRAD2DEG

Conversión de un radián a grados.

```
const real32_t kBMATH_RAD2DEGf = 57.2957795130823f;
const real64_t kBMATH_RAD2DEGd = 57.2957795130823;
const real BMath::kRAD2DEG;
```

kINFINITY

Infinito, representado por un valor muy grande.

```
const real32_t kBMATH_INFINITYf = ∞f;
const real64_t kBMATH_INFINITYd = ∞;
const real BMath::kINFINITY;
```

enum unicode_t

Representa las *"Codificaciones UTFCodificaciones UTF"* (Página 159).

ekUTF8	Codificación UTF8.
ekUTF16	Codificación UTF16.
ekUTF32	Codificación UTF32.

struct REnv

Entorno de *"Números aleatoriosNúmeros aleatorios"* (Página 162).

```
struct REnv;
```

35.2. Funciones

FPtr_destroy

Prototipo de función destructora.

```
void
(*FPtr_destroy)(type **item);
```

item Doble puntero al objeto a destruir. Debe ser puesto a `NULL` tras la destrucción para invalidar su uso.

FPtr_copy

Prototipo de constructor de copia.

```
type*
(*FPtr_copy)(const type *item);
```

item Puntero al objeto que debe ser copiado.

Retorna:

El nuevo objeto que es una copia exacta de la entrada.

FPtr_scopy

Prototipo de constructor de copia sin asignar memoria.

```
void
(*FPtr_scopy)(type *dest,
              const type *src);
```

dest Objeto destino (copia).

src Puntero al objeto que debe ser copiado (source).

Observaciones:

En esta operación de copia, la memoria que necesita el objeto ya ha sido asignada. Debemos crear memoria dinámica para los campos del objeto que lo requieran, pero no para el objeto en sí mismo. Suele utilizarse para copiar arrays de objetos (no punteros a objetos).

FPtr_compare

Prototipo de función de comparación.

```
int
(*FPtr_compare)(const type *item1,
                const type *item2);
```

item1 Primer elemento a comparar.

item2 Segundo elemento a comparar.

Retorna:

Resultado de la comparación.

FPtr_compare_ex

Similar a `FPtr_compare`, pero recibe un parámetro adicional que puede influir en la comparación.

```
int
(*FPtr_compare_ex)(const type *item1,
                   const type *item2,
                   const dtype *data);
```

item1	Primer elemento a comparar.
item2	Segundo elemento a comparar.
data	Parámetro adicional.

Retorna:

Resultado de la comparación.

FPtr_assert

Prototipo de función *callback* llamada cuando se produce un `assert`.

```
void
(*FPtr_assert)(type *item,
               const uint32_t group,
               const char_t *caption,
               const char_t *detail,
               const char_t *file,
               const uint32_t line);
```

item	Datos de usuario pasados como primer parámetro.
group	0 = Error fatal, 1 = La ejecución puede continuar.
caption	Título.
detail	Mensaje detallado.
file	Archivo fuente donde ocurrió el *assert*.
line	Línea dentro del archivo fuente.

unref

Marca el parámetro como no referenciado, desactivando los avisos del compilador.

```
void
unref(param);
```

```
static void i_OnClick(App *app, Event *e)
{
    unref(e);
    app_click_action(app);
}
```

param Parámetro.

cassert

Sentencia *assert* elemental. Si la condificón se evalua a `FALSE`, se lanzará un *assert* "continuable". El mensaje mostrado será el literal de la propia condición.

```
void
cassert(bool_t cond);
```

```
// "row < arrpt_size(layout->rows)"
// will be shown in the assert window
cassert(row < arrpt_size(layout->rows));
```

cond Expresión booleana.

cassert_msg

Igual que la sentencia `cassert()`, pero utilizando un mensaje personalizado, en lugar del literal de la condición.

```
void
cassert_msg(bool_t cond,
            const char_t *msg);
```

```
// "'row' out of range"
// will be shown in the assert window
cassert_msg(layout < layout->num_rows, "'row' out of range");
```

cond Expresión booleana.

msg Mensaje relacionado con el *assert*.

cassert_fatal

Igual que la sentencia `cassert()`, pero lanzando un assert **crítico** (no "continuable").

```
void
cassert_fatal(bool_t cond);
```

```
// "gravity > 0."
// will be shown in the assert window
cassert_fatal(gravity > 0.);
```

cond Expresión booleana.

cassert_fatal_msg

Igual que la sentencia `cassert_msg()`, pero lanzando un assert **crítico** (no "continuable".

```
void
cassert_fatal_msg(bool_t cond,
                  const char_t *msg);
```

```
// "'gravity' can't be negative."
// will be shown in the assert window
cassert_fatal_msg(gravity > 0., "'gravity' can't be negative");
```

cond Expresión booleana.

msg Mensaje relacionado con el *assert*.

cassert_no_null

Lanza un *assert* crítico si un puntero tiene valor `NULL`.

```
void
cassert_no_null(void *ptr);
```

ptr Puntero a evaluar.

cassert_no_nullf

Lanza un *assert* crítico si un puntero **a función** tiene valor `NULL`.

```
void
cassert_no_nullf(void *fptr);
```

fptr Puntero a evaluar.

cassert_default

Lanza un *assert* "continuable" si la sentencia **switch** alcanza el estado `default:` Útil para asegurar que, por ejemplo, todos los valores de un enum han sido considerados.

```
void
cassert_default(void);
```

```
switch(align) {
case LEFT:
    // Do something
    break;
case RIGHT:
    // Do something
    break;
// Others are not allowed.
cassert_default();
}
```

cassert_set_func

Establece una función personalizada para que ejecute un código alternativo cuando se produce un *assert*. Por defecto, en aplicaciones de escritorio, se muestra una ventana informativa (Figura 13.4) y se guarda el mensaje en una fichero *"Log"* (Página 186).

```
void
cassert_set_func(void *data,
                 FPtr_assert func_assert);
```

data	Datos de usuario o contexto de aplicación.
func_assert	Función *callback* llamada tras la activación de un *assert*.

Observaciones:

Al utilizar esta función se desactivará la gestión de *asserts* previa.

ptr_get

Acceso al contenido del puntero (dereferencia), verificando previamente que no sea `NULL`.

```
void
ptr_get(type *ptr,
        type);
```

```
void compute(const V2Df *v1, const V2Df *v2)
{
    /* Safer than t = *v1; */
```

```
    V2Df t = ptr_get(v1, V2Df);
    ...
}
```

ptr Puntero.

type Tipo de puntero.

ptr_dget

Accede al contenido de un doble puntero, invalidándolo posteriormente.

```
void
ptr_dget(type **ptr,
         type);
```

```
Ctrl *create(Model **model, View **view)
{
    Ctrl *ctrl = heap_new(Ctrl);
    ctrl->model = ptr_dget(model, Model);
    ctrl->view = ptr_dget(view, View);
    // *model = NULL
    // *view = NULL
    return ctrl;
}
```

ptr Double puntero.

type Tipo de puntero.

ptr_dget_no_null

Igual que `ptr_dget`, pero el contenido del double puntero (`*dptr`) no puede ser `NULL`.

```
void
ptr_dget_no_null(type **ptr,
                 type);
```

```
Ctrl *create(Model **model, View **view)
{
    // *model and *view can't be NULL
    Ctrl *ctrl = heap_new(Ctrl);
    ctrl->model = ptr_dget_no_null(model, Model);
    ctrl->view = ptr_dget_no_null(view, View);
    return ctrl;
}
```

ptr Double puntero.

type Tipo de puntero.

ptr_assign

Asigna el contenido de un puntero a otro, si el destino no es NULL.

```
void
ptr_assign(dest,
           src);
```

dest Puntero destino.

src Puntero origen.

ptr_destopt

Destruye un objeto, si no es NULL.

```
void
ptr_destopt(FPtr_destroy func_destroy,
            type dptr,
            type);
```

```
cassert_no_null(dptr);
if (*dptr != NULL)
{
    func_destroy(*dptr);
    *dptr = NULL;
}
```

func_destroy Destructor.

dptr Doble puntero al objeto a destruir.

type Tipo de objeto.

ptr_copyopt

Copia el objeto si no es NULL.

```
void
ptr_copyopt(FPtr_copy func_copy,
            type ptr,
            type);
```

```
if (ptr != NULL)
    return func_copy(ptr);
else
    return NULL;
```

func_copy	Constructor de copia.
ptr	Objeto a copiar (origen).
type	Tipo de objeto.

unicode_convers

Convierte una cadena Unicode de una codificación a otra.

```
uint32_t
unicode_convers(const char_t *from_str,
                char_t *to_str,
                const unicode_t from,
                const unicode_t to,
                const uint32_t osize);
```

```
const char32_t str[] = U"Hello World";
char_t utf8_str[256];
unicode_convers((const char_t*)str, utf8_str, ekUTF32, ekUTF8, 256);
```

from_str	Cadena de origen (terminada en carácter nulo '\0').
to_str	Buffer de destino.
from	Codificación de cadena origen.
to	Codificación requerida en `to_str`.
osize	Tamaño del búfer de salida. Número máximo de bytes que se escribirán en `to_str`, incluido el carácter nulo (`'\0'`). Si la cadena original no se puede copiar en su totalidad, se cortará y se agregará el carácter nulo.

Retorna:

Número de bytes escritos en `to_str` (incluido el carácter nulo).

unicode_convers_n

Igual que `unicode_convers`, pero indicando un tamaño máximo para la cadena de entrada.

```
uint32_t
unicode_convers_n(const char_t *from_str,
                  char_t *to_str,
                  const unicode_t from,
                  const unicode_t to,
                  const uint32_t isize,
                  const uint32_t osize);
```

from_str	Cadena de origen.
to_str	Buffer de destino.
from	Codificación de cadena origen.
to	Codificación requerida en `to_str`.
isize	Tamaño de la cadena en entrada (en bytes).
osize	Tamaño del búfer de salida.

Retorna:

Número de bytes escritos en `to_str`.

unicode_convers_nbytes

Calcula el número de bytes necesarios para convertir una cadena Unicode de una codificación a otra. Será útil calcular el espacio necesario en reserva dinámica de memoria.

```
uint32_t
unicode_convers_nbytes(const char_t *str,
                       const unicode_t from,
                       const unicode_t to);
```

```
const char32_t str[] = U"Hello World";
uint32_t size = unicode_convers_nbytes((char_t*)str, ekUTF32, ekUTF8);
/ * size == 12 * /
```

str	Cadena de origen (terminada en nulo).
from	Codificación de `str`.
to	Codificación requerida.

Retorna:

Número de bytes necesarios (incluido el carácter nulo).

unicode_nbytes

Obtiene el tamaño (en bytes) de una cadena Unicode.

```
uint32_t
unicode_nbytes(const char_t *str,
               const unicode_t format);
```

str	Cadena Unicode (terminada en '\0').
format	Codificación de `str`.

Retorna:

El tamaño en bytes ('\0' incluido).

unicode_nchars

Calcula la longitud (en caracteres) de una cadena Unicode.

```
uint32_t
unicode_nchars(const char_t *str,
               const unicode_t format);
```

str Cadena Unicode (terminada en '\0').

format Codificación de `str`.

Retorna:

El número de caracteres ('\0' **no incluido**).

Observaciones:

En cadenas ASCII, el número de bytes es igual al número de caracteres. En Unicode depende de la codificación y la cadena.

unicode_to_u32

Obtiene el valor del primer *codepoint* de la cadena Unicode.

```
uint32_t
unicode_to_u32(const char_t *str,
               const unicode_t format);
```

```
char_t str[] = "áéíóúÄÑ£";
uint32_t cp = unicode_to_u32(str, ekUTF8);
/* cp == 'á' == 225 == U+E1 */
```

str Cadena Unicode (terminada en '\0').

format Codificación de `str`.

Retorna:

El código del primer carácter de `str`.

unicode_to_u32b

Igual que `unicode_to_u32` pero con con campo adicional para almacenar la cantidad de bytes que ocupa el codepoint.

```
uint32_t
unicode_to_u32b(const char_t *str,
                const unicode_t format,
                uint32_t *bytes);
```

str Cadena Unicode (terminada en '\0').

format Codificación de `str`.

bytes Guarda el número de bytes necesarios para representar el codepoint mediante `format`.

Retorna:

El código del primer carácter de `str`.

unicode_to_char

Escribe el `codepoint` al comienzo de `str`, utilizando la codificación `format`.

```
uint32_t
unicode_to_char(const uint32_t codepoint,
                char_t *str,
                const unicode_t format);
```

```
char_t str[64] = \"\";
uint32_t n = unicode_to_char(0xE1, str, ekUTF8);
unicode_to_char(0, str + n, ekUTF8);
/* str == "á" */
/* n = 2 */
```

codepoint Código del carácter.

str Cadena de destino.

format Codificación para `codepoint`.

Retorna:

El número de bytes escritos (1, 2, 3 or 4).

Observaciones:

Para escribir varios *codepoints*, combinar `unicode_to_char` con `unicode_next`.

unicode_valid_str

Comprueba si una cadena es Unicode.

```
bool_t
unicode_valid_str(const char_t *str,
                  const unicode_t format);
```

str Cadena a comprobar (terminada en '\0').

format Codificación Unicode esperada.

Retorna:

`TRUE` si es válida.

unicode_valid_str_n

Igual que `unicode_valid_str`, pero indicando un tamaño máximo para la cadena de entrada.

```
bool_t
unicode_valid_str_n(const char_t *str,
                    const uint32_t size,
                    const unicode_t format);
```

str Cadena a comprobar.

size Máximo tamaño de la cadena (en bytes).

format Codificación Unicode esperada.

Retorna:

`TRUE` si es válida.

unicode_valid

Comprueba si un *codepoint* es válido.

```
bool_t
unicode_valid(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

`TRUE` si el parámetro es un *codepoint* válido. De lo contrario, `FALSE`.

unicode_next

Avanza al siguiente carácter de una cadena Unicode. En general no es posible el acceso aleatorio como hacemos en ANSI-C (`str[i++]`). Debemos iterar una cadena desde el principio. Ver *"Codificaciones UTFCodificaciones UTF"* (Página 159).

```
const char_t*
unicode_next(const char_t *str,
             const unicode_t format);
```

```
char_t str[] = "áéíóúÄ";
char_t *iter = str;                  /* iter == "áéíóúÄ" */
iter = unicode_next(iter, ekUTF8);   /* iter == "éíóúÄ" */
iter = unicode_next(iter, ekUTF8);   /* iter == "íóúÄ" */
iter = unicode_next(iter, ekUTF8);   /* iter == "óúÄ" */
iter = unicode_next(iter, ekUTF8);   /* iter == "úÄ" */
iter = unicode_next(iter, ekUTF8);   /* iter == "Ä" */
iter = unicode_next(iter, ekUTF8);   /* iter == "" */
iter = unicode_next(iter, ekUTF8);   /* Segmentation fault!! */
```

str Cadena Unicode.

format Codificación de `str`.

Retorna:

Puntero al siguiente carácter de la cadena.

Observaciones:

No verifica el final de la cadena. Debemos detener la iteración cuando `codepoint == 0`.

unicode_back

Retrocede al carácter anterior de una cadena Unicode.

```
const char_t*
unicode_back(const char_t *str,
             const unicode_t format);
```

str Cadena Unicode.

format Codificación de `str`.

Retorna:

Puntero al carácter anterior de la cadena.

Observaciones:

No verifica el inicio de la cadena.

unicode_isascii

Comprueba si `codepoint` es un carácter US-ASCII 7.

```
bool_t
unicode_isascii(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

unicode_isalnum

Comprueba si `codepoint` es un carácter alfanumérico.

```
bool_t
unicode_isalnum(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isalpha

Comprueba si `codepoint` es un carácter alfabético.

```
bool_t
unicode_isalpha(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_iscntrl

Comprueba si `codepoint` es un carácter de control.

```
bool_t
unicode_iscntrl(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isdigit

Comprueba si `codepoint` es dígito (0-9).

```
bool_t
unicode_isdigit(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isgraph

Comprueba si `codepoint` es un carácter imprimible (excepto el espacio blanco ' ').

```
bool_t
unicode_isgraph(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isprint

Comprueba si `codepoint` es un carácter imprimible (incluido el espacio blanco ' ').

```
bool_t
unicode_isprint(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_ispunct

Comprueba si `codepoint` es un carácter imprimible (expecto el espacio blanco ' ' y alfanuméricos).

```
bool_t
unicode_ispunct(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isspace

Comprueba si `codepoint` es un carácter de espaciado, nueva línea, retorno de carro, tabulador horizontal o vertical.

```
bool_t
unicode_isspace(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isxdigit

Comprueba si `codepoint` es un dígito hexadecimal **0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F**.

```
bool_t
unicode_isxdigit(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_islower

Comprueba si `codepoint` es una letra minúscula.

```
bool_t
unicode_islower(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_isupper

Comprueba si `codepoint` es una letra mayúscula.

```
bool_t
unicode_isupper(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

Resultado del test.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_tolower

Convierte una letra a minúscula.

```
uint32_t
unicode_tolower(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

La conversión a minúscula si la entrada es una letra mayúscula. De lo contrario, el mismo `codepoint`.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

unicode_toupper

Convierte una letra a mayúscula.

```
uint32_t
unicode_toupper(const uint32_t codepoint);
```

codepoint El código Unicode del carácter.

Retorna:

La conversión a mayúscula si la entrada es una letra minúscula. De lo contrario, el mismo `codepoint`.

Observaciones:

Solo tiene en cuenta caracteres US-ASCII.

bmath_cos

Obtiene el coseno de un ángulo.

```
real32_t
bmath_cosf(const real32_t angle);

real64_t
bmath_cosd(const real64_t angle);

real
BMath::cos(const real angle);
```

angle Ángulo en radianes.

Retorna:

El coseno del ángulo.

bmath_sin

Obtiene el seno de un ángulo.

```
real32_t
bmath_sinf(const real32_t angle);

real64_t
bmath_sind(const real64_t angle);

real
BMath::sin(const real angle);
```

angle Ángulo en radianes.

Retorna:

El seno del ángulo.

bmath_tan

Obtiene la tangente de un ángulo.

```
real32_t
bmath_tanf(const real32_t angle);

real64_t
bmath_tand(const real64_t angle);

real
BMath::tan(const real angle);
```

angle Ángulo en radianes.

Retorna:

La tangente del ángulo.

bmath_acos

Obtiene el arco coseno, o coseno inverso, que es el ángulo cuyo coseno es el valor.

```
real32_t
bmath_acosf(const real32_t cos);

real64_t
bmath_acosd(const real64_t cos);

real
BMath::acos(const real cos);
```

cos Coseno (-1, 1).

Retorna:

El ángulo (0, Pi).

bmath_asin

Obtiene el arco seno, o seno inverso, que es el ángulo cuyo seno es el valor.

```
real32_t
bmath_asinf(const real32_t sin);

real64_t
bmath_asind(const real64_t sin);

real
BMath::asin(const real sin);
```

sin Seno (-1, 1).

Retorna:

El ángulo (0, Pi).

bmath_atan2

Obtiene el arco tangente, o tangente inversa. Es es el ángulo medido desde el eje X hasta la línea que contiene el origen (0, 0) y el punto con las coordenadas (x,y).

```
real32_t
bmath_atan2f(const real32_t y,
```

```
              const real32_t x);

real64_t
bmath_atan2d(const real64_t y,
              const real64_t x);

real
BMath::atan2(const real y,
              const real x);
```

y Coordenada Y.

x Coordenada X.

Retorna:

El ángulo (-Pi, Pi).

bmath_norm_angle

Normaliza un ángulo, es decir, retorna el mismo ángulo expresado en el rango (-Pi, Pi).

```
real32_t
bmath_norm_anglef(const real32_t a);

real64_t
bmath_norm_angled(const real64_t a);

real
BMath::norm_angle(const real a);
```

a El ángulo en radianes.

Retorna:

El ángulo (-Pi, Pi).

bmath_sqrt

Obtiene la raíz cuadrada de un número.

```
real32_t
bmath_sqrtf(const real32_t value);

real64_t
bmath_sqrtd(const real64_t value);

real
BMath::sqrt(const real value);
```

value El número.

Retorna:

La ráiz cuadrada.

bmath_isqrt

Obtiene la raíz cuadrada inversa de un número (1/sqrt).

```
real32_t
bmath_isqrtf(const real32_t value);

real64_t
bmath_isqrtd(const real64_t value);

real
BMath::isqrt(const real value);
```

value El número.

Retorna:

La raíz cuadrada inversa.

bmath_log

Obtiene el logaritmo natural (base e) de un número.

```
real32_t
bmath_logf(const real32_t value);

real64_t
bmath_logd(const real64_t value);

real
BMath::log(const real value);
```

value El número.

Retorna:

El logaritmo.

bmath_log10

Obtiene el logaritmo en base 10 de un número.

```
real32_t
bmath_log10f(const real32_t value);

real64_t
bmath_log10d(const real64_t value);

real
BMath::log10(const real value);
```

value El número.

Retorna:

El logaritmo.

bmath_exp

Obtiene el número de Euler e (2.7182818) elevado a una potencia.

```
real32_t
bmath_expf(const real32_t value);

real64_t
bmath_expd(const real64_t value);

real
BMath::exp(const real value);
```

value El exponente.

Retorna:

La exponencial.

bmath_pow

Calcula una potencia, `base` elevado a `exponent`.

```
real32_t
bmath_powf(const real32_t base,
           const real32_t exponent);

real64_t
bmath_powd(const real64_t base,
           const real64_t exponent);

real
BMath::pow(const real base,
           const real exponent);
```

base Base.

exponent Exponente.

Retorna:

El resultado de la potencia.

bmath_abs

Obtiene el valor absoluto de un número.

```
real32_t
bmath_absf(const real32_t value);

real64_t
bmath_absd(const real64_t value);

real
BMath::abs(const real value);
```

value El número.

Retorna:

El valor absoluto.

bmath_max

Obtiene el máximo de dos valores.

```
real32_t
bmath_maxf(const real32_t value1,
           const real32_t value2);

real64_t
bmath_maxd(const real64_t value1,
           const real64_t value2);

real
BMath::max(const real value1,
           const real value2);
```

value1 Primer número.

value2 Segundo número.

Retorna:

El valor máximo.

bmath_min

Obtiene el mínimo de dos valores.

```
real32_t
bmath_minf(const real32_t value1,
           const real32_t value2);

real64_t
bmath_mind(const real64_t value1,
           const real64_t value2);

real
BMath::min(const real value1,
           const real value2);
```

value1 Primer número.

value2 Segundo número.

Retorna:

El valor mínimo.

bmath_clamp

Restringe un valor a un determinado rango.

```
real32_t
bmath_clampf(const real32_t value,
             const real32_t min,
             const real32_t max);

real64_t
bmath_clampd(const real64_t value,
             const real64_t min,
             const real64_t max);

real
BMath::clamp(const real value,
             const real min,
             const real max);
```

value El número.

min Mínimo valor del rango.

max Máximo valor del rango.

Retorna:

El valor acotado.

bmath_mod

Obtiene el módulo de dividir `num/den`.

```
real32_t
bmath_modf(const real32_t num,
           const real32_t den);

real64_t
bmath_modd(const real64_t num,
           const real64_t den);

real
BMath::mod(const real num,
           const real den);
```

num Numerador.

den Denominador.

Retorna:

El módulo.

bmath_modf

Obtiene la parte entera y fracción de un número real.

```
real32_t
bmath_modff(const real32_t value,
            real32_t *intpart);

real64_t
bmath_modfd(const real64_t value,
            real64_t *intpart);

real
BMath::modf(const real value,
            real *intpart);
```

value El número.

intpart Obtiene la parte entera.

Retorna:

La parte fraccionaria `[0,1)`.

bmath_prec

Obtiene el número de decimales (precisión) de un número real.

```
uint32_t
bmath_precf(const real32_t value);

uint32_t
bmath_precd(const real64_t value);

uint32_t
BMath::prec(const real value);
```

value El número.

Retorna:

El número de posiciones decimales.

bmath_round

Redondea un número al entero más cercano (por encima o por debajo).

```
real32_t
bmath_roundf(const real32_t value);

real64_t
bmath_roundd(const real64_t value);

real
BMath::round(const real value);
```

value El número.

Retorna:

El entero más cercano.

bmath_round_step

Redondea un número a la fracción más cercana.

```
real32_t
bmath_round_stepf(const real32_t value,
                  const real32_t step);

real64_t
bmath_round_stepd(const real64_t value,
                  const real64_t step);

real
BMath::round_step(const real value,
                  const real step);
```

value El número.

step La fracción.

Retorna:

El número más cercano.

bmath_floor

Redondea un número al entero por debajo.

```
real32_t
bmath_floorf(const real32_t value);

real64_t
bmath_floord(const real64_t value);

real
BMath::floor(const real value);
```

value El número.

Retorna:

El mayor número entero, menor o igual que el número.

bmath_ceil

Redondea un número al entero por encima.

```
real32_t
bmath_ceilf(const real32_t value);

real64_t
bmath_ceild(const real64_t value);

real
BMath::ceil(const real value);
```

value El número.

Retorna:

El menor número entero, mayor o igual que el número.

bmath_rand_seed

Establece una nueva semilla de números aleatorios.

```
void
bmath_rand_seed(const uint32_t seed);
```

seed La nueva semilla.

Observaciones:

Cada vez que cambia la semilla, comienza una nueva secuencia de números aleatorios. Para la misma semilla, obtendremos la misma secuencia, por lo que son números pseudo-aleatorios. Semillas similares (pe. `4`, `5`) producen secuencias radicalmente diferentes. Utiliza `bmath_rand_env` en aplicaciones multi-hilo.

bmath_rand

Obtiene un número real aleatorio, dentro de un intervalo.

```
real32_t
bmath_randf(const real32_t from,
            const real32_t to);

real64_t
bmath_randd(const real64_t from,
            const real64_t to);

real
BMath::rand(const real from,
            const real to);
```

from El límite inferior del intervalo.

to El límite superior del intervalo.

Retorna:

El número aleatorio.

bmath_randi

Obtiene un número entero aleatorio, dentro de un intervalo.

```
uint32_t
bmath_randi(const uint32_t from,
            const uint32_t to);
```

from El límite inferior del intervalo.

to El límite superior del intervalo.

Retorna:

El número aleatorio.

bmath_rand_env

Crea un entorno *thread-safe* para números aleatorios.

```
REnv*
bmath_rand_env(const uint32_t seed);
```

seed La semilla.

Retorna:

El entorno.

bmath_rand_destroy

Destruye un entorno de números aleatorios.

```
void
bmath_rand_destroy(REnv **env);
```

env El entorno. Será puesto a `NULL` tras la destrucción.

bmath_rand_mt

Obtiene un número real aleatorio, dentro de un intervalo.

```
real32_t
bmath_rand_mtf(REnv *env,
               const real32_t from,
               const real32_t to);

real64_t
bmath_rand_mtd(REnv *env,
               const real64_t from,
               const real64_t to);

real
BMath::rand_mt(REnv *env,
               const real from,
               const real to);
```

env El entorno de números aleatorios.

from El límite inferior del intervalo.

to El límite superior del intervalo.

Retorna:

El número aleatorio.

bmath_rand_mti

Obtiene un número entero aleatorio, dentro de un intervalo.

```
uint32_t
bmath_rand_mti(REnv *env,
               const uint32_t from,
               const uint32_t to);
```

env El entorno de números aleatorios.

from El límite inferior del intervalo.

to El límite superior del intervalo.

Retorna:

El número aleatorio.

blib_strlen

Retorna la longitud en bytes de una cadena de texto.

```
uint32_t
blib_strlen(const char_t *str);
```

str Cadena acabada en carácter nulo '\0'.

Retorna:

Longitud de la cadena sin incluir el carácter nulo.

Observaciones:

Ver *"Unicode"* (Página 157), el número de bytes no es equivalente al número de caracteres.

blib_strstr

Busca una subcadena dentro de una cadena más larga.

```
const char_t*
blib_strstr(const char_t *str,
            const char_t *substr);
```

str Cadena acabada en carácter nulo '\0'.

substr Subcadena a buscar acabada en carácter nulo '\0'.

Retorna:

Puntero al inicio de la primera subcadena encontrada o `NULL` si no existe ninguna.

blib_strcpy

Copia el contenido de una cadena en otra.

```
void
blib_strcpy(char_t *dest,
            const uint32_t size,
            const char_t *src);
```

dest Buffer destino.

size Tamaño del buffer destino en bytes.

src Cadena a copiar acabada en carácter nulo '\0'.

Observaciones:

Solo los primeros `size-1` bytes serán copiados, en el caso de que `src` sea más larga que la capacidad de `dest`.

blib_strncpy

Copia los primeros `n` bytes de na cadena en otra.

```
void
blib_strncpy(const char_t *dest,
             const uint32_t size,
             const char_t *src,
             const uint32_t n);
```

dest Buffer destino.

size Tamaño del buffer destino en bytes.

src Cadena a copiar acabada en carácter nulo '\0'.

n Número de bytes a copiar.

Observaciones:

Solo los primeros `size-1` bytes serán copiados, en el caso de que `n` sea mayor que `size`.

blib_strcat

Concatena dos cadenas.

```
void
blib_strcat(char_t *dest,
            const uint32_t size,
            const char_t *src);
```

dest Buffer origen y destino.

size Tamaño del buffer destino en bytes.

src Cadena a añadir a `dest`, acabada en carácter nulo '\0'.

Observaciones:

No se sobrepasarán los `size-1` bytes en `dest`, por lo que la concatenación será truncada si es necesario.

blib_strcmp

Compara dos cadenas.

```
int
blib_strcmp(const char_t *str1,
            const char_t *str2);
```

str1 Primera cadena a comparar, acabada en carácter nulo '\0'.

str2 Segunda cadena a comparar, acabada en carácter nulo '\0'.

Retorna:

Resultado de la comparación.

blib_strncmp

Compara los primeros `n` bytes de dos cadenas.

```
int
blib_strncmp(const char_t *str1,
             const char_t *str2,
             const uint32_t n);
```

str1 Primera cadena a comparar, acabada en carácter nulo '\0'.

str2 Segunda cadena a comparar, acabada en carácter nulo '\0'.

n Número máximo de bytes a comparar.

Retorna:

Resultado de la comparación.

blib_strtol

Transforma una cadena de texto en un número entero.

```
int64_t
blib_strtol(const char_t *str,
            char_t **endptr,
            uint32_t base,
            bool_t *err);
```

str Cadena que comienza con un número entero.

endptr Puntero cuyo valor será el primer carácter tras el número. Puede ser NULL.

base Base de numeración: 2, 8, 10, 16.

err Se asigna valor TRUE si hay algún error en el análisis de la cadena. Pueder ser NULL.

Retorna:

Número resultado del análisis de la cadena.

blib_strtoul

Transforma una cadena de texto en un número entero sin signo.

```
uint64_t
blib_strtoul(const char_t *str,
             char_t **endptr,
             uint32_t base,
             bool_t *err);
```

str Cadena que comienza con un número entero.

endptr Puntero cuyo valor será el primer carácter tras el número. Puede ser NULL.

base Base de numeración: 2, 8, 10, 16.

err Se asigna valor TRUE si hay algún error en el análisis de la cadena. Pueder ser NULL.

Retorna:

Número resultado del análisis de la cadena.

blib_strtof

Transforma una cadena de texto en un número real de 32 bits.

```
real32_t
blib_strtof(const char_t *str,
            char_t **endptr,
            bool_t *err);
```

str Cadena que comienza con un número real.

endptr Puntero cuyo valor será el primer carácter tras el número. Puede ser NULL.

err Se asigna valor TRUE si hay algún error en el análisis de la cadena. Pueder ser NULL.

Retorna:

Número resultado del análisis de la cadena.

blib_strtod

Transforma una cadena de texto en un número real de 64 bits.

```
real64_t
blib_strtod(const char_t *str,
            char_t **endptr,
            bool_t *err);
```

str Cadena que comienza con un número entero.

endptr Puntero cuyo valor será el primer carácter tras el número. Puede ser NULL.

err Se asigna valor TRUE si hay algún error en el análisis de la cadena. Pueder ser NULL.

Retorna:

Número resultado del análisis de la cadena.

blib_qsort

Ordena un vector de elementos utilizando el algoritmo *QuickSort*.

```
void
blib_qsort(byte_t *array,
           const uint32_t nelems,
           const uint32_t size,
           FPtr_compare func_compare);
```

array	Vector de elementos.
nelems	Número de elementos.
size	Tamaño de cada elemento.
func_compare	Función de comparación.

blib_qsort_ex

Ordena un vector de elementos utilizando el algoritmo *QuickSort*.

```
void
blib_qsort_ex(byte_t *array,
              const uint32_t nelems,
              const uint32_t size,
              FPtr_compare_ex func_compare,
              const byte_t *data);
```

array	Vector de elementos.
nelems	Número de elementos.
size	Tamaño de cada elemento.
func_compare	Función de comparación que acepta datos extra.
data	Datos extra que serán pasados en cada comparación.

blib_bsearch

Busca un elemento en un vector ordenado.

```
bool_t
blib_bsearch(const byte_t *array,
             const byte_t *key,
             const uint32_t nelems,
             const uint32_t size,
             FPtr_compare func_compare,
             uint32_t *pos);
```

array	Vector de elementos.
key	Clave de búsqueda.
nelems	Número de elementos.
size	Tamaño de cada elemento.
func_compare	Función de comparación.
pos	Posición del elemento encontrado. Puede ser NULL.

Retorna:

TRUE si se ha encontrado el elemento.

blib_bsearch_ex

Busca un elemento en un vector ordenado.

```
bool_t
blib_bsearch_ex(const byte_t *array,
                const byte_t *key,
                const uint32_t nelems,
                const uint32_t size,
                FPtr_compare_ex func_compare,
                const byte_t *data,
                uint32_t *pos);
```

array	Vector de elementos.
key	Clave de búsqueda.
nelems	Número de elementos.
size	Tamaño de cada elemento.
func_compare	Función de comparación que acepta datos extra.
data	Datos extra que serán pasados en cada comparación.
pos	Posición del elemento encontrado. Puede ser NULL.

Retorna:

TRUE si se ha encontrado el elemento.

blib_atexit

Añade una función que será llamada cuando el programa termine.

```
void
```

```
blib_atexit(void()(void) *func);
```

func Función.

blib_abort

Termina de forma abrupta la ejecución del programa.

```
void
blib_abort(void);
```

Observaciones:

No se liberan recursos ni se realiza un cierre controlado. El único caso donde se justifica su uso es salir del programa tras detectar un error irrecuperable (p.e `NULL` pointer).

blib_debug_break

Detiene la ejecución del programa en el punto donde se encuentra la función y devuelve el control de depurador para que podamos inspeccionar la pila, variables, etc.

```
void
blib_debug_break(void);
```

bstd_sprintf

Escribe una cadena con formato del `printf` en un búfer de memoria.

```
uint32_t
bstd_sprintf(char_t *str,
             const uint32_t size,
             const char_t *format,
             ...);
```

str Puntero al búfer donde será escrito el resultado. Terminará en carácter nulo `'\0'`.

size Tamaño de `str` en bytes.

format Cadena con el formato printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El número de bytes escritos, sin incluir el carácter nulo `'\0'`.

Observaciones:

Es una función segura y no escribirá más de `size` bytes. Para obtener el tamaño necesario de `str`, llamar a esta función con `str=NULL` y `size=0`.

bstd_vsprintf

Igual que `bstd_sprintf` pero con la lista de argumentos ya resuelta.

```
uint32_t
bstd_vsprintf(char_t *str,
              const uint32_t size,
              const char_t *format,
              va_list args);
```

str	Puntero al búfer donde será escrito el resultado. Terminará en carácter nulo `'\0'`.
size	Tamaño de `str` en bytes.
format	Cadena con el formato tipo-printf con un número variable de parámetros.
args	Argumentos.

Retorna:

El número de bytes escritos, sin incluir el carácter nulo `'\0'`.

Observaciones:

Es una función segura y no escribirá más de `size` bytes.

bstd_printf

Escribe una cadena con formato en la salida estándar `(stdout)`. Es equivalente a la función `printf` de la librería estándar.

```
uint32_t
bstd_printf(const char_t *format,
            ...);
```

format	Cadena con el formato tipo-printf con un número variable de parámetros.
...	Argumentos o variables del printf.

Retorna:

El número de bytes escritos en `stdout`.

bstd_eprintf

Escribe una cadena con formato en la salida de error `(stderr)`.

```
uint32_t
bstd_eprintf(const char_t *format,
             ...);
```

format Cadena con el formato tipo-printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El número de bytes escritos en `stderr`.

bstd_writef

Escribe una cadena C UTF 8 en la salida estándar `(stdout)`.

```
uint32_t
bstd_writef(const char_t *str);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

El número de bytes escritos en `stdout`.

bstd_ewritef

Escribe una cadena C UTF 8 en la salida de error `(stderr)`.

```
uint32_t
bstd_ewritef(const char_t *str);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

El número de bytes escritos en `stderr`.

bstd_read

Lee datos desde la entrada estándar `stdin`.

```
bool_t
bstd_read(byte_t *data,
          const uint32_t size,
          uint32_t *rsize);
```

data Búfer donde se escribirán los datos leídos.

size El número de bytes máximos a leer (tamaño del búfer).

rsize Recibe el número de bytes leídos realmente. Puede ser NULL.

Retorna:

TRUE si se han leído datos. FALSE si ha ocurrido algún error.

Observaciones:

"Standard streamStandard stream" (Página 200) implementa funciones de alto nivel para lectura/escritura en canales estándar.

bstd_write

Escribe datos en la salida estándar `stdout`.

```
bool_t
bstd_write(const byte_t *data,
           const uint32_t size,
           uint32_t *wsize);
```

data Búfer que contiene los datos a escribir.

size El número de bytes a escribir.

wsize Recibe el número de bytes escritos realmente. Puede ser NULL.

Retorna:

TRUE si se han escrito datos. FALSE si ha ocurrido algún error.

Observaciones:

"Standard streamStandard stream" (Página 200) implementa funciones de alto nivel para lectura/escritura en canales estándar.

bstd_ewrite

Escribe datos en la salida de error `stderr`.

```
bool_t
bstd_ewrite(const byte_t *data,
            const uint32_t size,
            uint32_t *wsize);
```

data	Búfer que contiene los datos a escribir.
size	El número de bytes a escribir.
wsize	Recibe el número de bytes escritos realmente. Puede ser `NULL`.

Retorna:

`TRUE` si se han escrito datos. `FALSE` si ha ocurrido algún error.

Observaciones:

"Standard streamStandard stream" (Página 200) implementa funciones de alto nivel para lectura/escritura en canales estándar.

bmem_malloc

Reserva un bloque de memoria con la alineación por defecto `sizeof(void*)`.

```
byte_t*
bmem_malloc(const uint32_t size);
```

size	Tamaño en bytes del bloque.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `bmem_free` cuando ya no sea necesario.

Observaciones:

Utiliza *"Heap - Gestor de memoria"* (Página 192) para reservas más eficientes y seguras.

bmem_realloc

Realoja un bloque de memoria existente debido a la expansión o reducción del mismo. Garantiza que se conserva el contenido previo del bloque `min(size, new_size)`. Intenta hacerlo sin mover memoria (in situ), pero si no es posible busca una nueva zona. También garantiza la alineación por defecto `sizeof(void*)` si hay que reservar un nuevo bloque.

```
byte_t*
bmem_realloc(byte_t *mem,
             const uint32_t size,
             const uint32_t new_size);
```

mem	Puntero al bloque original a realojar.
size	Tamaño en bytes del bloque original `mem`.
new_size	Nuevo tamaño requerido, en bytes.

Retorna:

Puntero al bloque realojado. Será el mismo que el puntero original `mem` si la reubicación "in-situ" ha tenido éxito. Debe ser liberado con `bmem_free` cuando ya no sea necesario.

Observaciones:

Utiliza *"Heap - Gestor de memoria"* (Página 192) para reservas más eficientes y seguras.

bmem_aligned_malloc

Reserva un bloque de memoria con alineación.

```
byte_t*
bmem_aligned_malloc(const uint32_t size,
                    const uint32_t align);
```

size	Tamaño en bytes del bloque.
align	Alineación. Debe ser potencia de 2.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `bmem_free` cuando ya no sea necesario.

Observaciones:

Utiliza *"Heap - Gestor de memoria"* (Página 192) para reservas más eficientes y seguras.

bmem_aligned_realloc

Igual que `bmem_realloc`, pero garantiza una alineación concreta.

```
byte_t*
bmem_aligned_realloc(byte_t *mem,
                     const uint32_t size,
                     const uint32_t new_size,
                     const uint32_t align);
```

mem	Puntero al bloque original a realojar.
size	Tamaño en bytes del bloque original `mem`.
new_size	Nuevo tamaño requerido, en bytes.
align	Alineación. Debe ser potencia de 2.

Retorna:

Puntero al bloque realojado.

Observaciones:

Utiliza *"Heap - Gestor de memoria"* (Página 192) para reservas más eficientes y seguras.

bmem_free

Libera la memoria apuntada por `mem`, previamente reservada por `bmem_malloc`, `bmem_realloc` o sus equivalentes con alineación.

```
void
bmem_free(byte_t *mem);
```

mem Puntero al bloque de memoria a liberar.

Observaciones:

Utiliza *"Heap - Gestor de memoria"* (Página 192) para reservas más eficientes y seguras.

bmem_set1

Rellena un bloque de memoria con la misma máscara de 1-byte.

```
void
bmem_set1(byte_t *dest,
          const uint32_t size,
          const byte_t mask);
```

dest Puntero al bloque de memoria.

size Tamaño en bytes del bloque `dest`.

mask Máscara.

bmem_set4

Rellena un bloque de memoria con la misma máscara de 4-bytes.

```
void
bmem_set4(byte_t *dest,
          const uint32_t size,
          const byte_t *mask);
```

```
byte_t mblock[10];
byte_t mask[4] = "abcd";
bmem_set4(mblock, 10, mask);
/* mblock = "abcdabcdab" */
```

dest Puntero al bloque de memoria.

size Tamaño en bytes del bloque `dest`. No es necesario que sea múltiplo de 4.

mask Máscara de 4 bytes.

bmem_set8

Rellena un bloque de memoria con la misma máscara de 8-bytes.

```
void
bmem_set8(byte_t *dest,
          const uint32_t size,
          const byte_t *mask);
```

dest Puntero al bloque de memoria.

size Tamaño en bytes del bloque `dest`. No es necesario que sea múltiplo de 8.

mask Máscara de 8 bytes.

bmem_set16

Rellena un bloque de memoria con la misma máscara de 16-bytes.

```
void
bmem_set16(byte_t *dest,
           const uint32_t size,
           const byte_t *mask);
```

dest Puntero al bloque de memoria.

size Tamaño en bytes del bloque `dest`. No es necesario que sea múltiplo de 16.

mask Máscara de 16 bytes.

bmem_set_u32

Rellena un array de tipo `uint32_t` con el mismo valor.

```
void
bmem_set_u32(uint32_t *dest,
             const uint32_t n,
             const uint32_t value);
```

dest Puntero al array.

n Tamaño del array (número de elementos).

value Valor de relleno.

bmem_set_r32

Rellena un array de tipo `real32_t` con el mismo valor.

```
void
bmem_set_r32(real32_t *dest,
             const uint32_t n,
             const real32_t value);
```

dest Puntero al array.

n Tamaño del array (número de elementos).

value Valor de relleno.

bmem_cmp

Compara dos bloques de memoria genéricos.

```
int
bmem_cmp(const byte_t *mem1,
         const byte_t *mem2,
         const uint32_t size);
```

mem1 Puntero al primer bloque de memoria.

mem2 Puntero al segundo bloque de memoria.

size Número de bytes a comparar.

Retorna:

Resultado de la comparación.

bmem_is_zero

Comprueba si un bloque de memoria está completamente relleno de 0s.

```
bool_t
bmem_is_zero(const byte_t *mem,
             const uint32_t size);
```

mem Puntero al bloque de memoria.

size Tamaño en bytes del bloque `mem`.

Retorna:

TRUE si todas las posiciones son 0, de lo contrario FALSE.

bmem_set_zero

Rellena un bloque de memoria con 0s.

```
void
bmem_set_zero(byte_t *dest,
              const uint32_t size);
```

dest	Puntero al bloque de memoria que debe ser rellenado.
size	Tamaño en bytes del bloque `dest`.

bmem_zero

Inicializa un objeto con 0s.

```
void
bmem_zero(type *dest,
          type);
```

```
typedef struct
{
    uint32_t f1;
    real32_t f2;
    String *f3;
    ...
} MyType;

MyType t1;
bmem_zero(&t1, MyType);
/* t1 = {0} */
```

dest	Puntero al objeto.
type	Tipo de objeto.

bmem_zero_n

Inicializa un array de objetos con 0s.

```
void
bmem_zero_n(type *dest,
            const uint32_t n,
            type);
```

dest Array de objetos.

n Tamaño del array.

type Tipo de objeto.

bmem_copy

Copia el contenido de un bloque en otro. Los bloques no deben estar superpuestos.

```
void
bmem_copy(byte_t *dest,
          const byte_t *src,
          const uint32_t size);
```

dest Puntero al bloque de destino.

src Puntero al bloque de origen.

size Número de bytes a copiar.

bmem_copy_n

Copia un array de objetos en otra localización.

```
void
bmem_copy_n(type *dest,
            const type *src,
            const uint32_t n,
            type);
```

```
real32_t v1[64];
real32_t v2[64]; = {1.f, 45.f, 12.4f, ...};
bmem_copy_n(v1, v2, 64, real32_t);
```

dest Puntero al array de destino.

src Puntero al array de origen.

n Tamaño del array (número de elementos, no bytes).

type Tipo de objeto.

bmem_move

Igual que `bmem_copy`, pero los bloques pueden estar superpuestos.

```
void
bmem_move(byte_t *dest,
          const byte_t *src,
          const uint32_t size);
```

dest Puntero al bloque de destino.

src Puntero al bloque de origen.

size Número de bytes a copiar.

Observaciones:

Si tenemos la certeza de que ambos bloque no se solapan, `bmem_copy` es mucho más eficiente.

bmem_overlaps

Comprueba si dos bloques de memoria se solapan.

```
bool_t
bmem_overlaps(byte_t *mem1,
              byte_t *mem2,
              const uint32_t size1,
              const uint32_t size2);
```

mem1 Puntero al primer bloque.

mem2 Puntero al segundo bloque.

size1 Tamaño del primer bloque (en bytes).

size2 Tamaño del segundo bloque (en bytes).

Retorna:

`TRUE` si hay solapamiento.

bmem_rev

Revierte un bloque de memoria `m[i] = m[n-i-1]`.

```
void
bmem_rev(byte_t *mem,
         const uint32_t size);
```

mem Puntero al bloque de memoria.

size Tamaño del bloque en bytes.

bmem_rev2

Revierte un bloque de memoria de 2-bytes.

```
void
bmem_rev2(byte_t *mem);
```

mem Puntero al bloque de memoria.

bmem_rev4

Revierte un bloque de memoria de 4-bytes.

```
void
bmem_rev4(byte_t *mem);
```

mem Puntero al bloque de memoria.

bmem_rev8

Revierte un bloque de memoria de 8-bytes.

```
void
bmem_rev8(byte_t *mem);
```

mem Puntero al bloque de memoria.

bmem_revcopy

Realiza una copia revertida de un bloque de memoria.

```
void
bmem_revcopy(byte_t *dest,
             const byte_t *src,
             const uint32_t size);
```

dest Puntero al bloque de destino.

src Puntero al bloque de origen.

size Número de bytes a copiar.

bmem_rev_elems

Revierte los elementos dentro de un array.

```
void
bmem_rev_elems(type*,
               const uint32_t num_elems,
               type);
```

type* Puntero al inicio del array.

num_elems Número de elementos del array.

type Tipo de objeto.

bmem_swap

Intercambia el contenido de dos bloques de memoria (no solapados). Al finalizar, `mem1[i] = mem2[i]` y `mem2[i] = mem1[i]`.

```
void
bmem_swap(byte_t *mem1,
          byte_t *mem2,
          const uint32_t size);
```

mem1 Puntero al primer bloque.

mem2 Puntero al segundo bloque.

size Número de bytes a intercambiar.

bmem_swap_type

Intercambia el contenido de dos objetos.

```
void
bmem_swap_type(type *obj1,
               type *obj2,
               type);
```

obj1 Primer objeto.

obj2 Segundo objeto.

type Tipo de objeto.

bmem_shuffle

Desordena aleatoriamente (barajar) un bloque de memoria.

```
void
bmem_shuffle(byte_t *mem,
             const uint32_t size,
             const uint32_t esize);
```

mem Puntero al bloque de memoria.

size Tamaño del bloque (número de elementos).

esize Tamaño de cada elemento.

Observaciones:

Esta función se basa en un generador de números pseudo-aleatorios. Utiliza `bmath_rand_seed` para cambiar la secuencia.

bmem_shuffle_n

Desordena aleatoriamente (barajar) un array de objetos.

```
void
bmem_shuffle_n(type *array,
               const uint32_t size,
               type);
```

array	Array de elementos.
size	Número de elementos.
type	Tipo de objeto.

Observaciones:

Esta función se basa en un generador de números pseudo-aleatorios. Utiliza `bmath_rand_seed` para cambiar la secuencia.

Capítulo 36

Librería Osbs

36.1. Tipos y Constantes

enum platform_t

Sistemas operativos soportados por NAppGUI.

ekWINDOWS	Microsoft Windows.
ekMACOS	Apple macOS.
ekLINUX	GNU/Linux.
ekIOS	Apple iOS.

enum device_t

Tipo de dispositivo.

ekDESKTOP	Ordenador de escritorio o portátil.
ekPHONE	Teléfono.
ekTABLET	Tableta.

enum win_t

Versiones de Microsoft Windows.

ekWIN_9x	Windows 95, 98 or ME.
ekWIN_NT4	Windows NT4.
ekWIN_2K	Windows 2000.
ekWIN_XP	Windows XP.

ekWIN_XP1	Windows XP Service Pack 1.
ekWIN_XP2	Windows XP Service Pack 2.
ekWIN_XP3	Windows XP Service Pack 3.
ekWIN_VI	Windows Vista.
ekWIN_VI1	Windows Vista Service Pack 1.
ekWIN_VI2	Windows Vista Service Pack 2.
ekWIN_7	Windows 7.
ekWIN_71	Windows 7 Service Pack 1.
ekWIN_8	Windows 8.
ekWIN_81	Windows 8 Service Pack 1.
ekWIN_10	Windows 10.
ekWIN_NO	El sistema no es Windows.

enum endian_t

Representa el *"Orden de bytesOrden de bytes"* (Página 211), o como los datos multi-byte son almacenados en memoria.

ekLITEND	*Little endian*. El byte más bajo primero.
ekBIGEND	*Big endian*. El byte más alto primero.

enum week_day_t

Día de la semana.

ekSUNDAY	Domingo.
ekMONDAY	Lunes.
ekTUESDAY	Martes.
ekWEDNESDAY	Miércoles.
ekTHURSDAY	Jueves.
ekFRIDAY	Viernes.
ekSATURDAY	Sábado.

enum month_t

Mes.

ekJANUARY	Enero.
ekFEBRUARY	Febrero.
ekMARCH	Marzo.
ekAPRIL	Abril.
ekMAY	Mayo.
ekJUNE	Junio.
ekJULY	Julio.
ekAUGUST	Agosto.
ekSEPTEMBER	Septiembre.
ekOCTOBER	Octubre.
ekNOVEMBER	Noviembre.
ekDECEMBER	Diciembre.

enum file_type_t

Tipo de archivo.

ekARCHIVE	Archivo ordinario.
ekDIRECTORY	Directorio.
ekOTHERFILE	Otro tipo de archivo reservado para el sistema operativo (dispositivos, tuberías, etc).

enum file_mode_t

Diferentes modos de abrir un archivo.

ekREAD	Solo lectura.
ekWRITE	Lectura y escritura.
ekAPPEND	Escritura al final del archivo.

enum file_seek_t

Posición inicial del puntero en `bfile_seek`.

ekSEEKSET	Inicio del archivo.
ekSEEKCUR	Posición actual del puntero.

`ekSEEKEND` Final del archivo.

enum ferror_t

Códigos de error manipulando archivos.

`ekFEXISTS` El archivo ya existe.

`ekFNOPATH` El directorio no existe.

`ekFNOFILE` El archivo no existe.

`ekFBIGNAME` El nombre del archivo excede la capacidad del buffer para almacenarlo.

`ekFNOFILES` No hay más archivos cuando recorremos un directorio. `bfile_dir_get`.

`ekFNOEMPTY` Se está tratando de eliminar un directorio no vacío. `hfile_dir_destroy`.

`ekFNOACCESS` No se puede acceder al archivo (posiblemente por falta de permisos).

`ekFLOCK` El archivo está siendo utilizado por otro proceso.

`ekFBIG` El archivo es muy grande. Puede aparecer en funciones que no puedan manejar archivos de más de 4Gb.

`ekFSEEKNEG` Posición negativa dentro de un archivo. Ver `bfile_seek`.

`ekFUNDEF` No hay más información acerca del error.

`ekFOK` No hay error.

enum perror_t

Códigos de error trabajando con procesos.

`ekPPIPE` Error en el canal de E/S estándar.

`ekPEXEC` Error al lanzar el proceso. Seguramente el comando es inválido.

`ekPOK` No hay error.

enum serror_t

Código de error en comunicaciones de red.

ekSNONET	No hay conexión a Internet en el dispositivo.
ekSNOHOST	No se puede conectar con el servidor remoto.
ekSTIMEOUT	Se ha excedido el tiempo máximo de espera por la conexión.
ekSSTREAM	Error en el canal E/S al leer o escribir.
ekSUNDEF	No hay más información acerca del error.
ekSOK	No hay error.

struct Date

Estructura pública que contiene los campos de una marca temporal (fecha + hora) para su acceso de forma directa.

```
struct Date
{
    int16_t year;
    uint8_t month;
    uint8_t wday;
    uint8_t mday;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
};
```

year	El año.
month	El mes (1-12). month_t.
wday	El día de la semana (0-6). week_day_t.
mday	El día del mes (1-31).
hour	La hora (0-23).
minute	El minuto (0-59).
second	El segundo (0-59).

struct Dir

Representa un directorio abierto, por el que se puede navegar. bfile_dir_open.

```
struct Dir;
```

struct File

Manejador de fichero en disco. bfile_open.

```
struct File;
```

struct Mutex

Mecanismo de exclusión mutua (**mutex**) utilizado para controlar el acceso concurrente a un recurso. *"CerrojosCerrojos"* (Página 177).

```
struct Mutex;
```

struct Proc

Representa un proceso en ejecución, con el que el programa principal puede comunicarse utilizando los canales E/S estándar. `bproc_exec`.

```
struct Proc;
```

struct DLib

Representa una librería cargada de forma dinámica en el proceso. `dlib_open`.

```
struct DLib;
```

struct Thread

Representa un hilo de ejecución, lanzado desde el proceso principal. `bthread_create`.

```
struct Thread;
```

struct Socket

Manejador de un *socket* o conexión en red. `bsocket_connect`.

```
struct Socket;
```

36.2. Funciones

FPtr_thread_main

Prototipo de function de inicio de una hebra de ejecución *(thread main)*. `bthread_create`.

```
uint32_t
(*FPtr_thread_main)(type *data);
```

data Datos pasados a la función *main* de la hebra.

Retorna:

El valor de retorno de la hebra.

osbs_start

Inicia la librería *osbs*, reservando espacio para las estructuras internas globales.

```
void
osbs_start(void);
```

osbs_finish

Finaliza la librería *osbs*, liberando el espacio de las estructuras internas globales.

```
void
osbs_finish(void);
```

osbs_platform

Obtiene el sistema operativo en el que está corriendo la aplicación.

```
platform_t
osbs_platform(void);
```

Retorna:

La plataforma.

osbs_windows

Obtiene la versión de Windows.

```
win_t
osbs_windows(void);
```

Retorna:

La versión de Microsoft Windows.

osbs_endian

Obtiene el *"Orden de bytesOrden de bytes"* (Página 211) de la plataforma.

```
endian_t
osbs_endian(void);
```

Retorna:

El orden de bytes en tipos de datos multi-byte.

bproc_exec

Lanza un nuevo proceso.

```
Proc*
bproc_exec(const char_t *command,
           perror_t *error);
```

command El comando a ejecutar (ruta y argumentos). p.e. `"ls -lh"` o `"C:\Programs\imgresize background.png -w640 -h480"`.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

Manejador del proceso hijo que podemos utilizar para comunicarnos con él. Si la función falla, retorna `NULL`.

Observaciones:

"Ejemplos multi-procesamientoEjemplos multi-procesamiento" (Página 170).

bproc_close

Cierra la comunicación con el proceso hijo y libera recursos.

```
void
bproc_close(Proc **proc);
```

proc Manejador del proceso. Será puesto a `NULL` tras el cierre.

Observaciones:

Si el proceso todavía se está ejecutando, esta función no lo finaliza. Solo cierra el canal de comunicación entre el padre y el hijo que continuarán ejecutándose de forma independiente. Como cualquier otro objeto, un proceso siempre debe cerrarse, incluso si ya ha terminado su ejecución. *"Ejemplos multi-procesamientoEjemplos multi-procesamiento"* (Página 170).

bproc_cancel

Fuerza la finalización del proceso.

```
bool_t
bproc_cancel(Proc *proc);
```

proc Manejador del proceso.

Retorna:

TRUE si el proceso ha terminado. FALSE si no.

bproc_wait

Espera hasta que el proceso hijo finalice.

```
uint32_t
bproc_wait(Proc *proc);
```

proc Manejador del proceso.

Retorna:

El valor de retorno del proceso hijo o UINT32_MAX si hay algún error.

bproc_finish

Comprueba si el proceso hijo sigue en ejecución.

```
bool_t
bproc_finish(Proc *proc,
             uint32_t *code);
```

proc Manejador del proceso.

code El valor de salida del proceso (si ha terminado). Puede ser NULL.

Retorna:

TRUE si el proceso hijo ha terminado, FALSE si no.

Observaciones:

Esta función retorna inmediatamente. No bloquea al proceso que la llama.

bproc_read

Lee datos desde la salida estándar del proceso (stdout).

```
bool_t
bproc_read(Proc *proc,
           byte_t *data,
           const uint32_t size,
           uint32_t *rsize,
           perror_t *error);
```

proc	Manejador del proceso.
data	Búfer donde se escribirán los datos leídos.
size	El número de bytes máximos a leer (tamaño del buffer).
rsize	Recibe el número de bytes leídos realmente. Puede ser NULL.
error	Código de error si la función falla. Puede ser NULL.

Retorna:

TRUE si se han leído datos. FALSE si ha ocurrido algún error.

Observaciones:

Esta función bloqueará al proceso padre hasta que el hijo escriba en su stdout. Si no hay datos en el canal y el hijo termina, retornará FALSE con rsize = 0 y error = ekPROC_SUCCESS. *"Ejemplos multi-procesamientoEjemplos multi-procesamiento"* (Página 170).

bproc_eread

Lee datos desde la salida de error del proceso (stderr).

```
bool_t
bproc_eread(Proc *proc,
            byte_t *data,
            const uint32_t size,
            uint32_t *rsize,
            perror_t *error);
```

proc	Manejador del proceso.
data	Búfer donde se escribirán los datos leídos.
size	El número de bytes máximos a leer.
rsize	Recibe el número de bytes leídos realmente. Puede ser NULL.
error	Código de error si la función falla. Puede ser NULL.

Retorna:

`TRUE` si se han leído datos. `FALSE` si ha ocurrido algún error.

Observaciones:

Esta función bloqueará al proceso padre hasta que el hijo escriba en su `stdout`. Si no hay datos en el canal y el hijo termina, retornará `FALSE` con `rsize = 0` y `error = ekPROC_SUCCESS`. *"Ejemplos multi-procesamientoEjemplos multi-procesamiento"* (Página 170).

bproc_write

Escribe datos en el canal de entrada del proceso (`stdin`).

```
bool_t
bproc_write(Proc *proc,
            const byte_t *data,
            const uint32_t size,
            uint32_t *wsize,
            perror_t *error);
```

proc	Manejador del proceso.
data	Búfer que contiene los datos a escribir.
size	El número de bytes a escribir.
wsize	Recibe el número de bytes escritos realmente. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si se han escrito datos. `FALSE` si ha ocurrido algún error.

Observaciones:

Esta función bloqueará al proceso padre si no hay espacio en el búfer para completar la escritura. Cuando el proceso hijo lea si `stdin` y libere espacio, se completará la escritura y el proceso padre continuará su ejecución. *"Ejemplos multi-procesamientoEjemplos multi-procesamiento"* (Página 170).

bproc_read_close

Cierra el canal `stdout` del proceso hijo.

```
bool_t
bproc_read_close(Proc *proc);
```

proc Manejador del proceso.

Retorna:

`TRUE` si el canal se ha cerrado. `FALSE` si ya estaba cerrado.

Observaciones:

Esta función permite ignorar la salida del proceso hijo, previniendo bloqueos por la saturación del canal. *"Lanzando procesosLanzando procesos"* (Página 169).

bproc_eread_close

Cierra el canal `stderr` del proceso hijo.

```
bool_t
bproc_eread_close(Proc *proc);
```

proc Manejador del proceso.

Retorna:

`TRUE` si el canal se ha cerrado. `FALSE` si ya estaba cerrado.

Observaciones:

Esta función permite ignorar la salida de errores del proceso hijo, previniendo bloqueos por la saturación del canal. *"Lanzando procesosLanzando procesos"* (Página 169).

bproc_write_close

Cierra el canal `stdin` del proceso hijo.

```
bool_t
bproc_write_close(Proc *proc);
```

proc Manejador del proceso.

Retorna:

`TRUE` si el canal se ha cerrado. `FALSE` si ya estaba cerrado.

Observaciones:

Algunos procesos necesitan leer todo el contenido de `stdin` antes de comenzar el trabajo. Al cerrar el canal, el proceso hijo recibe la señal `EOF` *End-Of-File* en `stdin`. *"Lanzando procesosLanzando procesos"* (Página 169).

bproc_exit

Termina el proceso actual (el que llama) y todos sus hijos de ejecución.

```
void
bproc_exit(const uint32_t code);
```

code El código de salida del proceso.

bthread_create

Crea un nuevo hilo de ejecución, que arranca en `thmain`.

```
Thread*
bthread_create(FPtr_thread_main thmain,
               type *data,
               type);
```

thmain La función de inicio de la hebra *thread_main*. Se pueden pasar datos compartidos mediante el puntero *data*.

data Datos pasados como parámetro a `thmain`.

type Tipo de `data`.

Retorna:

Manejador de la hebra. Si la función falla, retorna `NULL`.

Observaciones:

El hilo se ejecutará en paralelo hasta que `thmain` retorne o se llame a `bthread_cancel`. *"Lanzando hebrasLanzando hebras"* (Página 173).

bthread_current_id

Retorna el identificador manejador de la hebra actual, es decir, la que está corriendo cuando se llama a esta función.

```
int
bthread_current_id(void);
```

Retorna:

Manejador de la hebra.

bthread_close

Cierra el manejador de la hebra y libera recursos.

```
void
bthread_close(Thread **thread);
```

thread Manejador de la hebra. Será puesto a `NULL` tras el cierre.

Observaciones:

Si el hilo todavía se está ejecutando, esta función no lo finaliza. Como cualquier otro objeto, un hilo siempre debe cerrarse, incluso si ya ha terminado su ejecución. *"Lanzando hebrasLanzando hebras"* (Página 173).

bthread_cancel

Fuerza la terminación del hilo especificado.

```
void
bthread_cancel(Thread *thread);
```

thread Manejador de la hebra.

Observaciones:

No es recomendable llamar a esta función. No se realizará una salida "limpia" del hilo. Si se encuentra dentro de una sección crítica, esta no será liberada. Tampoco de liberará la memoria dinámica reservada de forma privada por el hilo. La forma correcta de finalizar un hilo de ejecución es retornando de `thmain`. Pueden utilizarse variables compartidas (*"Exclusión mutua"* (Página 177)) para indicarle a un hilo que debe terminar de forma limpia.

bthread_wait

Detiene al hilo que llama a esta función hasta que `thread` termina su ejecución.

```
uint32_t
bthread_wait(Thread *thread);
```

thread Manejador de la hebra a la que debemos esperar.

Retorna:

El valor de retorno del hilo. Si ocurre algún error, retorna `UINT32_MAX`.

bthread_finish

Comprueba si la hebra sigue en ejecución.

```
bool_t
bthread_finish(Thread *thread,
               uint32_t *code);
```

thread Manejador de la hebra.

code El valor de retorno de la función *thmain* (si ha terminado). Puede ser NULL.

Retorna:

TRUE si el hilo ha terminado, FALSE si no.

Observaciones:

Esta función retorna inmediatamente.

bthread_sleep

Suspende la ejecución de la hebra actual (la que llama a esta función) durante un número determinado de milisegundos.

```
void
bthread_sleep(const uint32_t milliseconds);
```

milliseconds Intervalo de tiempo (en milisegundos) que durará la suspensión.

Observaciones:

Realiza una suspensión "pasiva", donde ningún "bucle vacío" será ejecutado. El hilo es decartado por el *scheduler* y reactivado posteriormente.

bmutex_create

Crea un objeto de exclusión mutua que permite que varios hilos de ejecución compartan un mismo recurso, como una zona de memoria o archivo en disco, impidiendo que accedan al mismo tiempo.

```
Mutex*
bmutex_create(void);
```

Retorna:

El manejador de la exclusión mutua.

Observaciones:

"Hebras" (Página 172), *"Ejemplo multi-hiloEjemplo multi-hilo"* (Página 174).

bmutex_close

Cierra el objeto de exclusión mutua y libera memoria.

```
void
bmutex_close(Mutex **mutex);
```

mutex El manejador de la exclusión mutua. Será puesto a `NULL` tras el cierre.

Observaciones:

"Hebras" (Página 172), *"Ejemplo multi-hiloEjemplo multi-hilo"* (Página 174).

bmutex_lock

Marca el inicio de una sección crítica, bloqueando el acceso a un recurso compartido. Si otro hilo intenta bloquear, será detenido hasta que el hilo actual llame a `bmutex_unlock`.

```
void
bmutex_lock(Mutex *mutex);
```

mutex El manejador de la exclusión mutua.

Observaciones:

"Hebras" (Página 172), *"Ejemplo multi-hiloEjemplo multi-hilo"* (Página 174).

bmutex_unlock

Marca el final de una sección crítica, desbloqueando el acceso a un recurso compartido. Si otro hilo está esperando, se permitirá el acceso a su sección crítica y, por tanto, al recurso compartido.

```
void
bmutex_unlock(Mutex *mutex);
```

mutex El manejador de la exclusión mutua.

Observaciones:

Para evitar retrasos innecesarios, el tiempo entre `bmutex_lock` y `bmutex_unlock` debe ser lo más corto posible. Cualquier cálculo que la hebra pueda realizar en su espacio privado de memoria debe preceder a la llamada a `bmutex_lock`. *"Hebras"* (Página 172), *"Ejemplo multi-hiloEjemplo multi-hilo"* (Página 174).

dlib_open

Carga una librería dinámica en tiempo de ejecución.

```
DLib*
dlib_open(const char_t *path,
          const char_t *libname);
```

```
DLib *lib = dlib_open(NULL, "myplugin");
// myplugin.dll          In Windows
// libmyplugin.so        In Linux
// libmyplugin.dylib     In macOS
```

path Directorio donde se encuentra la librería. Puede ser `NULL`.

libname Nombre de la librería. Debe ser el nombre "plano" sin prefijos, sufijos o extensiones propias de cada sistema operativo.

Retorna:

Puntero a la librería o `NULL` si no ha podido cargarla.

Observaciones:

Si `path` es `NULL` se seguirá la estrategia de búsqueda de librerías de cada sistema operativo. Ver *"Rutas de búsqueda de librerías**Rutas de búsqueda de librerías"* (Página 179).

dlib_close

Cierra una librería previamente abierta con `dlib_open`.

```
void
dlib_close(DLib **dlib);
```

dlib Puntero a la librería. Será puesto a `NULL` tras la destrucción.

dlib_proc

Obtiene un puntero a un método de la librería.

```
type
dlib_proc(DLib *lib,
          const char_t *procname,
          type);
```

```
typedef uint32_t(*FPtr_add)(const uint32_t, const uint32_t);
FPtr_add func_add = dlib_proc(lib, "plugin_add", FPtr_add);
uint32_t ret = func_add(67, 44);
```

lib Librería.

procname Nombre del método.

type Tipo del método. Necesario para hacer la conversión desde un puntero genérico.

Retorna:

Puntero al método.

dlib_var

Obtiene un puntero a una variable de la librería.

```
type*
dlib_var(DLib *lib,
         const char_t *varname,
         type);
```

```
const V2Df *vzero = dlib_var(lib, "kV2D_ZEROf", V2Df);
```

lib Librería.

varname Nombre de la variable.

type Tipo de la variable.

Retorna:

Puntero a la variable.

bfile_dir_work

Obtiene el directorio de trabajo actual del proceso *(working dir)*. Es el directorio a partir del cual los *pathnames* relativos serán interpretados.

```
uint32_t
bfile_dir_work(char_t *pathname,
               const uint32_t size);
```

pathname Búfer donde se escribirá el directorio.

size Tamaño en bytes del búfer `pathname`.

Retorna:

El número de bytes escritos en `pathname`, incluyendo el carácter nulo `'\0'`.

Observaciones:

"Filename y pathnameFilename y pathname" (Página 181)

bfile_dir_set_work

Cambia el directorio actual de la aplicación *(working dir)*. Los *pathname* relativos serán interpretados a partir de aquí.

```
bool_t
bfile_dir_set_work(const char_t *pathname,
                   ferror_t *error);
```

pathname El nombre del directorio.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el directorio de trabajo ha cambiado, `FALSE` si ha habido algún error.

Observaciones:

"Filename y pathnameFilename y pathname" (Página 181)

bfile_dir_home

Obtiene el directorio `home` del usuario actual.

```
uint32_t
bfile_dir_home(char_t *pathname,
               const uint32_t size);
```

pathname Búfer donde se escribirá el directorio.

size Tamaño en bytes del búfer `pathname`.

Retorna:

El número de bytes escritos en `pathname`, incluyendo el carácter nulo `'\0'`.

Observaciones:

"Home y AppDataHome y AppData" (Página 181)

bfile_dir_data

Obtiene el directorio *AppData* donde pueden guardarse datos de configuración de la aplicación.

```
uint32_t
bfile_dir_data(char_t *pathname,
               const uint32_t size);
```

pathname Búfer donde se escribirá el directorio.

size Tamaño en bytes del búfer `pathname`.

Retorna:

El número de bytes escritos en `pathname`, incluyendo el carácter nulo `'\0'`.

Observaciones:

"Home y AppDataHome y AppData" (Página 181)

bfile_dir_exec

Obtiene el *pathname* absoluto del ejecutable actual.

```
uint32_t
bfile_dir_exec(char_t *pathname,
               const uint32_t size);
```

```
char_t path[512];
bfile_dir_exec(path, 512);
path = "C:\Program Files\TheApp\theapp.exe"
```

pathname Búfer donde se escribirá el directorio.

size Tamaño en bytes del búfer `pathname`.

Retorna:

El número de bytes escritos en `pathname`, incluyendo el carácter nulo `'\0'`.

bfile_dir_create

Crea un nuevo directorio. Fallará si algún directorio intermedio de `pathname` no existe.

```
bool_t
bfile_dir_create(const char_t *pathname,
                 ferror_t *error);
```

pathname Nombre del directorio a crear, terminado en carácter nulo `'\0'`.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el directorio ha sido creado, `FALSE` si ha habido algún error.

Observaciones:

`hfile_dir_create` crea todos los directorios intermedios de una vez.

bfile_dir_open

Abre un directorio para navegar por su contenido. Después hay que utilizar `bfile_dir_get` para iterar. No se ordenan los *filename* bajo ningún criterio. Al finalizar, debes llamar a `bfile_dir_close`.

```
Dir*
bfile_dir_open(const char_t *pathname,
               ferror_t *error);
```

pathname Nombre del directorio, terminado en carácter nulo `'\0'`.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

El manejador del directorio o `NULL` si se ha producido algún error.

bfile_dir_close

Cierra un directorio previamente abierto con `bfile_dir_open`.

```
void
bfile_dir_close(Dir **dir);
```

dir El manejador del directorio. Será puesto a `NULL` tras el cierre.

bfile_dir_get

Obtiene los atributos del archivo actual cuando recorremos un directorio. Previamente hemos de abrir el directorio con `bfile_dir_open`.

```
bool_t
bfile_dir_get(Dir *dir,
              char_t *filename,
              const uint32_t size,
              file_type_t *type,
              uint64_t *fsize,
              Date *updated,
              ferror_t *error);
```

dir	Manejador del directorio abierto.
filename	Aquí se escribirá el nombre del archivo o sub-directorio, terminado en carácter nulo `'\0'` y sin incluir ninguna ruta. Puede ser `NULL`.
size	Tamaño en bytes del búfer `name`.
type	Obtiene el tipo de archivo. Puede ser `NULL`.
fsize	Obtiene el tamaño del archivo en bytes. Puede ser `NULL`.
updated	Obtiene la fecha de la última actualización del archivo. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si los atributos del archivo han sido leidos correctamente. Cuando ya no quedan archivos por recorrer, retorna false `FALSE` con `error=ekFNOFILES`.

Observaciones:

Esta función avanzará al siguiente archivo dentro del directorio abierto después de obtener los datos del elemento actual. Si no hay suficiente espacio en `name`, retornará `FALSE` con `error=ekFBIGNAME` y no avanzará al siguiente fichero. Utiliza `hfile_dir_loop` para navegar por el contenido de un directorio más cómodamente.

bfile_dir_delete

Elimina un directorio. Fallará si el directorio no está completamente vacío. Utiliza `hfile_dir_destroy` para borrar completamente y de forma recursiva un directorio que pueda tener contenido.

```
bool_t
bfile_dir_delete(const char_t *pathname,
                 ferror_t *error);
```

pathname	Nombre del directorio, terminado en carácter nulo `'\0'`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el directorio ha sido eliminado, `FALSE` si no.

bfile_create

Crea un nuevo archivo. Si previamente ya existía su contenido será borrado. El nuevo archivo será abierto para escritura.

```
File*
bfile_create(const char_t *pathname,
             ferror_t *error);
```

pathname Nombre del archivo incluida su ruta absoluta o relativa.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

El manejador del archivo o `NULL` si se ha producido algún error.

bfile_open

Abre un archivo existente. No crea el archivo, si no existe la función fallará.

```
File*
bfile_open(const char_t *pathname,
           const file_mode_t mode,
           ferror_t *error);
```

pathname Nombre del archivo incluida su ruta absoluta o relativa.

mode Modo de apertura.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

El manejador del archivo o `NULL` si se ha producido algún error.

bfile_close

Cierra un archivo previamente abierto con `bfile_create` o `bfile_open`.

```
void
bfile_close(File **file);
```

file Manejador del archivo. Será puesto a `NULL` tras el cierre.

bfile_lstat

Obtiene los atributos de un archivo a través de su *pathname*.

```
bool_t
bfile_lstat(const char_t *pathname,
            file_type_t *type,
            uint64_t *fsize,
            Date *updated,
```

```
            ferror_t *error);
```

pathname	Nombre del archivo incluida su ruta absoluta o relativa.
type	Obtiene el tipo de archivo. Puede ser `NULL`.
fsize	Obtiene el tamaño del archivo en bytes. Puede ser `NULL`.
updated	Obtiene la fecha de la última actualización del archivo. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si ha funcionado correctamente, o `FALSE` si no.

bfile_fstat

Obtiene los atributos de un archivo a través de su manejador.

```
bool_t
bfile_fstat(File *file,
            file_type_t *type,
            uint64_t *fsize,
            Date *updated,
            ferror_t *error);
```

file	Manejador del archivo.
type	Obtiene el tipo de archivo. Puede ser `NULL`.
fsize	Obtiene el tamaño del archivo en bytes. Puede ser `NULL`.
updated	Obtiene la fecha de la última actualización del archivo. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si ha funcionado correctamente, o `FALSE` si no.

bfile_read

Lee datos desde un archivo abierto.

```
bool_t
bfile_read(File *file,
           byte_t *data,
           const uint32_t size,
           uint32_t *rsize,
           ferror_t *error);
```

file	Manejador del archivo.
data	Búfer donde se escribirán los datos leídos.
size	El número de bytes máximos a leer.
rsize	Recibe el número de bytes leídos realmente. Puede ser NULL.
error	Código de error si la función falla. Puede ser NULL.

Retorna:

TRUE si los datos se han leído correctamente. Si no hay más datos (final del fichero) retorna FALSE con `rsize = 0` y `error=ekFOK`.

Observaciones:

"File streamFile stream" (Página 198) implementa funciones de alto nivel para lectura/escritura en archivos.

bfile_write

Escribe datos en un archivo abierto.

```
bool_t
bfile_write(File *file,
            const byte_t *data,
            const uint32_t size,
            uint32_t *wsize,
            ferror_t *error);
```

file	Manejador del archivo.
data	Búfer que contiene los datos a escribir.
size	El número de bytes a escribir.
wsize	Recibe el número de bytes escritos realmente. Puede ser NULL.
error	Código de error si la función falla. Puede ser NULL.

Retorna:

TRUE si los datos se han escrito, o FALSE si ha habido algún error.

Observaciones:

"File streamFile stream" (Página 198) implementa funciones de alto nivel para lectura/escritura en archivos.

bfile_seek

Mueve el puntero de un archivo a una nueva ubicación.

```
bool_t
bfile_seek(File *file,
           const int64_t offset,
           const file_seek_t whence,
           ferror_t *error);
```

file	Manejador del archivo.
offset	Número de bytes a desplazar el puntero. Puede ser negativo.
whence	Posición del puntero a partir de la cual se sumará `offset`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si ha funcionado correctamente, o `FALSE` si no.

Observaciones:

Retornará `FALSE` y `error` `ekFSEEKNEG` si la posición final del puntero es negativa. No es un error establecer un puntero de en una posición más allá del final del archivo. El tamaño del archivo no aumenta hasta que se escriba en él. Una operación de escritura aumenta el tamaño del archivo a la posición del puntero más el tamaño del búfer escrito. Los bytes intermedios quedarían indeterminados.

bfile_pos

Retorna la posición actual del puntero de archivo.

```
uint64_t
bfile_pos(const File *file);
```

file	Manejador del archivo.

Retorna:

Posición a partir de inicio del archivo.

bfile_delete

Elimina un fichero del sistema de archivos.

```
bool_t
bfile_delete(const char_t *pathname,
             ferror_t *error);
```

pathname Nombre del archivo incluida su ruta absoluta o relativa.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el archivo ha sido eliminado, o `FALSE` si ha ocurrido algún error.

bsocket_connect

Crea un socket cliente e intenta establecer una conexión con un servidor remoto.

```
Socket*
bsocket_connect(const uint32_t ip,
                const uint16_t port,
                const uint32_t timeout_ms,
                serror_t *error);
```

ip La dirección IPv4 32-bit del host remoto. `bsocket_str_ip`.

port El puerto de conexión.

timeout_ms Número máximo de milisegundos que esperará para establecer conexión. Si es `0` se esperará indefinidamente.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

Manejador del socket, o `NULL` si la función falla.

Observaciones:

El proceso se bloqueará hasta que se obtenga respuesta desde el servidor o se cumpla el `timeout`. Ver *"Ejemplo Cliente/ServidorEjemplo Cliente/Servidor"* (Página 183).

bsocket_server

Crea un socket servidor.

```
Socket*
bsocket_server(const uint16_t port,
               const uint32_t max_connect,
               serror_t *error);
```

port El puerto donde "escuchará" el servidor.

max_connect El número máximo de conexiones que puede mantener en cola.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

Manejador del socket, o `NULL` si la función falla.

Observaciones:

Las peticiones de los clientes se irán almacenando en una cola hasta que se reciba una llamada a `bsocket_accept`. Ver *"Ejemplo Cliente/ServidorEjemplo Cliente/Servidor"* (Página 183).

bsocket_accept

Acepta una conexión al servidor creado con `bsocket_server` e inicia la conversación con el cliente.

```
Socket*
bsocket_accept(Socket *socket,
               const uint32_t timeout_ms,
               serror_t *error);
```

socket	Manejador devuelto por `bsocket_server`.
timeout_ms	Número máximo de milisegundos que esperará para recibir la petición. Si es `0` se esperará indefinidamente.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

Manejador del socket, o `NULL` si la función falla.

Observaciones:

El proceso se bloqueará hasta que se obtenga una petición por parte de un cliente o se cumpla el `timeout`. Ver *"Ejemplo Cliente/ServidorEjemplo Cliente/Servidor"* (Página 183).

bsocket_close

Cierra un socket previamente creado con `bsocket_connect`, `bsocket_server` o `bsocket_accept` .

```
void
bsocket_close(Socket **socket);
```

socket	El manejador del socket. Será puesto a `NULL` tras el cierre.

bsocket_local_ip

Obtiene la dirección ip y el puerto local asociado al socket.

```
void
bsocket_local_ip(Socket *socket,
                 uint32_t *ip,
                 uint16_t *port);
```

socket	Manejador del socket.
ip	Dirección IP local.
port	Puerto IP local.

bsocket_remote_ip

Obtiene la dirección ip y el puerto remoto asociado al otro interlocutor de la conexión.

```
void
bsocket_remote_ip(Socket *socket,
                  uint32_t *ip,
                  uint16_t *port);
```

socket	Manejador del socket.
ip	Dirección IP remota.
port	Puerto IP remoto.

bsocket_read_timeout

Establece el tiempo máximo de espera de la función `bsocket_read`.

```
void
bsocket_read_timeout(Socket *socket,
                     const uint32_t timeout_ms);
```

socket	Manejador del socket.
timeout_ms	Número máximo de milisegundos que esperará hasta que el interlocutor escriba datos en el canal. Si es `0` se esperará indefinidamente.

bsocket_write_timeout

Establece el tiempo máximo de espera de la función `bsocket_write`.

```
void
bsocket_write_timeout(Socket *socket,
                      const uint32_t timeout_ms);
```

socket	Manejador del socket.
timeout_ms	Número máximo de milisegundos que esperará hasta que el interlocutor lea los datos y desbloqueé en el canal. Si es `0` se esperará indefinidamente.

bsocket_read

Lee datos desde el socket.

```
bool_t
bsocket_read(Socket *socket,
             byte_t *data,
             const uint32_t size,
             uint32_t *rsize,
             serror_t *error);
```

socket	Manejador del socket.
data	Búfer donde se escribirán los datos leídos.
size	El número de bytes máximos a leer (tamaño del búfer).
rsize	Recibe el número de bytes leídos realmente. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si se han leído datos. `FALSE` si ha ocurrido algún error.

Observaciones:

El proceso se bloqueará hasta que el interlocutor escriba datos en el canal o venza el timeout. Ver `bsocket_read_timeout`.

bsocket_write

Escribe datos en el socket.

```
bool_t
bsocket_write(Socket *socket,
              const byte_t *data,
              const uint32_t size,
              uint32_t *wsize,
              serror_t *error);
```

socket	Manejador del socket.
data	Búfer que contiene los datos a escribir.
size	El número de bytes a escribir.
wsize	Recibe el número de bytes escritos realmente. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si se han escrito datos. `FALSE` si ha ocurrido algún error.

Observaciones:

El proceso se bloqueará si el canal está lleno hasta que el interlocutor lea los datos y desbloqueé o venza el timeout. Ver `bsocket_write_timeout`.

bsocket_url_ip

Obtiene la dirección IPv4 de un host a partir de su url.

```
uint32_t
bsocket_url_ip(const char_t *url,
               serror_t *error);
```

```
uint32_t ip = bsocket_url_ip("www.google.com", NULL);
if (ip != 0)
{
    Socket *sock = bsocket_connect(ip, 80, NULL);
    ...
}
```

url	La url del host p.e, `www.google.com`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

Valor de la dirección IPv4 del host o `0` si ha habido algún error.

bsocket_str_ip

Obtiene la dirección IPv4 a partir de una cadena tipo `"192.168.1.1"`.

```
uint32_t
bsocket_str_ip(const char_t *ip);
```

```
uint32_t ip = bsocket_str_ip("192.168.1.1");
Socket *sock = bsocket_connect(ip, 80, NULL);
    ...
}
```

ip La cadena con la IP.

Retorna:

Valor de la dirección IPv4 en formato binario 32bits.

bsocket_host_name

Obtiene el nombre del host.

```
const char_t*
bsocket_host_name(char_t *buffer,
                  const uint32_t size);
```

buffer Buffer para almacenar el nombre.

size Tamaño de `buffer`.

Retorna:

Puntero a la cadena `buffer`.

bsocket_host_name_ip

Obtiene el nombre del host a través de su IP.

```
const char_t*
bsocket_host_name_ip(uint32_t ip,
                     char_t *buffer,
                     const uint32_t size);
```

ip Valor de la dirección IPv4 en formato binario 32bits.

buffer Buffer para almacenar el nombre.

size Tamaño de `buffer`.

Retorna:

Puntero a la cadena `buffer`.

bsocket_ip_str

Obtiene la dirección IP en formato cadena de texto.

```
const char_t*
bsocket_ip_str(uint32_t ip,
               const char_t *ip);
```

ip Valor de la dirección IPv4 en formato binario 32bits.

ip La cadena con la IP.

Retorna:

Cadena tipo “192.168.1.1”.

Observaciones:

La cadena se devuelve en un búfer interno que será sobrescrito en la siguiente llamada. Hacer una copia de la cadena si necesitamos que sea persistente.

bsocket_hton2

Cambia el “endianness” de un valor de 16bits previamente a ser enviado por el socket *Host-to-Network*.

```
void
bsocket_hton2(byte_t *dest,
              const byte_t *src);
```

```
uint16_t value = 45321;
byte_t dest[2];
bsocket_hton2(dest, (const byte_t*)&value);
bsocket_write(sock, dest, 2, NULL, NULL);
```

dest Búfer destino (al menos 2 bytes).

src Búfer (variable).

bsocket_hton4

Igual que `bsocket_hton2`, para valores de 4 bytes.

```
void
bsocket_hton4(byte_t *dest,
              const byte_t *src);
```

dest Búfer destino (al menos 4 bytes).

src Búfer (variable).

bsocket_hton8

Igual que `bsocket_hton2`, para valores de 8 bytes.

```
void
bsocket_hton8(byte_t *dest,
              const byte_t *src);
```

dest Búfer destino (al menos 8 bytes).

src Búfer (variable).

bsocket_ntoh2

Cambia el "endianness" de un valor de 16bits tras ser recibido por el socket *Network-to-Host*.

```
void
bsocket_ntoh2(byte_t *dest,
              const byte_t *src);
```

```
byte_t src[2];
uint16_t value;
bsocket_read(sock, src, 2, NULL, NULL);
bsocket_ntoh2((byte_t*)&value, src);
// value = 45321
```

dest Búfer (variable) destino de 16bits.

src Búfer recibido por el socket.

bsocket_ntoh4

Igual que `bsocket_ntoh2`, para valores de 4 bytes.

```
void
bsocket_ntoh4(byte_t *dest,
              const byte_t *src);
```

dest Búfer (variable) destino de 32bits.

src Búfer recibido por el socket.

bsocket_ntoh8

Igual que `bsocket_ntoh2`, para valores de 8 bytes.

```
void
bsocket_ntoh8(byte_t *dest,
              const byte_t *src);
```

dest Búfer (variable) destino de 64bits.

src Búfer recibido por el socket.

btime_now

Obtiene el número de micro-segundos trascurridos desde el 1 de Enero de 1970 UTC (Unix Time) hasta este preciso momento. Utiliza la diferencia entre instantes para saber el tiempo consumido por un proceso.

```
uint64_t
btime_now(void);
```

Retorna:

El número de micro-segundos transcurridos, es decir, el número de intervalos de 1/1000000 segundos.

Observaciones:

El instante inicial es el 1 de Enero de 1970 en sistemas Unix/Linux y el 1 de Enero de 1601 en Windows ya que es el primer año del ciclo Gregoriano en el que fue activado Windows NT. Esta función equipara ambos inicios, devolviendo siempre el tiempo Unix.

btime_date

Obtiene la fecha actual del sistema.

```
void
btime_date(Date *date);
```

date La fecha actual.

btime_to_micro

Convierte una fecha en Tiempo Unix.

```
uint64_t
btime_to_micro(const Date *date);
```

date La fecha a convertir.

Retorna:

El número de micro-segundos desde 1 de Enero de 1970 UTC.

btime_to_date

Transforma el Tiempo Unix en una fecha.

```
void
btime_to_date(const uint64_t micro,
              Date *date);
```

micro Número de micro-segundos desde el 1 de Enero de 1970 UTC.

date Fecha resultado.

log_printf

Escribe un mensaje en el *log*, con el formato del `printf`.

```
uint32_t
log_printf(const char_t *format,
           ...);
```

```
log_printf("Leaks of object '%s' (%d bytes)", object->name, object->size);
[12:34:23] Leaks of object 'String' (96 bytes)
```

format Cadena con el formato tipo-printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El número de bytes escritos.

log_output

Establece si el contenido del *log* será redirigido o no a la salida estándar.

```
void
log_output(const bool_t std,
           const bool_t err);
```

std Si `TRUE` las líneas se enviarán a la salida estándar `stdout`. Por defecto, `TRUE`.

err Si `TRUE` las líneas se enviarán a la salida de error `stderr`. Por defecto, `FALSE`.

log_file

Establece un archivo de destino, donde se escribirán las lineas del *log*.

```
void
log_file(const char_t *pathname);
```

pathname Nombre del archivo incluida su ruta absoluta o relativa. Si el archivo no existe será creado y si ya existe, las futuras líneas se añadirán el final del mismo. Si `NULL` se deshabilitará la escritura en archivo del *log*.

log_get_file

Obtiene el archivo actual asociado al *log*.

```
const char_t*
log_get_file(void);
```

Retorna:

El *pathname* absoluto del archivo.

Capítulo 37

Librería Core

37.1. Tipos y Constantes

DeclSt

Dado un `struct`, habilita macros para la comprobación de tipos en tiempo de compilación en *"Arrays"* (Página 212) y *"Árboles binarios de búsqueda"* (Página 222). Uso: `DeclSt(Product)` inmediatamente después de la definición del `struct`. Ver *"Registros o punterosRegistros o punteros"* (Página 214).

DeclPt

Igual que `DeclSt` para contenedores de punteros.

kSTDIN

Stream conectado a la entrada estándar `stdin`.

```
Stream* kSTDIN;
```

kSTDOUT

Stream conectado a la salida estándar `stdout`.

```
Stream* kSTDOUT;
```

kSTDERR

Stream conectado a la salida de errores `stderr`.

```
Stream* kSTDERR;
```

kDEVNULL

Stream de escritura nulo. Será ignorado todo el contenido que se mande por este canal.

```
Stream* kDEVNULL;
```

kDATE_NULL

Representa una fecha inválida.

```
Date kDATE_NULL;
```

enum core_event_t

Tipos de evento en la librería *core*.

ekEASSERT	Redirección de los *"Asserts"* (Página 155).
ekEFILE	Un archivo detectado mientras recorremos un directorio. `hfile_dir_loop`.
ekEENTRY	Entrada en un sub-directorio mientras recorremos un directorio. `hfile_dir_loop`.
ekEEXIT	Salida de un sub-directorio.

enum sstate_t

Estado en *"Streams"* (Página 197).

ekSTOK	Todo bien, no errores.
ekSTEND	No hay más datos en el canal.
ekSTCORRUPT	Los datos en el canal no son válidos o no se han leído correctamente.
ekSTBROKEN	Error en el canal de comunicación.

enum vkey_t

Códigos de teclado. Ver *"Uso del tecladoUso del teclado"* (Página 322).

ekKEY_UNDEF

ekKEY_A

```
      ekKEY_S
      ekKEY_D
      ekKEY_F
      ekKEY_H
      ekKEY_G
      ekKEY_Z
      ekKEY_X
      ekKEY_C
      ekKEY_V
ekKEY_BSLASH
      ekKEY_B
      ekKEY_Q
      ekKEY_W
      ekKEY_E
      ekKEY_R
      ekKEY_Y
      ekKEY_T
      ekKEY_1
      ekKEY_2
      ekKEY_3
      ekKEY_4
      ekKEY_6
      ekKEY_5
      ekKEY_9
      ekKEY_7
      ekKEY_8
      ekKEY_0
ekKEY_RCURLY
```

```
          ekKEY_O
          ekKEY_U
      ekKEY_LCURLY
          ekKEY_I
          ekKEY_P
      ekKEY_RETURN
          ekKEY_L
          ekKEY_J
   ekKEY_SEMICOLON
          ekKEY_K
       ekKEY_QUEST
       ekKEY_COMMA
       ekKEY_MINUS
          ekKEY_N
          ekKEY_M
      ekKEY_PERIOD
         ekKEY_TAB
       ekKEY_SPACE
        ekKEY_GTLT
        ekKEY_BACK
      ekKEY_ESCAPE
         ekKEY_F17
  ekKEY_NUMDECIMAL
     ekKEY_NUMMULT
      ekKEY_NUMADD
     ekKEY_NUMLOCK
      ekKEY_NUMDIV
      ekKEY_NUMRET
```

```
ekKEY_NUMMINUS
     ekKEY_F18
     ekKEY_F19
ekKEY_NUMEQUAL
    ekKEY_NUM0
    ekKEY_NUM1
    ekKEY_NUM2
    ekKEY_NUM3
    ekKEY_NUM4
    ekKEY_NUM5
    ekKEY_NUM6
    ekKEY_NUM7
    ekKEY_NUM8
    ekKEY_NUM9
      ekKEY_F5
      ekKEY_F6
      ekKEY_F7
      ekKEY_F3
      ekKEY_F8
      ekKEY_F9
     ekKEY_F11
     ekKEY_F13
     ekKEY_F16
     ekKEY_F14
     ekKEY_F10
     ekKEY_F12
     ekKEY_F15
  ekKEY_PAGEUP
```

```
ekKEY_HOME
ekKEY_SUPR
ekKEY_F4
ekKEY_PAGEDOWN
ekKEY_F2
ekKEY_END
ekKEY_F1
ekKEY_LEFT
ekKEY_RIGHT
ekKEY_DOWN
ekKEY_UP
ekKEY_LSHIFT
ekKEY_RSHIFT
ekKEY_LCTRL
ekKEY_RCTRL
ekKEY_LALT
ekKEY_RALT
ekKEY_INSERT
ekKEY_EXCLAM
ekKEY_MENU
ekKEY_LWIN
ekKEY_RWIN
ekKEY_CAPS
ekKEY_TILDE
ekKEY_GRAVE
ekKEY_PLUS
```

enum mkey_t

Teclas modificadoras.

ekMKEY_NONE

ekMKEY_SHIFT

ekMKEY_CONTROL

ekMKEY_ALT

ekMKEY_COMMAND

enum token_t

Tipos de *tokens* en `stm_read_token`.

ekTSLCOM	Comentario de una sola línea, que comienza por `//`.
ekTMLCOM	Comentario multi-línea, encerrado entre `/*` y `*/`.
ekTSPACE	Representa una serie de espacios en blanco (`' '`, `'\t'`, `'\v'`, `'\f'`, `'\r'`).
ekTEOL	Representa el carácter nueva línea (`'\n'`).
ekTLESS	Signo menor que `'<'`.
ekTGREAT	Signo mayor que `'>'`.
ekTCOMMA	Signo coma `','`.
ekTPERIOD	Signo punto `'.'`.
ekTSCOLON	Signo punto y coma `';'`.
ekTCOLON	Signo dos puntos `':'`.
ekTOPENPAR	Paréntesis de apertura `'('`.
ekTCLOSPAR	Paréntesis de cierre `')'`.
ekTOPENBRAC	Corchete de apertura `'['`.
ekTCLOSBRAC	Corchete de cierre `']'`.
ekTOPENCURL	Llave de apertura `'{'`.
ekTCLOSCURL	Llave de cierre `'}'`.
ekTPLUS	Signo más `'+'`.
ekTMINUS	Signo menos `'-'`.
ekTASTERK	Signo asterisco `'*'`.
ekTEQUALS	Signo igual `'='`.

ekTDOLLAR	Signo dolar.
ekTPERCEN	Signo porcentaje `'%'`.
ekTPOUND	Signo almohadilla `'#'`.
ekTAMPER	Signo *ampersand* `'&'`.
ekTAPOST	Signo apóstrofe `'''`.
ekTQUOTE	Signo comillas `'"'`.
ekTCIRCUM	Signo acento circunflejo `'^'`.
ekTTILDE	Signo tilde `'~'`.
ekTEXCLA	Signo exclamación `'!'`.
ekTQUEST	Signo interrogación `'?'`.
ekTVLINE	Signo barra vertical `'|'`.
ekTSLASH	Signo barra oblicua `'/'`.
ekTBSLASH	Signo barra invertida `'\'`.
ekTAT	Signo arroba `'@'`.
ekTINTEGER	Número entero. *"NúmerosNúmeros"* (Página 206).
ekTOCTAL	Número octal. *"NúmerosNúmeros"* (Página 206).
ekTHEX	Número hexadecimal. *"NúmerosNúmeros"* (Página 206).
ekTREAL	Número real. *"NúmerosNúmeros"* (Página 206).
ekTSTRING	Cadena de caracteres Unicode, entre comillas. *"CadenasCadenas"* (Página 206).
ekTIDENT	Identificador. *"IdentificadoresIdentificadores"* (Página 205).
ekTUNDEF	Token desconocido.
ekTCORRUP	Error en el *"Streams"* (Página 197) de entrada.
ekTEOF	Final del *"Streams"* (Página 197). No hay más tokens.
ekTRESERVED	Palabras clave. Al ser de propósito general, el analizador no etiqueta ningún identificador como palabra reservada. Hay que hacerlo en fases posteriores al análisis.

struct Buffer

Bloque de memoria de propósito general, reservado dinámicamente. Una vez creado, ya no se puede redimensionar. *"Buffers"* (Página 196).

```
struct Buffer;
```

struct String

Cadena de caracteres UTF8 reservada dinámicamente. Son objetos "parcialmente mutables". La memoria reservada no puede crecer, pero se pueden sustituir caracteres siempre que no se desborde la capacidad inicial del buffer. *"Strings"* (Página 196).

```
struct String;
```

struct Stream

Canal genérico de entrada/salida, donde es posible leer y escribir datos con formato. *"Streams"* (Página 197).

```
struct Stream;
```

struct ArrSt

Array de registros. El tipo de objeto se indica entre paréntesis. *"Arrays"* (Página 212).

```
struct ArrSt;
```

struct ArrPt

Array de punteros. El tipo de objeto se indica entre paréntesis. *"Arrays (punteros)"* (Página 222).

```
struct ArrPt;
```

struct SetSt

Conjunto de registros. El tipo de objeto se indica entre paréntesis. *"Árboles binarios de búsqueda"* (Página 222).

```
struct SetSt;
```

struct SetPt

Conjunto de punteros. El tipo de objeto se indica entre paréntesis. *"Árboles binarios de búsqueda (punteros)"* (Página 227).

```
struct SetPt;
```

struct RegEx

Expresión regular. *"Expresiones regulares"* (Página 227).

```
struct RegEx;
```

struct Event

Contiene información referente a un evento. *"Eventos"* (Página 235).

```
struct Event;
```

struct KeyBuf

Búfer de teclado con el estado de cada tecla (pulsada/liberada). *"Búfer de teclado"* (Página 236).

```
struct KeyBuf;
```

struct Listener

Enlaza al generador y receptor de un eventos a través de una función *callback*. *"Eventos"* (Página 235).

```
struct Listener;
```

struct IListener

Interfaz C++ para utilizar miembros de clase como manejadores de eventos. *"Uso de C++"* (Página 45).

```
struct IListener;
```

struct DirEntry

Elemento de un directorio, obtenido mediante `hfile_dir_list`.

```
struct DirEntry
{
    String* name;
    file_type_t type;
    uint64_t size;
    Date date;
};
```

`name`	Nombre del archivo o subdirectorio, sin directorios intermedios.
`type`	Tipo de elemento.
`size`	Tamaño en bytes.
`date`	Fecha de la última modificación.

struct EvFileDir

Parámetros del evento `ekEFILE` y `ekEENTRY` durante la navegación automática por directorios. `hfile_dir_loop`.

```
struct EvFileDir
{
    const char_t* pathname;
    uint32_t level;
};
```

`pathname`	El camino parcial desde el parámetro `pathname` de `hfile_dir_loop`.
`level`	La profundidad del directorio desde `pathname`.

struct ResPack

Paquete de recursos que serán cargados conjuntamente. Utilizar `ResId` para acceder a un recurso concreto. *"Recursos"* (Página 131).

```
struct ResPack;
```

struct ResId

Identificador de un recurso. Son generados automáticamente por *nrc NAppGUI Resource Compiler*. *"Recursos"* (Página 131).

```
struct ResId;
```

struct Clock

Mide el tiempo transcurrido entre dos instantes dentro de la aplicación, con precisión de micro-segundos. También es útil para lanzar eventos a intervalos regulares de tiempo.

```
struct Clock;
```

37.2. Funciones

FPtr_remove

Libera la memoria de los campos de un objeto, pero no el objeto en sí mismo. *"Registros o punterosRegistros o punteros"* (Página 214).

```
void
(*FPtr_remove)(type *obj);
```

obj Puntero al objeto cuyos campos deben ser liberados.

FPtr_event_handler

Manejador de evento. Son funciones `callback` que serán llamadas por el generador de un evento cuando este ocurra. *"Eventos"* (Página 235).

```
void
(*FPtr_event_handler)(type *obj,
                      Event *event);
```

obj Datos generales pasados como primer parámetro de la función.

event El evento.

FPtr_read

Crea un objeto a partir de datos leídos desde un *"Streams"* (Página 197). *"Serialización-Serialización"* (Página 218).

```
type*
(*FPtr_read)(Stream *stream);
```

stream El canal E/S donde está serializado el objeto.

Retorna:

El objeto creado, deserializando los datos del stream.

FPtr_read_init

Similar a `FPtr_read` donde la memoria del objeto ya ha sido reservada, pero no inicializada. *"SerializaciónSerialización"* (Página 218).

```
void
(*FPtr_read_init)(Stream *stream,
                  type *obj);
```

stream El canal E/S donde está serializado el objeto.

obj El objeto cuyos campos hay que deserializar.

FPtr_write

Escribe un objeto en un *"Streams"* (Página 197). *"SerializaciónSerialización"* (Página 218).

```
void
(*FPtr_write)(Stream *stream,
              const type *obj);
```

stream El canal E/S donde serializar el objeto.

obj El objeto a escribir.

core_start

Inicia la librería *core*, reservando espacio para las estructuras internas globales. Internamente llama a `osbs_start`.

```
void
core_start(void);
```

core_finish

Finaliza la librería *core*, liberando el espacio de las estructuras internas globales. Internamente llama a `osbs_finish`.

```
void
core_finish(void);
```

heap_start_mt

Inicia una sección multi-hilo.

```
void
heap_start_mt(void);
```

Observaciones:

Ver *"Memoria múlti-hiloMemoria múlti-hilo"* (Página 193).

heap_end_mt

Finaliza una sección multi-hilo.

```
void
heap_end_mt(void);
```

Observaciones:

Ver *"Memoria múlti-hiloMemoria múlti-hilo"* (Página 193).

heap_verbose

Activa/desactiva del modo 'verbose' del auditor de memoria.

```
void
heap_verbose(bool_t verbose);
```

verbose `TRUE` para activar.

Observaciones:

Por defecto `FALSE`.

heap_stats

Activa/desactiva las estadísticas del auditor de memoria.

```
void
heap_stats(bool_t stats);
```

stats `TRUE` para activar.

Observaciones:

Por defecto `TRUE`.

heap_leaks

Retorna `TRUE` si existen fugas de memoria al acabar la ejecución.

```
bool_t
heap_leaks(void);
```

Retorna:

`TRUE` si existen fugas.

heap_malloc

Reserva un bloque de memoria con la alineación por defecto `sizeof(void*)`.

```
byte_t*
heap_malloc(const uint32_t size,
            const char_t *name);
```

```
byte_t *mem = heap_malloc(1024 * 768, "PixelBuffer");
...
heap_free(&mem, 1024 * 768, "PixelBuffer");
```

size Tamaño en bytes del bloque.

name Texto de referencia para el auditor.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `heap_free` cuando ya no sea necesario.

Observaciones:

Usa esta función para bloques genéricos. Para tipos utiliza `heap_new`.

heap_calloc

Igual que `heap_malloc`, pero inicializando el bloque con 0s.

```
byte_t*
heap_calloc(const uint32_t size,
            const char_t *name);
```

```
byte_t *mem = heap_calloc(256 * 256, "DrawCanvas");
/* mem = {0, 0, 0, 0, ..., 0}; */
...
heap_free(&mem, 256 * 256, "DrawCanvas");
```

size Tamaño en bytes del bloque.

name Texto de referencia para el auditor.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `heap_free` cuando ya no sea necesario.

Observaciones:

Usa esta función para bloques genéricos. Para tipos utiliza `heap_new`.

heap_realloc

Realoja un bloque de memoria existente debido a la expansión o reducción del mismo. Garantiza que se conserva el contenido previo del bloque `min(size, new_size)`. Intenta hacerlo sin mover memoria (in situ), pero si no es posible busca una nueva zona. También garantiza la alineación por defecto `sizeof(void*)` si hay que reservar un nuevo bloque.

```
byte_t*
heap_realloc(byte_t *mem,
             const uint32_t size,
             const uint32_t new_size,
             const char_t *name);
```

```
byte_t *mem = heap_malloc(64, "ArrayData");
...
mem = heap_realloc(mem, 64, 128, ArrayData);
...
heap_free(&mem, 128, "ArrayData");
```

mem	Puntero al bloque original a realojar.
size	Tamaño en bytes del bloque original `mem`.
new_size	Nuevo tamaño requerido, en bytes.
name	Texto de referencia para el auditor. Debe ser el mismo que el utilizado en `heap_malloc`.

Retorna:

Puntero al bloque realojado. Será el mismo que el puntero original `mem` si la reubicación "in-situ" ha tenido éxito. Debe ser liberado con `heap_free` cuando ya no sea necesario.

Observaciones:

Usa esta función para bloques genéricos. Para tipos utiliza `heap_realloc_n`.

heap_aligned_malloc

Reserva un bloque de memoria con alineación.

```
byte_t*
heap_aligned_malloc(const uint32_t size,
                    const uint32_t align,
                    const char_t *name);
```

```
byte_t *sse_data = heap_aligned_malloc(256 * 16, 16, "Vectors");
...
heap_free(&mem, 256 * 16, "Vectors");
```

size	Tamaño en bytes del bloque.
align	Alineación. Debe ser potencia de 2.
name	Texto de referencia para el auditor.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `heap_free` cuando ya no sea necesario.

heap_aligned_calloc

Igual que `heap_aligned_malloc`, pero inicializando el bloque con 0s.

```
byte_t*
heap_aligned_calloc(const uint32_t size,
                    const uint32_t align,
                    const char_t *name);
```

```
byte_t *sse_data = heap_aligned_calloc(256 * 16, 16, "Vectors");
/* see_data = {0, 0, 0, 0, ..., 0}; */
...
heap_free(&mem, 256 * 16, "Vectors");
```

size	Tamaño en bytes del bloque.
align	Alineación. Debe ser potencia de 2.
name	Texto de referencia para el auditor.

Retorna:

Puntero al nuevo bloque. Debe ser liberado con `heap_free` cuando ya no sea necesario.

heap_aligned_realloc

Igual que `heap_realloc`, pero garantizando la alineación de memoria.

```
byte_t*
heap_aligned_realloc(byte_t *mem,
                     const uint32_t size,
                     const uint32_t new_size,
                     const uint32_t align,
                     const char_t *name);
```

```
byte_t *sse_data = heap_aligned_malloc(256 * 16, 16, "Vectors");
...
sse_data = heap_aligned_realloc(sse_data, 256 * 16, 512 * 16, 16, "Vectors");
...
heap_free(&mem, 512 * 16, "Vectors");
```

mem	Puntero al bloque original a realojar.
size	Tamaño en bytes del bloque original `mem`.
new_size	Nuevo tamaño requerido, en bytes.
align	Alineación. Debe ser potencia de 2.
name	Texto de referencia para el auditor. Debe ser el mismo que el utilizado en `heap_aligned_malloc`.

Retorna:

Puntero al bloque realojado. Debe ser liberado con `heap_free` cuando ya no sea necesario.

heap_free

Libera la memoria apuntada por `mem`, previamente reservada por `heap_malloc`, `heap_realloc` o sus equivalentes con alineación.

```
void
heap_free(byte_t **mem,
          const uint32_t size,
          const char_t *name);
```

mem	Doble puntero al bloque a liberar. Será puesto a `NULL` tras la liberación.
size	Tamaño del bloque de memoria.
name	Texto de referencia para el auditor, debe ser el mismo que el utilizado en `heap_malloc`.

Observaciones:

Usa esta función para bloques de memoria genéricos. Para tipos utiliza `heap_delete`.

heap_new

Reserva memoria para un objeto. El puntero de retorno es convertido a `type`.

```
type*
heap_new(type);
```

```
MyAppCtrl *ctrl = heap_new(MyAppCtrl);
...
heap_delete(&ctrl, MyAppCtrl);
```

type	Tipo de objeto.

Retorna:

Puntero al objeto creado. Debe ser destruido por `heap_delete` cuando ya no sea necesario.

heap_new0

Igual que `heap_new`, pero inicializando el objeto con 0s.

```
type*
heap_new0(type);
```

```
MyAppModel *model = heap_new0(MyAppModel);
/* model = {0} */
...
heap_delete(&model, MyAppModel);
```

type Tipo de objeto.

Retorna:

Puntero al objeto creado. Debe ser destruido por `heap_delete` cuando ya no sea necesario.

heap_new_n

Reserva memoria para `n` objetos. El puntero de retorno es convertido a `type`.

```
type*
heap_new_n(const uint32_t n,
           type);
```

```
Car *cars = heap_new_n(10, Car);
...
heap_delete_n(&cars, 10, Car);
```

n Número de objetos a crear.

type Tipo de objeto.

Retorna:

Puntero al array recién creado. Debe ser destruido por `heap_delete_n` cuando ya no sea necesario.

heap_new_n0

Igual que `heap_new_n`, pero inicializando el array con 0s.

```
type*
heap_new_n0(const uint32_t n,
            type);
```

```
Car *cars = heap_new_n0(10, Car);
/* cars = {0, 0, 0, ..., 0}; */
...
heap_delete_n(&cars, 10, Car);
```

n	Número de objetos a crear.
type	Tipo de objeto.

Retorna:

Puntero al array recién creado. Debe ser destruido por `heap_delete_n` cuando ya no sea necesario.

heap_realloc_n

Realoja un array de objetos creados dinámicamente con `heap_new_n` o `heap_new_n0`. Garantiza que los objetos previos permanezcan inalterados `min(size, new_size)`.

```
type*
heap_realloc_n(type *mem,
               const uint32_t size,
               const uint32_t new_size,
               type);
```

```
Car *cars = heap_new_n(10, Car);
...
cars = heap_realloc_n(cars, 10, 20, Car);
/* cars[0]-[9] remains untouched. */
...
heap_delete_n(&cars, 20, Car);
```

mem	Puntero al array a realojar.
size	Número de elementos del array original `mem`.
new_size	Nuevo tamaño requerido (en elementos).
type	Tipo de objeto.

Retorna:

Puntero al array realojado. Debe ser destruido por heap_delete_n cuando ya no sea necesario.

heap_delete

Libera el objeto apuntado por obj, previamente reservado por heap_new o heap_new0.

```
void
heap_delete(type **obj,
            type);
```

obj Doble puntero al objeto a liberar. Será puesto a NULL tras la liberación.

type Tipo de objeto.

heap_delete_n

Libera n objetos apuntados por obj, previamente reservado por heap_new_n, heap_new_n0 .

```
void
heap_delete_n(type **obj,
              const uint32_t n,
              type);
```

obj Doble puntero al array a liberar. Será puesto a NULL tras la liberación.

n Número de objetos a liberar, el mismo que en la reserva.

type Tipo de objeto.

heap_auditor_add

Añade un objeto opaco al auditor de memoria.

```
void
heap_auditor_add(const char_t *name);
```

name Nombre del objeto a añadir.

heap_auditor_delete

Libera un objeto opaco del auditor de memoria.

```
void
heap_auditor_delete(const char_t *name);
```

name Nombre del objeto a liberar.

buffer_create

Crea un nuevo buffer.

```
Buffer*
buffer_create(const uint32_t size);
```

size Tamaño del búfer en bytes.

Retorna:

El nuevo búfer.

buffer_with_data

Crea un nuevo buffer y lo inicializa.

```
Buffer*
buffer_with_data(const byte_t *data,
                 const uint32_t size);
```

data Datos para inicializar el buffer.

size Tamaño del búfer en bytes.

Retorna:

El nuevo búfer.

buffer_destroy

Destruye el búfer.

```
void
buffer_destroy(Buffer **buffer);
```

buffer El búfer. Será puesto a `NULL` tras la destrucción.

buffer_size

Obtiene el tamaño del búfer.

```
uint32_t
buffer_size(const Buffer *buffer);
```

buffer Búfer.

Retorna:

El tamaño del búfer en bytes.

buffer_data

Obtiene un puntero al contenido del búfer.

```
byte_t*
buffer_data(Buffer *buffer);
```

buffer Búfer.

Retorna:

Puntero al contenido del búfer que puede ser utilizado para leer o escribir.

buffer_const

Obtiene un puntero *const* al contenido del búfer.

```
const byte_t*
buffer_const(const Buffer *buffer);
```

buffer Búfer.

Retorna:

Puntero al contenido del búfer que puede ser utilizado solo para leer.

tc

Devuelve la cadena C interna en formato *"UTF-8UTF-8"* (Página 160) contenida en el String.

```
const char_t*
tc(const String *str);
```

str El objeto string.

Retorna:

Puntero a la cadena C.

tcc

Devuelve la cadena C (no const) interna en formato *"UTF-8UTF-8"* (Página 160) contenida en el String.

```
char_t*
tcc(String *str);
```

str El objeto string.

Retorna:

Puntero a la cadena C.

str_c

Crea un String a partir de una cadena C codificada en *"UTF-8UTF-8"* (Página 160).

```
String*
str_c(const char_t *str);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

El objeto string.

str_cn

Crea un String copiando los primeros `n` bytes de una cadena C.

```
String*
str_cn(const char_t *str,
       const uint32_t n);
```

str Cadena C UTF8.

n El número de bytes a copiar.

Retorna:

El objeto string.

Observaciones:

En cadenas *"UTF-8UTF-8"* (Página 160), el número de bytes no corresponde con el número de caracteres.

str_trim

Crea un String a partir de una cadena C recortando los espacios en blanco, tanto al principio como al final.

```
String*
str_trim(const char_t *str);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

El objeto string.

str_trim_n

Crea un String a partir a partir de los primeros `n` bytes de una cadena C recortando los espacios en blanco, tanto al principio como al final.

```
String*
str_trim_n(const char_t *str,
           const uint32_t n);
```

str Cadena C UTF8.

n El número de bytes a considerar de la cadena original. La copia puede contener 'n' o menos bytes, según la cantidad de blancos.

Retorna:

El objeto string.

str_copy

Crea un copia exacta del String.

```
String*
str_copy(const String *str);
```

str El objeto string original.

Retorna:

El objeto string copiado.

Observaciones:

Los Strings son un tipo especial de objeto mutable. Copiar implica crear un nuevo objeto y no incrementar un contador de referencias.

str_printf

Compone un String desde varios campos, utilizando el formato del `printf`.

```
String*
str_printf(const char_t *format,
           ...);
```

format Cadena con el formato tipo-printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El objeto string.

Observaciones:

El uso de esta función previene vulnerabilidades del tipo **buffer overflow**, asociadas a las clásicas funciones C como `strcpy`.

str_path

Igual que `str_printf`, pero considera que la cadena es un *pathname* y, por tanto, utiliza el separador conveniente según `platform`.

```
String*
str_path(const platform_t platform,
         const char_t *format,
         ...);
```

```
String *path = str_path(ekWINDOWS, "/%s/img/%s.png", tc(product->category), tc(
    ↪ product->name));
path = "\\camera\\img\\sony_a5000.png"
```

platform Plataforma para la que se crea el *pathname*.

format Cadena con el formato tipo-printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El objeto string.

str_cpath

Igual que `str_path`, pero considerando la plataforma donde está corriendo el programa.

```
String*
str_cpath(const char_t *format,
          ...);
```

```
String *path = str_cpath("/%s/img/%s.png", tc(product->category), tc(product->
    ↪ name));
path = "\\camera\\img\\sony_a5000.png"  // In Windows
path = "/camera/img/sony_a5000.png"     // In Unix-like
```

format Cadena con el formato tipo-printf con un número variable de parámetros.

... Argumentos o variables del printf.

Retorna:

El objeto string.

str_relpath

Calcula la ruta relativa a `path1` para llegar a `path2`.

```
String*
str_relpath(const platform_t platform,
            const char_t *path1,
            const char_t *path2);
```

platform Plataforma para la que se calcula la ruta (separador de directorios).

path1 La ruta de origen.

path2 La ruta de destino.

Retorna:

El objeto string que contiene la ruta relativa.

str_crelpath

Calcula la ruta relativa a `path1` para llegar a `path2`.

```
String*
str_crelpath(const char_t *path1,
             const char_t *path2);
```

path1 La ruta de origen.

path2 La ruta de destino.

Retorna:

El objeto string que contiene la ruta relativa.

Observaciones:

Igual que `str_relpath`, pero utilizando el separador de directorios de la plataforma donde está corriendo el programa.

str_repl

Crea un String reemplazando un número indeterminado de sub-cadenas. El primer parámetro es la cadena original. Los siguiente pares indican la sub-cadena a buscar y la sub-cadena que debe sustituirla. El último parámetro debe ser 0.

```
String*
str_repl(const char_t *str,
         ...);
```

```
String *str = str_repl("const Product **pr;", "const", "", "*", "", " ", "", 0)
    ↪ ;
str = "Productpr;"
```

str Cadena original C UTF8 terminada en carácter nulo `'\0'`.

... Número de parámetros variable, por pares. El primer elemento del par indica la sub-cadena a buscar en `str`. El segundo elemento la sustituta.

Retorna:

El objeto string.

str_reserve

Crea un String con n+1 bytes, pero sin asignarle ningún contenido.

```
String*
str_reserve(const uint32_t n);
```

n Número de bytes. Reserva espacio para uno más (el `'\n'`).

Retorna:

El objeto string. Su contenido será indeterminado (basura). Debe escribirse posteriormente.

str_fill

Crea un String repitiendo `n` veces un mismo carácter.

```
String*
str_fill(const uint32_t n,
         const char_t c);
```

n Número de caracteres.

c Carácter patrón.

Retorna:

El objeto string.

str_read

Crea un String leyendo su contenido de un `Stream` (de-serialización). El String ha debido ser escrito previamente por `str_write`.

```
String*
str_read(Stream *stream);
```

stream Un *stream* de lectura.

Retorna:

El objeto string.

Observaciones:

Es una operación **binaria**. El tamaño del string se deserializa primero.

str_write

Escribe un string en un *"Streams"* (Página 197) (serialización).

```
void
str_write(Stream *stream,
          String *str);
```

stream Un *stream* de escritura.

str El objeto string.

Observaciones:

Es una operación **binaria**. El tamaño del string se serializa primero. Utiliza `str_writef` para escribir solo el texto.

str_writef

Escribe en un *"Streams"* (Página 197) la cadena C contenida en el string.

```
void
str_writef(Stream *stream,
           String *str);
```

stream Un *stream* de escritura.

str El objeto string.

Observaciones:

Escribe únicamente el texto del *string*, **sin el carácter nulo `'0'`** final. Es equivalente a `stm_writef(stream, tc(str));` pero más eficiente, ya que no hay que calcular el tamaño de `str`.

str_copy_c

Copia la cadena C `src` en el búfer apuntado por `dest`, incluyendo el carácter nulo `'\0'`.

```
void
str_copy_c(char_t *dest,
           const uint32_t size,
           const char_t *str);
```

dest Búfer de destino.

size Tamaño en bytes de `dest`.

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Observaciones:

Es una operación segura. No se escribirán en `dest` más de `size` bytes y nunca se truncará un carácter. `dest` siempre acabará con el carácter nulo `'\0'`.

str_copy_cn

Copia en `dest` un máximo de `n` bytes de la cadena C UTF8 apuntada por `src`, incluyendo el carácter nulo `'\0'`.

```
void
str_copy_cn(char_t *dest,
            const uint32_t size,
            const char_t *str,
            const uint32_t n);
```

dest Búfer de destino.

size Tamaño en bytes de `dest`.

str Cadena C UTF8.

n Máximo número de bytes a copiar en `dest`.

Observaciones:

Es una operación segura. No se escribirán en `dest` más de `n` bytes y nunca se truncará un carácter. `dest` siempre acabará con el carácter nulo `'\0'`.

str_cat

Concatena dinámicamente el contenido de `src` en `dest`.

```
void
str_cat(String **dest,
        const char_t *src);
```

**dest	Objeto *string* de origen y destino.
src	Cadena C UTF8 a concatenar.

Observaciones:

Esta operación implica reasignar memoria dinámica. Para componer textos largos es más eficiente utilizar `Stream`.

str_cat_c

Concatena el contenido de `src` en `dest`. El carácter nulo en `dest` será sobrescrito por el primer carácter de `src`.

```
void
str_cat_c(char_t *dest,
          const uint32_t size,
          const char_t *src);
```

dest	Cadena C UTF8 de origen y destino.
size	Tamaño en bytes de `dest`.
src	Cadena C UTF8 a concatenar.

Observaciones:

Es una operación segura. No se escribirán en `dest` más de `size` bytes y nunca se truncará un carácter. `dest` siempre acabará con el carácter nulo `'\0'`.

str_upd

Cambia el contenido de un *string* por otro.

```
void
str_upd(String **str,
        const char_t *new_str);
```

```
// Equivalent code
String *str = ..original content..
String *temp = str_c(new_str);
str_destroy(&str);
str = temp;
temp = NULL;
```

str Objeto *string* destino. El contenido original será borrado.

new_str Cadena C UTF8 que sustituirá a la original.

str_destroy

Destruye un objeto *string*.

```
void
str_destroy(String **str);
```

str El objeto string. Será puesto a `NULL` tras la destrucción.

str_destopt

Destruye un objeto *string* si su contenido no es `NULL` (destructor opcional).

```
void
str_destopt(String **str);
```

str El objeto string. Será puesto a `NULL` tras la destrucción.

str_len

Retorna el tamaño en bytes de un string.

```
uint32_t
str_len(const String *str);
```

str Objeto string.

Retorna:

El número de bytes, sin incluir el carácter nulo `'\0'`.

Observaciones:

En cadenas *"UTF-8UTF-8"* (Página 160) el número de bytes no es el mismo que el caracteres. `str_nchars`.

str_len_c

Retorna el tamaño en bytes de una cadena C UTF8.

```
uint32_t
str_len_c(const char_t *str);
```

str Cadena C UTF8 terminada en carácter nulo '\0'.

Retorna:

El número de bytes, sin incluir el carácter nulo '\0'.

Observaciones:

En cadenas *"UTF-8UTF-8"* (Página 160) el número de bytes no es el mismo que el caracteres. unicode_nchars.

str_nchars

Retorna el número de caracteres de un objeto string.

```
uint32_t
str_nchars(const String *str);
```

str Objeto string.

Retorna:

El número de caracteres, sin incluir el carácter nulo '\0'.

Observaciones:

En cadenas *"UTF-8UTF-8"* (Página 160) el número de bytes no es el mismo que el caracteres.

str_prefix

Localiza el inicio común de dos cadenas.

```
uint32_t
str_prefix(const char_t *str1,
           const char_t *str2);
```

str1 Primera cadena C UTF8 terminada en carácter nulo '\0'.

str2 Segunda cadena C UTF8 terminada en carácter nulo '\0'.

Retorna:

El número de bytes que son idénticos al comienzo de ambas cadenas.

str_is_prefix

Comprueba si una cadena es prefijo de otra.

```
bool_t
str_is_prefix(const char_t *str,
              const char_t *prefix);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

prefix Prefijo de `str` terminado en carácter nulo `'\0'`.

Retorna:

`TRUE` si `preffix` es prefijo de `str`.

str_is_sufix

Comprueba si una cadena es sufijo de otra.

```
bool_t
str_is_sufix(const char_t *str,
             const char_t *sufix);
```

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

sufix Sufijo de `str` terminado en carácter nulo `'\0'`.

Retorna:

`TRUE` si `sufix` es sufijo de `str`.

str_scmp

Compara alfabéticamente dos strings.

```
int
str_scmp(const String *str1,
         const String *str2);
```

str1 Primer string.

str2 Segundo string.

Retorna:

Resultado de la comparación.

str_cmp

Compara alfabéticamente un string con una cadena C UTF8.

```
int
str_cmp(const String *str1,
        const char_t *str2);
```

str1 Objeto string.

str2 Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

Resultado de la comparación.

str_cmp_c

Compara alfabéticamente dos cadenas C UTF8 terminadas en carácter nulo `'\0'`.

```
int
str_cmp_c(const char_t *str1,
          const char_t *str2);
```

str1 Primera cadena C UTF8.

str2 Segunda cadena C UTF8.

Retorna:

Resultado de la comparación.

str_cmp_cn

Compara alfabéticamente los primeros `n` bytes de dos cadenas C UTF8 terminadas en carácter nulo `'\0'`.

```
int
str_cmp_cn(const char_t *str1,
           const char_t *str2,
           const uint32_t n);
```

str1 Primera cadena C UTF8.

str2 Segunda cadena C UTF8.

n Máximo número de bytes a comparar.

Retorna:

Resultado de la comparación.

Observaciones:

Es una operación segura. Si alguna de las dos cadenas llega al final antes de llegar a `n` bytes, la comparación termina.

str_empty

Comprueba si un string está vacío (`str->data[0] == '\0'`).

```
bool_t
str_empty(const String *str);
```

str Objeto string.

Retorna:

`TRUE` si está vacío o es `NULL`.

str_empty_c

Comprueba si una cadena C UTF8 está vacía (`str[0] == '\0'`).

```
bool_t
str_empty_c(const char_t *str);
```

str Cadena C UTF8.

Retorna:

`TRUE` si está vacía o es `NULL`.

str_equ

Comprueba si el contenido de un string es igual a una cadena C.

```
bool_t
str_equ(const String *str1,
        const char_t *str2);
```

str1 Objeto string.

str2 Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

`TRUE` si son iguales.

str_equ_c

Comprueba si dos cadenas C UTF8 son iguales.

```
bool_t
str_equ_c(const char_t *str1,
          const char_t *str2);
```

str1 Primera cadena C UTF8 acabada en carácter nulo '\0'.

str2 Segunda cadena C UTF8 acabada en carácter nulo '\0'.

Retorna:

TRUE si son iguales.

str_equ_cn

Comprueba si los primeros bytes de dos cadenas C UTF8 son iguales.

```
bool_t
str_equ_cn(const char_t *str1,
           const char_t *str2,
           const uint32_t n);
```

str1 Primera cadena C UTF8 acabada en carácter nulo '\0'.

str2 Segunda cadena C UTF8 acabada en carácter nulo '\0'.

n Primeros 'n' bytes a comparar.

Retorna:

TRUE si son iguales.

Observaciones:

Si se alcanza '\0' en alguna de las dos cadenas, se devolverá TRUE.

str_equ_nocase

Comprueba si dos cadenas C UTF8 son iguales, ignorando mayúsculas o minúsculas.

```
bool_t
str_equ_nocase(const char_t *str1,
               const char_t *str2);
```

str1 Primera cadena C UTF8 acabada en carácter nulo '\0'.

str2 Segunda cadena C UTF8 acabada en carácter nulo '\0'.

Retorna:

`TRUE` si son iguales.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127).

str_equ_end

Comprueba la terminación de una cadena.

```
bool_t
str_equ_end(const char_t *str,
            const char_t *end);
```

str Cadena C UTF8 acabada en carácter nulo `'\0'`.

end Cadena C UTF8 con la terminación.

Retorna:

`TRUE` si `str` acaba en `end`.

str_upper

Cambia los letras minúsculas por mayúsculas.

```
void
str_upper(String *str);
```

str Objeto string.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127). La cadena original cambiará, pero no los requisitos de memoria.

str_lower

Cambia los letras mayúsculas por minúsculas.

```
void
str_lower(String *str);
```

str Objeto string.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127). La cadena original cambiará, pero no los requisitos de memoria.

str_upper_c

Convierte una cadena a mayúsculas.

```
void
str_upper_c(char_t *dest,
            const uint32_t size,
            const char_t *str);
```

dest Búfer destino.

size Tamaño en bytes del búfer destino.

str Cadena C UTF8 acabada en carácter nulo `'\0'`.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127).

str_lower_c

Convierte una cadena a minúsculas.

```
void
str_lower_c(char_t *dest,
            const uint32_t size,
            const char_t *str);
```

dest Búfer destino.

size Tamaño en bytes del búfer destino.

str Cadena C UTF8 acabada en carácter nulo `'\0'`.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127).

str_subs

Cambia todas las instancias de un carácter por otro.

```
void
str_subs(String *str,
         const char_t replace,
         const char_t with);
```

```
String *str = str_c("customer.service.company.com");
str_subs(str, '.', '_');
str_uppercase(str);
str="CUSTOMER_SERVICE_COMPANY_COM"
```

str	Objeto string.
replace	Carácter a reemplazar.
with	Carácter de reemplazo.

Observaciones:

Solo se consideran caracteres US-ASCII (0-127). La cadena original cambiará, pero no los requisitos de memoria.

str_repl_c

Cambia todas las instancias de una subcadena por otra.

```
void
str_repl_c(String *str,
           const char_t *replace,
           const char_t *with);
```

str	Objeto string.
replace	Subcadena a reemplazar.
with	Subcadena de reemplazo.

Observaciones:

Las subcadenas `replace` y `with` deben tener el mismo tamaño, de lo contrario se producirá un *"Asserts"* (Página 155). Utiliza `str_repl` para el caso general.

str_str

Busca una subcadena dentro de otra de mayor tamaño.

```
const char_t*
str_str(const char_t *str,
        const char_t *substr);
```

str	Cadena C UTF8 acabada en carácter nulo `'\0'`.
substr	Subcadena a buscar acabada en carácter nulo `'\0'`.

Retorna:

Puntero a la primera ocurrencia de `substr` en `str` o `NULL` si no existe ninguna.

str_split

Divide una cadena en dos, utilizando la primera ocurrencia de una subcadena.

```
bool_t
str_split(const char_t *str,
          const char_t *substr,
          String **left,
          String **right);
```

```
const char_t *str = "one::two";
String *str1, *str2, *str3;
bool_t ok1, ok2;
ok1 = str_split(str, "::", &str1, &str2);
ok2 = str_split(tc(str1), "::", NULL, &str3);
str1 = "one"
str2 = "two"
str3 = ""
ok1 = TRUE
ok2 = FALSE
```

str	Cadena C UTF8 acabada en carácter nulo `'\0'`.
substr	Subcadena a buscar.
left	Subcadena izquierda. Será igual a `str` si `substr` no existe. El parámetro puede ser `NULL` si no es necesario.
right	Subcadena derecha. Será igual a `""` si `substr` no existe. El parámetro puede ser `NULL` si no es necesario.

Retorna:

`TRUE` si `substr` existe en `str`.

str_split_trim

Igual que `str_split` pero eliminando todos los espacios en blanco al principio y final de `left` y `right`.

```
bool_t
str_split_trim(const char_t *str,
               const char_t *substr,
               String **left,
               String **right);
```

str	Cadena C UTF8 acabada en carácter nulo `'\0'`.
substr	Subcadena a buscar.
left	Subcadena izquierda.
right	Subcadena derecha.

Retorna:

`TRUE` si `substr` existe en `str`.

str_splits

Divide una cadena en varias, utilizando una subcadena como separador.

```
ArrPt(String)*
str_splits(const char_t *str,
           const char_t *substr,
           const bool_t trim);
```

str	Cadena C UTF8 acabada en carácter nulo `'\0'`.
substr	Subcadena a buscar (separador).
trim	Si `TRUE`, las subcadenas eliminarán espacios en blanco al inicio y final.

Retorna:

Array con las subcadenas encontradas. Debe ser destruido con `arrpt_destroy(&array, str_destroy, String)`.

Observaciones:

Igual que `str_split` o `str_split_trim`, pero considerando más de una subcadena.

str_split_pathname

Divide un *pathname* en `path` y `file` *"Filename y pathnameFilename y pathname"* (Página 181).

```
void
str_split_pathname(const char_t *pathname,
                   String **path,
                   String **file);
```

```
String *path, *name, *name2;
str_split_pathname("C:\\Users\\john\\Desktop\\image.png", &path, &name);
str_split_pathname(tc(path), NULL, name2);
path = "C:\\Users\\john\\Desktop"
```

```
name = "image.png"
name2 = "Desktop"
```

pathname Pathname de entrada.

path Ruta de directorios. El parámetro puede ser NULL si no es necesario.

file Nombre de archivo o directorio final. El parámetro puede ser NULL si no es necesario.

str_split_pathext

Igual que str_split_pathname pero extrayendo también la extensión del archivo.

```
void
str_split_pathext(const char_t *pathname,
                  String **path,
                  String **file,
                  String **ext);
```

```
String *path, *name, *ext;
str_split_pathext("C:\\Users\\john\\Desktop\\image.png", &path, &name, &ext);
path = "C:\\Users\\john\\Desktop"
name = "image"
ext = "png"
```

pathname Pathname de entrada.

path Parte path de la ruta.

file Parte file de la ruta.

ext Extensión del archivo.

str_filename

Retorna la parte final de un *pathname*. *"Filename y pathnameFilename y pathname"* (Página 181).

```
const char_t*
str_filename(const char_t *pathname);
```

```
const char_t *name = str_filename("C:\\Users\\john\\Desktop\\image.png");
name = "image.png"
```

pathname Pathname de entrada.

Retorna:

El último tramo de una ruta de directorios.

str_filext

Retorna la extensión del archivo, a partir de un *pathname*. *"Filename y pathnameFilename y pathname"* (Página 181).

```
const char_t*
str_filext(const char_t *pathname);
```

```
const char_t *ext = str_fileext("C:\\Users\\john\\Desktop\\image.png");
ext = "png"
```

pathname Pathname de entrada.

Retorna:

La extensión del archivo.

str_find

Busca una cadena en un array.

```
uint32_t
str_find(const ArrPt(String) *array,
         const char_t *str);
```

array El array.

str La cadena a buscar.

Retorna:

La posición de la cadena o `UINT32_MAX` si no existe.

str_to_i8

Convierte una cadena de texto en un entero.

```
int8_t
str_to_i8(const char_t *str,
          const uint32_t base,
          bool_t *error);
```

str Cadena de texto, terminada en carácter nulo `'\0'`.

base Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).

error Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_i16

Convierte una cadena de texto en un entero.

```
int16_t
str_to_i16(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_i32

Convierte una cadena de texto en un entero.

```
int32_t
str_to_i32(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_i64

Convierte una cadena de texto en un entero.

```
int64_t
str_to_i64(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_u8

Convierte una cadena de texto en un entero.

```
uint8_t
str_to_u8(const char_t *str,
          const uint32_t base,
          bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_u16

Convierte una cadena de texto en un entero.

```
uint16_t
str_to_u16(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_u32

Convierte una cadena de texto en un entero.

```
uint32_t
str_to_u32(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Cadena de texto, terminada en carácter nulo `'\0'`.
base	Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_u64

Convierte una cadena de texto en un entero.

```
uint64_t
str_to_u64(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str Cadena de texto, terminada en carácter nulo `'\0'`.

base Base numérica: 8 (octal), 10 (decimal), 16 (hexadecimal).

error Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0 con `error=TRUE`.

str_to_r32

Convierte una cadena de texto en un real.

```
real32_t
str_to_r32(const char_t *str,
           bool_t *error);
```

str Cadena de texto, terminada en carácter nulo `'\0'`.

error Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0.0 con `error=TRUE`.

str_to_r64

Convierte una cadena de texto en un real.

```
real64_t
str_to_r64(const char_t *str,
           bool_t *error);
```

str Cadena de texto, terminada en carácter nulo `'\0'`.

error Obtiene `TRUE` si hay algún error en la conversión. Puede ser `NULL`.

Retorna:

El valor numérico.

Observaciones:

Si la cadena es errónea o el valor está fuera de rango, devuelve 0.0 con `error=TRUE`.

stm_from_block

Crea un stream de lectura a partir de un bloque de memoria ya existente.

```
Stream*
stm_from_block(const byte_t *data,
               const uint32_t size);
```

data Puntero al bloque de memoria.

size Tamaño en bytes del bloque de memoria.

Retorna:

El objeto stream.

Observaciones:

El bloque original no será modificado (solo lectura). Cuando se alcance el final del bloque `stm_state` devolverá `ekSTEND`. *"Block streamBlock stream"* (Página 199).

stm_memory

Crea un stream de lectura/escritura en memoria.

```
Stream*
stm_memory(const uint32_t size);
```

size Tamaño inicial del búfer (en bytes). Crecerá si es necesario.

Retorna:

El objeto stream.

Observaciones:

Puede utilizarse como tubería interna para el intercambio de información entre funciones o hilos. Se comporta como un búfer FIFO *(First In Fist Out)*. Para acceso multi-hilo debe estar protegido con un `Mutex`. *"Memory streamMemory stream"* (Página 200).

stm_from_file

Crea un stream para leer desde un archivo en disco.

```
Stream*
stm_from_file(const char_t *pathname,
              ferror_t *error);
```

pathname *Pathname* del archivo. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser NULL.

Retorna:

El objeto stream o NULL si falla la apertura del archivo.

Observaciones:

"File streamFile stream" (Página 198).

stm_to_file

Crea un stream para escribir datos a un archivo en disco.

```
Stream*
stm_to_file(const char_t *pathname,
            ferror_t *error);
```

pathname *Pathname* del archivo. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser NULL.

Retorna:

El objeto stream o NULL si falla la creación del archivo.

Observaciones:

Si el archivo ya existe será sobrescrito. *"File streamFile stream"* (Página 198).

stm_append_file

Crea un stream para escribir datos al final de un archivo existente.

```
Stream*
stm_append_file(const char_t *pathname,
                ferror_t *error);
```

pathname *Pathname* del archivo. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

El objeto stream o `NULL` si falla la apertura del archivo.

Observaciones:

Fallará si el archivo no existe (no lo crea). *"File streamFile stream"* (Página 198).

stm_socket

Crea un stream a partir de un socket.

```
Stream*
stm_socket(Socket *socket);
```

socket Socket cliente o servidor.

Retorna:

El objeto stream.

Observaciones:

Permite utilizar la funcionalidad de los streams para leer o escribir en un proceso remoto. El socket debe haber sido previamente creado con `bsocket_connect` (cliente) o `bsocket_accept` (servidor). Ver *"Socket streamSocket stream"* (Página 199).

stm_close

Cierra el stream. Todos los recursos como descriptores de archivos o *sockets* serán liberados. Previamente al cierre, se escribirán en el canal los datos en caché `stm_flush`.

```
void
stm_close(Stream **stm);
```

stm El objeto stream. Será puesto a `NULL` tras la destrucción.

stm_get_write_endian

Obtiene el orden de bytes actual al escribir en el stream.

```
endian_t
stm_get_write_endian(const Stream *stm);
```

stm El objeto stream.

Retorna:

El *"Orden de bytesOrden de bytes"* (Página 211).

stm_get_read_endian

Obtiene el orden de bytes actual al leer del stream.

```
endian_t
stm_get_read_endian(const Stream *stm);
```

stm El objeto stream.

Retorna:

El *"Orden de bytesOrden de bytes"* (Página 211).

stm_set_write_endian

Establece el orden de bytes al escribir en el stream, a partir de ahora.

```
void
stm_set_write_endian(Stream *stm,
                     const endian_t endian);
```

stm El objeto stream.

endian El *"Orden de bytesOrden de bytes"* (Página 211).

Observaciones:

Por defecto es `ekLITEND`, salvo en sockets que será `ekBIGEND`.

stm_set_read_endian

Establece el orden de bytes al leer del stream, a partir de ahora.

```
void
stm_set_read_endian(Stream *stm,
                    const endian_t endian);
```

stm El objeto stream.

endian El *"Orden de bytesOrden de bytes"* (Página 211).

Observaciones:

Por defecto es `ekLITEND`, salvo en sockets que será `ekBIGEND`.

stm_get_write_utf

Obtiene la codificación UTF con la que se están escribiendo los textos en el stream.

```
unicode_t
stm_get_write_utf(const Stream *stm);
```

stm El objeto stream.

Retorna:

"Codificaciones UTFCodificaciones UTF" (Página 159).

Observaciones:

Ver *"Stream de textoStream de texto"* (Página 202).

stm_get_read_utf

Obtiene la codificación UTF con la que se están leyendo los textos en el stream.

```
unicode_t
stm_get_read_utf(const Stream *stm);
```

stm El objeto stream.

Retorna:

"Codificaciones UTFCodificaciones UTF" (Página 159).

Observaciones:

Ver *"Stream de textoStream de texto"* (Página 202).

stm_set_write_utf

Establece la codificación UTF al escribir textos en el stream, a partir de ahora.

```
void
stm_set_write_utf(Stream *stm,
                  const unicode_t format);
```

stm El objeto stream.

format *"Codificaciones UTFCodificaciones UTF"* (Página 159).

Observaciones:

Ver *"Stream de textoStream de texto"* (Página 202).

stm_set_read_utf

Establece la codificación UTF al leer textos en el stream, a partir de ahora.

```
void
stm_set_read_utf(Stream *stm,
                 const unicode_t format);
```

stm El objeto stream.

format *"Codificaciones UTFCodificaciones UTF"* (Página 159).

Observaciones:

Ver *"Stream de textoStream de texto"* (Página 202).

stm_is_memory

Obtiene si es un stream residente en memoria.

```
bool_t
stm_is_memory(const Stream *stm);
```

stm El objeto stream.

Retorna:

`TRUE` si ha sido creado mediante `stm_from_block` o `stm_memory`.

stm_bytes_written

Obtiene el total de bytes escritos en el stream desde su creación.

```
uint64_t
stm_bytes_written(const Stream *stm);
```

stm El objeto stream.

Retorna:

El número total de bytes escritos.

stm_bytes_readed

Obtiene el total de bytes leídos desde el stream desde su creación.

```
uint64_t
stm_bytes_readed(const Stream *stm);
```

stm El objeto stream.

Retorna:

El número total de bytes leídos.

stm_col

Obtiene la columna en streams de texto.

```
uint32_t
stm_col(const Stream *stm);
```

stm El objeto stream.

Retorna:

Número de columna.

Observaciones:

Cuando leemos caracteres en streams de texto con `stm_read_char` o derivadas, se contabilizan las columnas y filas de forma similar a como lo hacen los editores de texto. Esta información puede ser útil a la hora de mostrar avisos o mensajes de error. En streams mixtos (binarios + texto), la cuenta se detiene al leer datos binarios y continúa cuando se retoma la lectura del texto. Ver *"Stream de textoStream de texto"* (Página 202).

stm_row

Obtiene la fila en streams de texto.

```
uint32_t
stm_row(const Stream *stm);
```

stm El objeto stream.

Retorna:

Número de fila.

Observaciones:

Ver `stm_col`.

stm_token_col

Obtiene la columna del último token leído.

```
uint32_t
stm_token_col(const Stream *stm);
```

stm El objeto stream.

Retorna:

Número de columna.

Observaciones:

Solo tiene efecto tras llamar a `stm_read_token` o derivadas. Ver `stm_col` y *"Tokens-Tokens"* (Página 203).

stm_token_row

Obtiene la fila del último token leído.

```
uint32_t
stm_token_row(const Stream *stm);
```

stm El objeto stream.

Retorna:

Número de fila.

Observaciones:

Solo tiene efecto tras llamar a `stm_read_token` o derivadas. Ver `stm_col` y *"Tokens-Tokens"* (Página 203).

stm_token_lexeme

Obtiene el lexema del último token leído.

```
const char_t*
stm_token_lexeme(const Stream *stm);
```

stm El objeto stream.

Retorna:

El lexema. Está almacenado en un buffer temporal y se perderá al leer el siguiente token. Si lo necesitas, haz una copia con `str_c`.

Observaciones:

Solo tiene efecto tras llamar a stm_read_token o derivadas. Ver "TokensTokens" (Página 203).

stm_token_escapes

Opción de secuencias de escape al leer tokens.

```
void
stm_token_escapes(const Stream *stm,
                  const bool_t active_escapes);
```

stm	El objeto stream.
active_escapes	TRUE se procesarán las secuencias de escape al leer tokens ekTSTRING. Por ejemplo, la secuencia "\n" se transformará en el carácter 0x0A (10). FALSE ignorará las secuencias de escape, leyendo las cadenas de texto de forma literal. Por defecto FALSE.

Observaciones:

Tendrá efecto en la siguiente llamada a stm_read_token. Ver "TokensTokens" (Página 203).

stm_token_spaces

Opción de espacios en blanco al leer tokens.

```
void
stm_token_spaces(const Stream *stm,
                 const bool_t active_spaces);
```

stm	El objeto stream.
active_spaces	TRUE se devolverán tokens ekTSPACE al encontrar secuencias de espacios en blanco. FALSE ignorarán los espacios en blanco. Por defecto FALSE.

Observaciones:

Tendrá efecto en la siguiente llamada a stm_read_token. Ver "TokensTokens" (Página 203).

stm_token_comments

Opción de comentarios al leer tokens.

```
void
stm_token_comments(const Stream *stm,
                   const bool_t active_comments);
```

stm — El objeto stream.

active_comments — `TRUE` se devolverá un token `ekTMLCOM` cada vez que encuentre comentarios de C `/* Comment */` y `ekTSLCOM` para comentarios C++ `// Comment`. `FALSE` se ignorarán los comentarios. Por defecto `FALSE`.

Observaciones:

Tendrá efecto en la siguiente llamada a `stm_read_token`. Ver *"TokensTokens"* (Página 203).

stm_state

Obtiene el estado actual del stream.

```
sstate_t
stm_state(const Stream *stm);
```

stm — El objeto stream.

Retorna:

El *"Estado del streamEstado del stream"* (Página 211).

stm_file_err

Obtiene información adicional sobre el error, en streams de disco.

```
ferror_t
stm_file_err(const Stream *stm);
```

stm — El objeto stream.

Retorna:

Error de archivo.

Observaciones:

Solo es relevante en *"File streamFile stream"* (Página 198) con el estado `ekSTBROKEN`.

stm_sock_err

Obtiene información adicional sobre el error, en streams de red.

```
serror_t
stm_sock_err(const Stream *stm);
```

stm El objeto stream.

Retorna:

Error de socket.

Observaciones:

Solo es relevante en *"Socket streamSocket stream"* (Página 199) con el estado ekSTBROKEN .

stm_corrupt

Establece el estado del stream a ekSTCORRUPT.

```
void
stm_corrupt(Stream *stm);
```

stm El objeto stream.

Observaciones:

En ocasiones, es la aplicación la que detecta que los datos están corruptos ya que la semántica de los mismos no coincide con lo esperado.

stm_str

Crea un string con el contenido actual del búfer interno. Solo es válido para stream en memoria. stm_memory.

```
String*
stm_str(const Stream *stm);
```

stm El objeto stream.

Retorna:

El string con el contenido del búfer.

stm_buffer

Obtiene un puntero al contenido actual del búfer interno. Solo es válido para stream en memoria. `stm_memory`.

```
const byte_t*
stm_buffer(const Stream *stm);
```

stm El objeto stream.

Retorna:

Puntero al búfer interno.

Observaciones:

Este puntero es de solo lectura. Escribir aquí tendrá consecuencias indeseadas.

stm_buffer_size

Obtiene el tamaño actual del búfer interno. Solo es válido para stream en memoria. `stm_memory`.

```
uint32_t
stm_buffer_size(const Stream *stm);
```

stm El objeto stream.

Retorna:

El tamaño del búfer interno (en bytes).

stm_write

Escribe bytes en el stream.

```
void
stm_write(Stream *stm,
          const byte_t *data,
          const uint32_t size);
```

stm El objeto stream.

data Puntero al bloque de datos a escribir.

size Número de bytes a escribir.

Observaciones:

El bloque se escribe tal cual llega, sin tener en cuenta el *"Orden de bytesOrden de bytes"* (Página 211) ni las *"Codificaciones UTFCodificaciones UTF"* (Página 159).

stm_write_char

Escribe un carácter Unicode en el stream.

```
void
stm_write_char(Stream *stm,
               const uint32_t codepoint);
```

stm	El objeto stream.
codepoint	El código *"Unicode"* (Página 157) del carácter.

Observaciones:

La codificación se puede cambiar con `stm_set_write_utf`.

stm_printf

Escribe texto en el stream, utilizando el formato del `printf`.

```
uint32_t
stm_printf(Stream *stm,
           const char_t *format,
           ...);
```

```
stm_printf(stream, Code: %-10s Price %5.2f\n", code, price);
```

stm	El objeto stream.
format	Cadena con el formato tipo-printf con un número variable de parámetros.
...	Argumentos o variables del printf.

Retorna:

El número de bytes escritos.

Observaciones:

El carácter nulo final (`'\0'`) no será escrito. La codificación se puede cambiar con `stm_set_write_utf`.

stm_writef

Escribe una cadena C UTF8 en el stream.

```
uint32_t
stm_writef(Stream *stm,
           const char_t *str);
```

stm El objeto stream.

str Cadena C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

El número de bytes escritos.

Observaciones:

El carácter nulo final (`'\0'`) no será escrito. Está función es más rápida que `stm_printf` cuando la cadena es constante y no necesita formato. Para objetos `String` utiliza `str_writef` . La codificación se puede cambiar con `stm_set_write_utf`.

stm_write_bool

Escribe una variable `bool_t` en el stream.

```
void
stm_write_bool(Stream *stm,
               const bool_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto “puros”.

stm_write_i8

Escribe una variable `int8_t` en el stream.

```
void
stm_write_i8(Stream *stm,
             const int8_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto “puros”.

stm_write_i16

Escribe una variable int16_t en el stream.

```
void
stm_write_i16(Stream *stm,
              const int16_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_i32

Escribe una variable int32_t en el stream.

```
void
stm_write_i32(Stream *stm,
              const int32_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_i64

Escribe una variable int64_t en el stream.

```
void
stm_write_i64(Stream *stm,
              const int64_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_u8

Escribe una variable `uint8_t` en el stream.

```
void
stm_write_u8(Stream *stm,
             const uint8_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros".

stm_write_u16

Escribe una variable `uint16_t` en el stream.

```
void
stm_write_u16(Stream *stm,
              const uint16_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_u32

Escribe una variable `uint32_t` en el stream.

```
void
stm_write_u32(Stream *stm,
              const uint32_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_u64

Escribe una variable uint64_t en el stream.

```
void
stm_write_u64(Stream *stm,
              const uint64_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_r32

Escribe una variable real32_t en el stream.

```
void
stm_write_r32(Stream *stm,
              const real32_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_r64

Escribe una variable real64_t en el stream.

```
void
stm_write_r64(Stream *stm,
              const real64_t value);
```

stm El objeto stream.

value Variable a escribir.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_write_enum

Escribe una variable `enum` en el stream.

```
void
stm_write_enum(Stream *stm,
               const type value,
               type);
```

stm — El objeto stream.

value — Variable a escribir.

type — El tipo del **enum**.

Observaciones:

Es una escritura binaria. No utilizar en streams de texto "puros". *"Orden de bytesOrden de bytes"* (Página 211).

stm_read

Lee bytes desde el stream.

```
uint32_t
stm_read(Stream *stm,
         byte_t *data,
         const uint32_t size);
```

stm — El objeto stream.

data — Puntero al búfer donde se escribirán los datos leídos.

size — La cantidad de bytes a leer (tamaño del búfer).

Retorna:

El número de bytes realmente leídos.

stm_read_char

Lee un carácter de texto desde el stream.

```
uint32_t
stm_read_char(Stream *stm);
```

stm — El objeto stream.

Retorna:

El código Unicode del carácter.

Observaciones:

La codificación del texto de entrada puede ajustarse con `stm_set_read_utf`. Actualizará el contador de filas y columnas. Ver `stm_col`.

stm_read_chars

Lee varios caracteres desde el stream.

```
const char_t*
stm_read_chars(Stream *stm,
               const uint32_t n);
```

stm El objeto stream.

n La cantidad de caracteres a leer.

Retorna:

Puntero a la cadena C UTF8 leída. Terminará con el carácter nulo `'\0'`.

Observaciones:

El puntero devuelto es temporal y será sobrescrito en la próxima lectura. Si es necesario, haz una copia con `str_c`. La codificación del texto de entrada puede ajustarse con `stm_set_read_utf`. Actualizará el contador de filas y columnas. Ver `stm_col`.

stm_read_line

Lee caracteres del stream hasta alcanzar un final de línea `'\n'`.

```
const char_t*
stm_read_line(Stream *stm);
```

stm El objeto stream.

Retorna:

Puntero a la cadena C UTF8, terminada con el carácter nulo `'\0'`. Los caracteres `'\n'` o `'\r\n'` no serán incluidos en el resultado. Se devolverá `NULL` cuando se alcance el final del stream.

Observaciones:

El puntero devuelto es temporal y será sobrescrito en la próxima lectura. Si es necesario, haz una copia con `str_c`. La codificación del texto de entrada puede ajustarse con `stm_set_read_utf`. Actualizará el contador de filas y columnas. Ver `stm_col`.

stm_read_trim

Lee la siguiente secuencia de caracteres eliminando los espacios en blanco.

```
const char_t*
stm_read_trim(Stream *stm);
```

stm El objeto stream.

Retorna:

Puntero a la cadena C UTF8 leída. Terminará con el carácter nulo `'\0'`.

Observaciones:

Útil para leer cadenas desde streams de texto. Ignorará todos los espacios en blanco iniciales y leerá caracteres hasta encontrar el primer blanco (`' ', '\\t', '\\n', '\\v', '\\f', '\\r'`). Si necesitas mayor control sobre los *tokens* utiliza `stm_read_token`. El puntero devuelto es temporal y será sobrescrito en la próxima lectura. Si es necesario, haz una copia con `str_c`. La codificación del texto de entrada puede ajustarse con `stm_set_read_utf`. Actualizará el contador de filas y columnas. Ver `stm_col`.

stm_read_token

Obtiene el siguiente token en *"Stream de textoStream de texto"* (Página 202).

```
token_t
stm_read_token(Stream *stm);
```

stm El objeto stream.

Retorna:

El tipo de token obtenido.

Observaciones:

Para obtener la cadena de texto asociada al token, utiliza `stm_token_lexeme`. Ver *"TokensTokens"* (Página 203).

stm_read_i8_tok

Lee el siguiente token con `stm_read_token` y, si es un número entero, lo convierte a `int8_t`.

```
int8_t
stm_read_i8_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

En caso que no se pueda leer un token de tipo `ekTINTEGER` (con o sin `ekTMINUS`) o que el valor numérico esté fuera de rango, se devolverá 0 y se marcará el stream como corrupto con `stm_corrupt`.

stm_read_i16_tok

Lee el siguiente token y lo convierte a `int16_t`.

```
int16_t
stm_read_i16_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_i8_tok`.

stm_read_i32_tok

Lee el siguiente token y lo convierte a `int32_t`.

```
int32_t
stm_read_i32_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_i8_tok`.

stm_read_i64_tok

Lee el siguiente token y lo convierte a `int64_t`.

```
int64_t
stm_read_i64_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_i8_tok`.

stm_read_u8_tok

Lee el siguiente token con `stm_read_token` y, si es un número entero, lo convierte a `uint8_t`.

```
uint8_t
stm_read_u8_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

En caso que no se pueda leer un token de tipo `ekTINTEGER` o que el valor numérico esté fuera de rango, se devolverá 0 y se marcará el stream como corrupto con `stm_corrupt`.

stm_read_u16_tok

Lee el siguiente token y lo convierte a `uint16_t`.

```
uint16_t
stm_read_u16_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_u8_tok`.

stm_read_u32_tok

Lee el siguiente token y lo convierte a `uint32_t`.

```
uint32_t
stm_read_u32_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_u8_tok`.

stm_read_u64_tok

Lee el siguiente token y lo convierte a `uint64_t`.

```
uint64_t
stm_read_u64_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_u8_tok`.

stm_read_r32_tok

Lee el siguiente token con `stm_read_token` y, si es un número real, lo convierte a `real32_t` .

```
real32_t
stm_read_r32_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

En caso que no se pueda leer un token de tipo `ekTINTEGER` o `ekTREAL` (con o sin `ekTMINUS`), se devolverá 0 y se marcará el stream como corrupto con `stm_corrupt`.

stm_read_r64_tok

Lee el siguiente token y lo convierte a `real64_t`.

```
real64_t
stm_read_r64_tok(Stream *stm);
```

stm El objeto stream.

Retorna:

El valor numérico del token.

Observaciones:

Ver `stm_read_r32_tok`.

stm_read_bool

Lee una variable `bool_t` desde el stream.

```
bool_t
stm_read_bool(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerá 1 byte. Si el valor es diferente de 0 o 1, se marcará el stream como corrupto con `stm_corrupt`. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_i8

Lee una variable `int8_t` desde el stream.

```
int8_t
stm_read_i8(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerá 1 byte. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_i16

Lee una variable `int16_t` desde el stream.

```
int16_t
stm_read_i16(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 2 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_i32

Lee una variable `int32_t` desde el stream.

```
int32_t
stm_read_i32(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 4 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_i64

Lee una variable `int64_t` desde el stream.

```
int64_t
stm_read_i64(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 8 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_u8

Lee una variable `uint8_t` desde el stream.

```
uint8_t
stm_read_u8(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerá 1 byte. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_u16

Lee una variable `uint16_t` desde el stream.

```
uint16_t
stm_read_u16(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 2 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_u32

Lee una variable `uint32_t` desde el stream.

```
uint32_t
stm_read_u32(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 4 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_u64

Lee una variable `uint64_t` desde el stream.

```
uint64_t
stm_read_u64(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 8 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_r32

Lee una variable `real32_t` desde el stream.

```
real32_t
stm_read_r32(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 4 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_r64

Lee una variable `real64_t` desde el stream.

```
real64_t
stm_read_r64(Stream *stm);
```

stm El objeto stream.

Retorna:

Variable leída.

Observaciones:

Se leerán 8 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_read_enum

Lee una variable `enum` desde el stream.

```
type
stm_read_enum(Stream *stm,
              type);
```

stm El objeto stream.

type El tipo del **enum**.

Retorna:

Variable leída.

Observaciones:

Se leerán 4 bytes. Ver *"Stream binarioStream binario"* (Página 202).

stm_skip

Salta e ignora los próximos bytes del stream.

```
void
stm_skip(Stream *stm,
         const uint32_t size);
```

stm El objeto stream.

size La cantidad de bytes a saltar.

stm_skip_bom

Salta la posible secuencia *Byte Order Mark* `"ï»?`"` que se encuentra al comienzo de algunos streams de texto.

```
void
stm_skip_bom(Stream *stm);
```

stm El objeto stream.

Observaciones:

Esta función no tendrá efecto si no existe dicha secuencia al inicio del stream. El BOM es habitual en streams provenientes de algunos servidores Web.

stm_skip_token

Salta el siguiente token del stream. Si el token no corresponde al indicado, se marcará el stream como corrupto.

```
void
stm_skip_token(Stream *stm,
               const token_t token);
```

```
void stm_skip_token(Stream *stm, const token_t token)
{
    token_t tok = stm_read_token(stm);
    if (tok != token)
        stm_corrupt(stm);
}
```

stm El objeto stream.

token Token esperado.

stm_flush

Escribe en el canal la información existente en la caché.

```
void
stm_flush(Stream *stm);
```

stm El objeto stream.

Observaciones:

Para mejorar el rendimiento, las operaciones de escritura en streams de disco o E/S estándar se almacenan en una caché interna. Esta función fuerza la escritura en el canal y

limpia el búfer. Será útil con protocolos *full-duplex* donde el receptor espera contestación para continuar.

stm_pipe

Conecta dos streams, leyendo datos de uno y escribiéndolos en otro.

```
void
stm_pipe(Stream *from,
         Stream *to,
         const uint32_t n);
```

from El objeto stream de entrada (para leer).

to El objeto stream de salida (para escribir).

n La cantidad de bytes a trasvasar.

Observaciones:

El trasvase se realizará sobre datos "en bruto", sin tener en cuenta *"Orden de bytesOrden de bytes"* (Página 211) o *"Codificaciones UTFCodificaciones UTF"* (Página 159). Si tienes claro que esto no afecta, es mucho más rápido que utilizar operaciones atómicas de lectura/escritura.

stm_lines

Itera sobre todas las líneas en un *"Stream de textoStream de texto"* (Página 202). Debes usar `stm_next` para cerrar el bucle.

```
void
stm_lines(const char_t *line,
          Stream *stm);
```

```
uint32_t i = 1;
Stream *stm = stm_from_file("/home/john/friends.txt", NULL);
stm_lines(line, stm)
    bstd_printf("Friend %d, name %s\n", i++, line);
stm_next(line, stm);
stm_close(&stm);
```

line Nombre de la variable que albergará temporalmente la línea. Utiliza una caché interna del stream, por lo que deberás hacer una copia con `str_c` si necesitas conservarla.

stm Stream.

stm_next

Cierra un bucle abierto por `stm_lines`.

```
void
stm_next(const char_t *line,
         Stream *stm);
```

line Nombre de la variable línea.

stm Stream.

arrst_create

Crea un array vacío.

```
ArrSt(type)*
arrst_create(type);
```

type Tipo de objeto.

Retorna:

El nuevo array.

arrst_copy

Crea una copia de un array.

```
ArrSt(type)*
arrst_copy(const ArrSt(type) *array,
           FPtr_scopy func_copy,
           type);
```

array El array original.

func_copy Función que debe copiar los campos de cada objeto.

type Tipo de objeto.

Retorna:

La copia del array original.

Observaciones:

La función de copia debe asignar memoria a los campos que lo requieran, pero NO al objeto en sí mismo. Si pasamos `NULL`, se realizará una copia byte a byte del objeto original, lo que puede suponer un riesgo de integridad si los elementos del array contienen objetos

`String` o de otro tipo que necesiten memoria dinámica. Ver *"Copia de objetosCopia de objetos"* (Página 218).

arrst_read

Crea un array leyendo su contenido de un *"Streams"* (Página 197) (de-serialización).

```
ArrSt(type)*
arrst_read(Stream *stream,
           FPtr_read_init func_read,
           type);
```

stream — Un *stream* de lectura.

func_read — Función para inicializar un objeto a partir de los datos obtenidos de un stream. Esta función no debe reservar memoria para el propio objeto (el contenedor ya lo hace). *"SerializaciónSerialización"* (Página 218).

type — Tipo de objeto.

Retorna:

El array leído.

arrst_destroy

Destruye un array y todos sus elementos.

```
void
arrst_destroy(ArrSt(type) **array,
              FPtr_remove func_remove,
              type);
```

array — El array. Será puesto a `NULL` tras la destrucción.

func_remove — Función que debe liberar la memoria asociada a los campos del objeto, pero no el objeto en si mismo *"DestructoresDestructores"* (Página 219). Si es `NULL` se liberará solo el array y no el contenido interno de los elementos.

type — Tipo de objeto.

arrst_destopt

Destruye un array y todos sus elementos, siempre y cuando el objeto array no sea `NULL`.

```
void
arrst_destopt(ArrSt(type) **array,
              FPtr_remove func_remove,
```

```
            type);
```

array El array.

func_remove Ver `arrst_destroy`.

type Tipo de objeto.

arrst_clear

Borra el contenido del array, sin destruir el contenedor que quedará con cero elementos.

```
void
arrst_clear(ArrSt(type) *array,
            FPtr_remove func_remove,
            type);
```

array El array.

func_remove Función 'remove'. Ver `arrst_destroy`.

type Tipo de objeto.

arrst_write

Escribe un array en un *"Streams"* (Página 197) (serialización).

```
void
arrst_write(Stream *stream,
            const ArrSt(type) *array,
            FPtr_write func_write,
            type);
```

stream Un *stream* de escritura.

array El array.

func_write Función que escribe el contenido de un elemento en un stream *"SerializaciónSerialización"* (Página 218).

type Tipo de objeto.

arrst_size

Obtiene el número de elementos en un array.

```
uint32_t
arrst_size(const ArrSt(type) *array,
           type);
```

array El array.

type Tipo de objeto.

Retorna:

Número de elementos.

arrst_get

Obtiene un puntero al elemento en la posición `pos`.

```
type*
arrst_get(ArrSt(type) *array,
          const uint32_t pos,
          type);
```

array El array.

pos Posición o índice del elemento.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_get_const

Obtiene un puntero **const** al elemento en la posición `pos`.

```
const type*
arrst_get_const(const ArrSt(type) *array,
                const uint32_t pos,
                type);
```

array El array.

pos Posición o índice del elemento.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_first

Obtiene un puntero al primer elemento del array.

```
type*
arrst_first(ArrSt(type) *array,
            type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_first_const

Obtiene un puntero **const** al primer elemento del array.

```
const type*
arrst_first_const(const ArrSt(type) *array,
                  type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_last

Obtiene un puntero al último elemento del array.

```
type*
arrst_last(ArrSt(type) *array,
           type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_last_const

Obtiene un puntero **const** al último elemento del array.

```
const type*
arrst_last_const(const ArrSt(type) *array,
                 type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrst_all

Obtiene un puntero a la memoria interna del array, que da acceso directo a todos los elementos.

```
type*
arrst_all(ArrSt(type) *array,
          type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero base. Incrementándolo uno a uno iteraremos sobre los elementos.

Observaciones:

Utiliza `arrst_foreach` para iterar sobre los elementos de forma más segura y elegante.

arrst_all_const

Obtiene un puntero **const** a la memoria interna del array, que da acceso directo a todos los elementos.

```
const type*
arrst_all_const(const ArrSt(type) *array,
                type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero base. Incrementándolo uno a uno iteraremos sobre los elementos.

Observaciones:

Utiliza `arrst_foreach_const` para iterar sobre los elementos de forma más segura y elegante.

arrst_grow

Añade `n` elementos, sin inicializar, al final del array.

```
void
arrst_grow(ArrSt(type) *array,
           const uint32_t n,
           type);
```

array	El array.
n	Cantidad de elementos a añadir.
type	Tipo de objeto.

arrst_new

Reserva espacio para un elemento al final del array.

```
type*
arrst_new(ArrSt(type) *array,
          type);
```

```
// arrst_append copies 'product'
Product product;
i_init_product(&product, ...);
arrst_append(array, product, Product);

// arrst_new avoids the copy
Product *product = arrst_new(array, Product);
i_init_product(product, ...);
```

array	El array.
type	Tipo de objeto.

Retorna:

Puntero al elemento añadido.

Observaciones:

Es ligeramente más rápido que `arrst_append`, especialmente en estructuras grandes, ya que evita copiar el contenido del objeto. El contenido inicial de la memoria es indeterminado.

arrst_new0

Reserva espacio para un elemento al final del array y lo inicializa a 0.

```
type*
arrst_new0(ArrSt(type) *array,
           type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento añadido.

Observaciones:

Igual que `arrst_new` pero inicializando toda la memoria a 0.

arrst_new_n

Reserva espacio para varios elementos al final del array.

```
type*
arrst_new_n(ArrSt(type) *array,
            const uint32_t n,
            type);
```

array El array.

n Número de elementos a añadir.

type Tipo de objeto.

Retorna:

Puntero al primer elemento añadido.

Observaciones:

Igual que `arrst_new` pero reservando varios elementos en la misma llamada. El contenido inicial de la memoria es indeterminado.

arrst_new_n0

Reserva espacio para varios elementos al final del array y los inicializa a 0.

```
type*
arrst_new_n0(ArrSt(type) *array,
```

```
                const uint32_t n,
                type);
```

array El array.

n Número de elementos a añadir.

type Tipo de objeto.

Retorna:

Puntero al primer elemento añadido.

Observaciones:

Igual que `arrst_new_n` pero pero inicializando toda la memoria a 0.

arrst_prepend_n

Reserva espacio para varios elemento al principio del array. El resto de elementos serán desplazados a la derecha.

```
type*
arrst_prepend_n(ArrSt(type) *array,
                const uint32_t n,
                type);
```

array El array.

n Número de elementos a insertar.

type Tipo de objeto.

Retorna:

Puntero al primer elemento insertado.

Observaciones:

El contenido inicial de la memoria es indeterminado.

arrst_insert_n

Reserva espacio para varios elemento en una posición arbitraria del array.

```
type*
arrst_insert_n(ArrSt(type) *array,
               const uint32_t pos,
               const uint32_t n,
               type);
```

array El array.

pos Posición donde será insertado. El actual elemento en `pos` y siguientes serán desplazados a la derecha.

n Número de elementos a insertar.

type Tipo de objeto.

Retorna:

Puntero al primer elemento insertado.

Observaciones:

El contenido inicial de la memoria es indeterminado.

arrst_append

Añade un elemento al final del array.

```
void
arrst_append(ArrSt(type) *array,
             type value,
             type);
```

array El array.

value Elemento a añadir.

type Tipo de objeto.

arrst_prepend

Inserta un elemento al inicio del array. El resto de elementos serán desplazados a la derecha.

```
void
arrst_prepend(ArrSt(type) *array,
              type value,
              type);
```

array El array.

value Elemento a insertar.

type Tipo de objeto.

arrst_insert

Inserta un elemento en una posición arbitraria del array.

```
void
arrst_insert(ArrSt(type) *array,
             const uint32_t pos,
             type value,
             type);
```

array El array.

pos Posición donde será insertado. El actual elemento en `pos` y siguientes serán desplazados a la derecha.

value Elemento a insertar.

type Tipo de objeto.

arrst_join

Une dos vectores. Añade todos los elementos de `src` al final de `dest`.

```
void
arrst_join(ArrSt(type) *dest,
           const ArrSt(type) *src,
           FPtr_scopy func_copy,
           type);
```

```
ArrSt(Product) *products = create_products(...);
ArrSt(Product) *new_products = new_products(...);

// Join without 'copy' func. Dynamic 'Product' fields will be reused.
arrst_join(products, new_products, NULL, Product);
arrst_destroy(&new_products, NULL, Product);
...
arrst_destroy(&products, i_remove, Product);

// Join with 'copy' func. Dynamic 'Product' fields will be duplicate.
arrst_join(products, new_products, i_copy_data, Product);
arrst_destroy(&new_products, i_remove, Product);
...
arrst_destroy(&products, i_remove, Product);
```

dest El array destino.

src El array cuyos elementos serán añadidos a `dest`.

func_copy Función de copia del objeto.

type Tipo de objeto.

Observaciones:

La función de copia debe crear memoria dinámica para los campos que lo requieran, pero NO para el objeto en sí mismo. Ver `arrst_copy`. Si es `NULL` se realizará una copia byte a byte del elemento.

arrst_delete

Elimina un elemento del array.

```
void
arrst_delete(ArrSt(type) *array,
             const uint32_t pos,
             FPtr_remove func_remove,
             type);
```

array	El array.
pos	Posición del elemento a borrar. El actual elemento en `pos+1` y siguientes serán desplazados a la izquierda.
func_remove	Función 'remove'. Ver `arrst_destroy`.
type	Tipo de objeto.

arrst_pop

Elimina el último elemento del array.

```
void
arrst_pop(ArrSt(type) *array,
          FPtr_remove func_remove,
          type);
```

array	El array.
func_remove	Función 'remove'. Ver `arrst_destroy`.
type	Tipo de objeto.

arrst_sort

Ordena los elementos del array utilizando Quicksort.

```
void
arrst_sort(ArrSt(type) *array,
           FPtr_compare func_compare,
           type);
```

array	El array.
func_compare	Función para comparar dos elementos. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
type	Tipo de objeto.

arrst_sort_ex

Ordena los elementos del array utilizando Quicksort y datos adicionales.

```
void
arrst_sort_ex(ArrSt(type) *array,
              FPtr_compare_ex func_compare,
              type,
              dtype);
```

array	El array.
func_compare	Función para comparar dos elementos utilizando un dato adicional.
type	Tipo de objeto.
dtype	Tipo de dato en la función de comparación.

arrst_search

Busca un elemento en el array de forma lineal `O(n)`.

```
type*
arrst_search(ArrSt(type) *array,
             FPtr_compare func_compare,
             const ktype *key,
             uint32_t *pos,
             type,
             ktype);
```

```
uint32_t pos;
uint32_t key = 345;
Product *product = arrst_search(arrst, i_compare_key, &key, &pos, Product,
    ↪ uint32_t);
```

array	El array.
func_compare	Función de comparación. El primer parámetro es el elemento, el segundo la clave de búsqueda. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
key	Clave a buscar. Puntero a un tipo de dato que puede ser diferente al tipo de elemento del array.
pos	Posición del elemento en el array (si existe), o `UINT32_MAX` si no existe. Puede ser `NULL`.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al primer elemento que coincida con el criterio de búsqueda o `NULL` si no existe ninguno.

arrst_search_const

Versión **const** de `arrst_search`.

```
const type*
arrst_search_const(const ArrSt(type) *array,
                   FPtr_compare func_compare,
                   const ktype *key,
                   uint32_t *pos,
                   type,
                   ktype);
```

array	El array.
func_compare	Función de comparación.
key	Clave a buscar. Puntero a un tipo de dato que puede ser diferente al tipo de elemento del array.
pos	Posición del elemento en el array.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al elemento.

arrst_bsearch

Busca un elemento en el array de forma logarítmica `O(logn)`.

```
type*
arrst_bsearch(ArrSt(type) *array,
              FPtr_compare func_compare,
              const ktype *key,
              uint32_t *pos,
              type,
              ktype);
```

array	El array.
func_compare	Función de comparación. El primer parámetro es el elemento, el segundo la clave de búsqueda. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
key	Clave a buscar. Puntero a un tipo de dato que puede ser diferente al tipo de elemento del array.
pos	Posición del elemento en el array (si existe), o `UINT32_MAX` si no existe. Puede ser `NULL`.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al primer elemento que coincida con el criterio de búsqueda o `NULL` si no existe ninguno.

Observaciones:

El array debe estar ordenado según el mismo criterio que la búsqueda. De no ser así el resultado es impredecible.

arrst_bsearch_const

Versión **const** de `arrst_bsearch`.

```
const type*
arrst_bsearch_const(const ArrSt(type) *array,
                    FPtr_compare func_compare,
                    const ktype *key,
                    uint32_t *pos,
                    type,
                    ktype);
```

array	El array.
func_compare	Función de comparación.
key	Clave a buscar.
pos	Posición del elemento en el array.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al elemento.

arrst_foreach

Itera sobre todos los elementos del array. Usa `arrst_end` para cerrar el bucle.

```
void
arrst_foreach(type *elem,
              ArrSt(type) *array,
              type);
```

```
arrst_foreach(product, array, Product)
    bstd_printf("Index: %d, Id: %d\n", product_i, product->id);
arrst_end()
```

elem	Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.
array	El array.
type	Tipo de objeto.

arrst_foreach_const

Versión **const** de `arrst_foreach`.

```
void
arrst_foreach_const(const type *elem,
                    const ArrSt(type) *array,
                    type);
```

elem	Elemento.
array	El array.
type	Tipo de objeto.

arrst_forback

Itera sobre todos los elementos del array hacia atrás, desde el último al primero. Usa `arrst_end` para cerrar el bucle.

```
void
arrst_forback(type *elem,
              ArrSt(type) *array,
              type);
```

```
// Now in reverse order
arrst_forback(product, array, Product)
    bstd_printf("Index: %d, Id: %d\n", product_i, product->id);
arrst_end()
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.

array El array.

type Tipo de objeto.

arrst_forback_const

Versión **const** de `arrst_forback`.

```
void
arrst_forback_const(const type *elem,
                    const ArrSt(type) *array,
                    type);
```

elem Elemento.

array El array.

type Tipo de objeto.

arrst_end

Cierra el bucle abierto por `arrst_foreach`, `arrst_foreach_const`, `arrst_forback` o `arrst_forback_const`.

```
void
arrst_end(void);
```

arrpt_create

Crea un array de punteros vacío.

```
ArrPt(type)*
arrpt_create(type);
```

type Tipo de objeto.

Retorna:

El nuevo array.

arrpt_copy

Crea una copia de un array de punteros.

```
ArrPt(type)*
arrpt_copy(const ArrPt(type) *array,
           FPtr_copy func_copy,
           type);
```

array El array original.

func_copy Función de copia del objeto.

type Tipo de objeto.

Retorna:

La copia del array original.

Observaciones:

La función de copia debe crear un objeto dinámico y asignar memoria para los campos internos que lo requieran. Si pasamos `NULL`, se realizará una copia de los punteros originales, lo que puede suponer un riesgo de integridad ya que un mismo objeto puede ser destruido dos veces si no ponemos especial cuidado. Ver *"Copia de objetosCopia de objetos"* (Página 218).

arrpt_read

Crea un array leyendo su contenido de un *"Streams"* (Página 197) (de-serialización).

```
ArrPt(type)*
arrpt_read(Stream *stream,
           FPtr_read func_read,
           type);
```

stream Un *stream* de lectura.

func_read Constructor para crear un objeto a partir de los datos obtenidos de un stream. *"SerializaciónSerialización"* (Página 218).

type Tipo de objeto.

Retorna:

El array leído.

arrpt_destroy

Destruye un array y todos sus elementos.

```
void
arrpt_destroy(ArrPt(type) **array,
              FPtr_destroy func_destroy,
              type);
```

array El array. Será puesto a NULL tras la destrucción.

func_destroy Función para destruir un elemento del array *"DestructoresDestructores"* (Página 219). Si es NULL se destruirá solo el array, pero no sus elementos.

type Tipo de objeto.

arrpt_destopt

Destruye un array y todos sus elementos, siempre y cuando el objeto array no sea NULL.

```
void
arrpt_destopt(ArrSt(type) **array,
              FPtr_destroy func_destroy,
              type);
```

array El array.

func_destroy Ver arrpt_destroy.

type Tipo de objeto.

arrpt_clear

Borra el contenido del array, sin destruir el contenedor que quedará con cero elementos.

```
void
arrpt_clear(ArrPt(type) *array,
```

```
            FPtr_destroy func_destroy,
            type);
```

array	El array.
func_destroy	Destructor del elemento. Puede ser `NULL`. Ver `arrpt_destroy`.
type	Tipo de objeto.

arrpt_write

Escribe un array en un *"Streams"* (Página 197) (serialización).

```
void
arrpt_write(Stream *stream,
            const ArrPt(type) *array,
            FPtr_write func_write,
            type);
```

stream	Un *stream* de escritura.
array	El array.
func_write	Función que escribe el contenido de un elemento en un stream *"SerializaciónSerialización"* (Página 218).
type	Tipo de objeto.

arrpt_size

Obtiene el número de elementos en un array.

```
uint32_t
arrpt_size(const ArrPt(type) *array,
           type);
```

array	El array.
type	Tipo de objeto.

Retorna:

Número de elementos.

arrpt_get

Obtiene un puntero al elemento en la posición `pos`.

```
type*
arrpt_get(ArrPt(type) *array,
          const uint32_t pos,
          type);
```

array El array.

pos Posición o índice del elemento.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_get_const

Obtiene un puntero **const** al elemento en la posición `pos`.

```
const type*
arrpt_get_const(const ArrPt(type) *array,
                const uint32_t pos,
                type);
```

array El array.

pos Posición o índice del elemento.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_first

Obtiene un puntero al primer elemento del array.

```
type*
arrpt_first(ArrPt(type) *array,
            type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_first_const

Obtiene un puntero **const** al primer elemento del array.

```
const type*
arrpt_first_const(const ArrPt(type) *array,
                  type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_last

Obtiene un puntero al último elemento del array.

```
type*
arrpt_last(ArrPt(type) *array,
           type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_last_const

Obtiene un puntero **const** al último elemento del array.

```
const type*
arrpt_last_const(const ArrPt(type) *array,
                 type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero al elemento.

arrpt_all

Obtiene un puntero a la memoria interna del array, que da acceso a todos los elementos.

```
type**
arrpt_all(ArrPt(type) *array,
          type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero base. Incrementándolo uno a uno iteraremos sobre los elementos.

Observaciones:

Utiliza `arrpt_foreach` para iterar sobre los elementos de forma más segura y elegante.

arrpt_all_const

Obtiene un puntero **const** a la memoria interna del array, que da acceso a todos los elementos.

```
const type**
arrpt_all_const(const ArrPt(type) *array,
                type);
```

array El array.

type Tipo de objeto.

Retorna:

Puntero base. Incrementándolo uno a uno iteraremos sobre los elementos.

Observaciones:

Utiliza `arrpt_foreach_const` para iterar sobre los elementos de forma más segura y elegante.

arrpt_grow

Añade `n` elementos, sin inicializar, al final del array.

```
type**
arrpt_grow(ArrPt(type) *array,
           const uint32_t n,
           type);
```

array El array.

n Cantidad de elementos a añadir.

type Tipo de objeto.

Retorna:

Puntero al primer elemento añadido.

arrpt_append

Añade un puntero al final del array.

```
void
arrpt_append(ArrPt(type) *array,
             type *value,
             type);
```

array El array.

value Puntero al elemento a añadir.

type Tipo de objeto.

arrpt_prepend

Inserta un puntero al inicio del array. El resto de elementos serán desplazados a la derecha.

```
void
arrpt_prepend(ArrPt(type) *array,
              type *value,
              type);
```

array El array.

value Puntero al elemento a insertar.

type Tipo de objeto.

arrpt_insert

Inserta un puntero en una posición arbitraria del array.

```
void
arrpt_insert(ArrPt(type) *array,
             const uint32_t pos,
             type *value,
             type);
```

array El array.

pos Posición donde será insertado. El actual elemento en `pos` y siguientes serán desplazados a la derecha.

value Puntero al elemento a insertar.

type Tipo de objeto.

arrpt_join

Une dos vectores. Añade todos los elementos de `src` al final de `dest`.

```
void
arrpt_join(ArrPt(type) *dest,
           const ArrPt(type) *src,
           FPtr_copy func_copy,
           type);
```

```
ArrPt(Product) *products = create_products(...);
ArrPt(Product) *new_products = new_products(...);

// Join without 'copy' func. Dynamic 'Product' objects will be reused.
arrpt_join(products, new_products, NULL, Product);
arrpt_destroy(&new_products, NULL, Product);
...
arrpt_destroy(&products, i_destroy, Product);

// Join with 'copy' func. Dynamic 'Product' objects will be duplicate.
arrpt_join(products, new_products, i_copy, Product);
arrpt_destroy(&new_products, i_destroy, Product);
...
arrpt_destroy(&products, i_destroy, Product);
```

dest El array destino.

src El array cuyos elementos serán añadidos a `dest`.

func_copy Función de copia del objeto.

type Tipo de objeto.

Observaciones:

La función de copia debe crear memoria dinámica tanto para el objeto como para los campos que lo requieran. Si es `NULL` se solo se añadirá una copia del puntero original a `dest`.

arrpt_delete

Elimina un puntero del array.

```
void
arrpt_delete(ArrPt(type) *array,
             const uint32_t pos,
             FPtr_destroy func_destroy,
             type);
```

array	El array.
pos	Posición del elemento a borrar. El actual elemento en `pos+1` y siguientes serán desplazados a la izquierda.
func_destroy	Destructor del elemento. Puede ser `NULL`. Ver `arrpt_destroy`.
type	Tipo de objeto.

arrpt_pop

Elimina el último puntero del array.

```
void
arrpt_pop(ArrPt(type) *array,
          FPtr_destroy func_destroy,
          type);
```

array	El array.
func_destroy	Destructor del elemento. Puede ser `NULL`. Ver `arrpt_destroy`.
type	Tipo de objeto.

arrpt_sort

Ordena los elementos del array utilizando Quicksort.

```
void
arrpt_sort(ArrPt(type) *array,
           FPtr_compare func_compare,
           type);
```

array	El array.
func_compare	Función para comparar dos elementos. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
type	Tipo de objeto.

arrpt_sort_ex

Ordena los elementos del array utilizando Quicksort y datos adicionales.

```
void
arrpt_sort_ex(ArrPt(type) *array,
              FPtr_compare_ex func_compare,
              type,
              dtype);
```

array	El array.
func_compare	Función para comparar dos elementos utilizando un dato adicional.
type	Tipo de objeto.
dtype	Tipo de dato en la función de comparación.

arrpt_find

Busca un determinado puntero en el array.

```
uint32_t
arrpt_find(const ArrPt(type) *array,
           type *elem,
           type);
```

array	El array.
elem	Puntero a buscar.
type	Tipo de objeto.

Retorna:

La posición del puntero si existe, o `UINT32_MAX` si no.

arrpt_search

Busca un elemento en el array de forma lineal `O(n)`.

```
type*
arrpt_search(ArrPt(type) *array,
             FPtr_compare func_compare,
             ktype key,
             uint32_t *pos,
             type,
             ktype);
```

array	El array.
func_compare	Función de comparación. El primer parámetro es el elemento, el segundo la clave de búsqueda. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
key	Clave a buscar. Puntero a un tipo de dato que puede ser diferente al tipo de elemento del array.
pos	Posición del elemento en el array (si existe), o `UINT32_MAX` si no existe. Puede ser `NULL`.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al primer elemento que coincida con el criterio de búsqueda o `NULL` si no existe ninguno.

arrpt_search_const

Versión **const** de `arrpt_search`.

```
const type*
arrpt_search_const(const ArrPt(type) *array,
                   FPtr_compare func_compare,
                   const ktype *key,
                   uint32_t *pos,
                   type,
                   ktype);
```

array	El array.
func_compare	Función de comparación.
key	Clave a buscar.
pos	Posición del elemento en el array.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Elemento.

arrpt_bsearch

Busca un elemento en el array de forma logarítmica `O(logn)`.

```
type*
arrpt_bsearch(ArrPt(type) *array,
              FPtr_compare func_compare,
              ktype key,
              uint32_t *pos,
              type,
              ktype);
```

array	El array.
func_compare	Función de comparación. El primer parámetro es el elemento, el segundo la clave de búsqueda. *"Ordenar y buscarOrdenar y buscar"* (Página 221).
key	Clave a buscar. Puntero a un tipo de dato que puede ser diferente al tipo de elemento del array.
pos	Posición del elemento en el array (si existe), o `UINT32_MAX` si no existe. Puede ser `NULL`.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Puntero al primer elemento que coincida con el criterio de búsqueda o `NULL` si no existe ninguno.

Observaciones:

El array debe estar ordenado según el mismo criterio que la búsqueda. De no ser así el resultado es impredecible.

arrpt_bsearch_const

Versión **const** de `arrpt_bsearch`.

```
const type*
arrpt_bsearch_const(const ArrPt(type) *array,
                    FPtr_compare func_compare,
                    const ktype *key,
                    uint32_t *pos,
                    type,
                    ktype);
```

array	El array.
func_compare	Función de comparación.
key	Clave a buscar.
pos	Posición del elemento en el array.
type	Tipo de objeto.
ktype	Tipo de clave.

Retorna:

Elemento.

arrpt_foreach

Itera sobre todos los elementos del array. Usa `arrpt_end` para cerrar el bucle.

```
void
arrpt_foreach(type *elem,
              ArrPt(type) *array,
              type);
```

```
arrpt_foreach(product, array, Product)
    bstd_printf("Index: %d, Id: %d\n", product_i, product->id);
arrpt_end()
```

elem	Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.
array	El array.
type	Tipo de objeto.

arrpt_foreach_const

Versión **const** de `arrpt_foreach`.

```
void
arrpt_foreach_const(const type *elem,
                    const ArrPt(type) *array,
                    type);
```

elem	Elemento.
array	El array.
type	Tipo de objeto.

arrpt_forback

Itera sobre todos los elementos del array hacia atrás, desde el último al primero. Usa `arrpt_end` para cerrar el bucle.

```
void
arrpt_forback(type *elem,
              ArrPt(type) *array,
              type);
```

```
// Now in reverse order
arrpt_forback(product, array, Product)
    bstd_printf("Index: %d, Id: %d\n", product_i, product->id);
arrpt_end()
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.

array El array.

type Tipo de objeto.

arrpt_forback_const

Versión `const` de `arrpt_forback`.

```
void
arrpt_forback_const(const type *elem,
                    const ArrPt(type) *array,
                    type);
```

elem Elemento.

array El array.

type Tipo de objeto.

arrpt_end

Cierra el bucle abierto por `arrpt_foreach`, `arrpt_foreach_const`, `arrpt_forback` o `arrpt_forback_const`.

```
void
arrpt_end(void);
```

setst_create

Crea un set de registros vacío.

```
SetSt(type)*
setst_create(FPtr_compare func_compare,
             type);
```

func_compare — Función para comparar dos elementos. *"Ordenar y buscarOrdenar y buscar"* (Página 221).

type — Tipo de objeto.

Retorna:

El nuevo set.

setst_destroy

Destruye un set y todos sus elementos.

```
void
setst_destroy(SetSt(type) **set,
              FPtr_remove func_remove,
              type);
```

set — El set. Será puesto a `NULL` tras la destrucción.

func_remove — Función que debe liberar la memoria asociada a los campos del objeto, pero no el objeto en si mismo *"DestructoresDestructores"* (Página 219). Si es `NULL` se liberará solo el set y no el contenido interno de los elementos.

type — Tipo de objeto.

setst_size

Obtiene el número de elementos del set.

```
uint32_t
setst_size(const SetSt(type) *set,
           type);
```

set — El set.

type — Tipo de objeto.

Retorna:

Número de elementos.

setst_get

Busca un elemento en `O(logn)`. Es equivalente a `arrst_bsearch`. Si existe, el iterador interno de la estructura quedará fijado en él.

```
type*
setst_get(SetSt(type) *set,
          const type *key,
          type);
```

```
Product key;
Product *pr;
key.id = 453;
pr = setst_get(setst, &key, Product);
```

set El set.

key Clave a buscar. Es un puntero a un objeto donde solo los campos relevantes de la búsqueda deben ser inicializados.

type Tipo de objeto.

Retorna:

Puntero al elemento si existe, o `NULL` si no.

Observaciones:

"IteradoresIteradores" (Página 225).

setst_get_const

Versión **const** de `setst_get`.

```
const type*
setst_get_const(const SetSt(type) *set,
                const type *key,
                type);
```

set El set.

key Clave a buscar.

type Tipo de objeto.

Retorna:

Elemento.

setst_insert

Inserta un nuevo elemento en el set.

```
type*
setst_insert(SetSt(type) *set,
             type *key,
             type);
```

```
Product *pr;
Product key;
key.id = 345;
pr = setst_insert(setst, &key, Product);
if (pr != NULL)
    i_init(pr, 345, 100.45f);
else
    error("Already exists");
```

set	El set.
key	Clave a insertar. Es un puntero a un objeto donde solo los campos relevantes de la búsqueda deben ser inicializados.
type	Tipo de objeto.

Retorna:

Puntero al elemento insertado, que debe utilizarse para inicializar el objeto. Si ya existe un elemento con la misma clave, retorna `NULL`.

Observaciones:

Insertar o eliminar elementos invalida el iterador interno del set *"IteradoresIteradores"* (Página 225). Debes volver a inicializarlo con `setst_first`.

setst_delete

Elimina un elemento del set.

```
bool_t
setst_delete(SetSt(type) *set,
             type *key,
             FPtr_remove func_remove,
             type);
```

```
Product key;
key.id = 345;
if (setst_delete(setst, &key, product_remove, Product) == FALSE)
    error("Doesn't exists");
```

set	El set.
key	Clave a eliminar. Es un puntero a un objeto donde solo los campos relevantes de la búsqueda deben ser inicializados.
func_remove	Función 'remove'. Ver `setst_destroy`.
type	Tipo de objeto.

Retorna:

`TRUE` si el elemento ha sido eliminado, o `FALSE` si no existe un elemento con dicha clave.

Observaciones:

Insertar o eliminar elementos invalida el iterador interno del set *"IteradoresIteradores"* (Página 225). Debes inicializarlo con `setst_first`.

setst_first

Obtiene el primer elemento del set e inicializa el iterador interno.

```
type*
setst_first(SetSt(type) *set,
            type);
```

set	El set.
type	Tipo de objeto.

Retorna:

Puntero al primer elemento o `NULL` si el set está vacío.

Observaciones:

"IteradoresIteradores" (Página 225).

setst_first_const

Versión **const** de `setst_first`.

```
const type*
setst_first_const(const SetSt(type) *set,
                  type);
```

set	El set.
type	Tipo de objeto.

Retorna:

Elemento.

setst_last

Obtiene el último elemento del set e inicializa el iterador interno.

```
type*
setst_last(SetSt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al último elemento o `NULL` si el set está vacío.

Observaciones:

"IteradoresIteradores" (Página 225).

setst_last_const

Versión **const** de `setst_last`.

```
const type*
setst_last_const(const SetSt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setst_next

Obtiene el siguiente elemento del set, tras incrementar el iterador interno.

```
type*
setst_next(SetSt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al siguiente elemento o `NULL` si el iterador ha alcanzado el último.

Observaciones:

Utiliza `setst_first` para inicializar el iterador interno *"IteradoresIteradores"* (Página 225).

setst_next_const

Versión **const** de `setst_next`.

```
const type*
setst_next_const(const SetSt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setst_prev

Obtiene el elemento anterior del set, tras decrementar el iterador interno.

```
type*
setst_prev(SetSt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al elemento anterior o `NULL` si el iterador ha alcanzado el primero.

Observaciones:

Utiliza `setst_last` para inicializar el iterador interno en bucles revertidos *"IteradoresIteradores"* (Página 225).

setst_prev_const

Versión **const** de `setst_prev`.

```
const type*
setst_prev_const(const SetSt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setst_foreach

Recorre todos los elementos del set. Utiliza `setst_fornext` para cerrar el bucle.

```
void
setst_foreach(type *elem,
              SetSt(type) *set,
              type);
```

```
setst_foreach(product, set, Product)
    bstd_printf("Position: %d, Id: %d\n", product_i, product->id);
setst_fornext(product, set, Product)
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.

set El set.

type Tipo de objeto.

setst_foreach_const

Versión **const** de `setst_foreach`.

```
void
setst_foreach_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setst_fornext

Cierra el bucle abierto por `setst_foreach`, incrementando el iterador interno.

```
void
setst_fornext(type *elem,
              SetSt(type) *set,
              type);
```

elem Nombre de la variable 'elemento'. Debe ser el mismo que `setst_foreach` .

set El set.

type Tipo de objeto.

setst_fornext_const

Versión **const** de `setst_fornext`.

```
void
setst_fornext_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setst_forback

Recorre todos los elementos del set en orden inverso. Utiliza `setst_forprev` para cerrar el bucle.

```
void
setst_forback(type *elem,
              SetSt(type) *set,
              type);
```

```
// Now in reverse order
setst_forback(product, set, Product)
    bstd_printf("Position: %d, Id: %d\n", product_i, product->id);
setst_forprev(product, set, Product)
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.

set El set.

type Tipo de objeto.

setst_forback_const

Versión **const** de `setst_forback`.

```
void
setst_forback_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setst_forprev

Cierra el bucle abierto por `setst_forback`, decrementando el iterador interno.

```
void
setst_forprev(type *elem,
              SetSt(type) *set,
              type);
```

elem Nombre de la variable 'elemento'. Debe ser el mismo que `setst_forback` .

set El set.

type Tipo de objeto.

setst_forprev_const

Versión **const** de `setst_forprev`.

```
void
setst_forprev_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setpt_create

Crea un set de punteros vacío.

```
SetPt(type)*
setpt_create(FPtr_compare func_compare,
             type);
```

func_compare Función para comparar dos elementos. *"Ordenar y buscarOrdenar y buscar"* (Página 221).

type Tipo de objeto.

Retorna:

El nuevo set.

setpt_destroy

Destruye un set y todos sus elementos.

```
void
setpt_destroy(SetPt(type) **set,
              FPtr_destroy func_destroy,
              type);
```

set El set. Será puesto a `NULL` tras la destrucción.

func_destroy Función para destruir un elemento del set *"DestructoresDestructores"* (Página 219). Si es `NULL` se destruirá solo el set, pero no sus elementos.

type Tipo de objeto.

setpt_size

Obtiene el número de elementos del set.

```
uint32_t
setpt_size(const SetPt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Número de elementos.

setpt_get

Busca un elemento en `O(logn)`. Es equivalente a `arrpt_bsearch`. El iterador interno de la estructura quedará fijado en él.

```
type*
setpt_get(SetPt(type) *set,
          type *key,
          type);
```

```
Product key;
Product *pr;
key.id = 453;
pr = setpt_get(setpt, &key, Product);
```

set El set.

key Clave a buscar. Es un puntero a un objeto donde solo los campos relevantes de la búsqueda deben ser inicializados.

type Tipo de objeto.

Retorna:

Puntero al elemento buscado si existe, o `NULL` si no.

Observaciones:

"IteradoresIteradores" (Página 225).

setpt_get_const

Versión **const** de `setpt_get`.

```
const type*
setpt_get_const(const SetPt(type) *set,
                const type *key,
                type);
```

set	El set.
key	Clave a buscar.
type	Tipo de objeto.

Retorna:

Elemento.

setpt_insert

Inserta un nuevo elemento en el set.

```
bool_t
setpt_insert(SetPt(type) *set,
             type *value,
             type);
```

```
Product *pr = product_create(...);
if (setpt_insert(setpt, pr, Product) == FALSE)
{
    error("Already exists");
    product_destroy(&pr);
}
```

set	El set.
value	Puntero al elemento a insertar.
type	Tipo de objeto.

Retorna:

`TRUE` si el elemento ha sido insertado. `FALSE` si ya existe otro elemento con la misma clave.

Observaciones:

Insertar o eliminar elementos invalida el iterador interno del set *"IteradoresIteradores"* (Página 225). Debes volver a inicializarlo con `setpt_first`.

setpt_delete

Elimina un elemento del set.

```
bool_t
setpt_delete(SetPt(type) *set,
             type *key,
             FPtr_destroy func_destroy,
```

```
            type);
```

```
Product key;
key.id = 345;
if (setpt_delete(setpt, &key, product_destroy, Product) == FALSE)
    error("Doesn't exists");
```

set	El set.
key	Clave a eliminar. Es un puntero a un objeto donde solo los campos relevantes de la búsqueda deben ser inicializados.
func_destroy	Destructor del elemento. Puede ser `NULL`. Ver `setpt_destroy`.
type	Tipo de objeto.

Retorna:

`TRUE` si el elemento ha sido eliminado, o `FALSE` si no existe un elemento con dicha clave.

Observaciones:

Insertar o eliminar elementos invalida el iterador interno del set *"IteradoresIteradores"* (Página 225). Debes inicializarlo con `setpt_first`.

setpt_first

Obtiene el primer elemento del set e inicializa el iterador interno.

```
type*
setpt_first(SetPt(type) *set,
            type);
```

set	El set.
type	Tipo de objeto.

Retorna:

Puntero al primer elemento o `NULL` si el set está vacío.

Observaciones:

"IteradoresIteradores" (Página 225).

setpt_first_const

Versión **const** de `setpt_first`.

```
const type*
setpt_first_const(const SetPt(type) *set,
                  type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setpt_last

Obtiene el último elemento del set e inicializa el iterador interno.

```
type*
setpt_last(SetPt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al último elemento o `NULL` si el set está vacío.

Observaciones:

"IteradoresIteradores" (Página 225).

setpt_last_const

Versión **const** de `setpt_last`.

```
const type*
setpt_last_const(const SetPt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setpt_next

Obtiene el siguiente elemento del set, tras incrementar el iterador interno.

```
type*
setpt_next(SetPt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al siguiente elemento o `NULL` si el iterador ha alcanzado el último.

Observaciones:

Utiliza `setpt_first` para inicializar el iterador interno *"IteradoresIteradores"* (Página 225).

setpt_next_const

Versión **const** de `setpt_next`.

```
const type*
setpt_next_const(const SetPt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setpt_prev

Obtiene el elemento anterior del set, tras decrementar el iterador interno.

```
type*
setpt_prev(SetPt(type) *set,
           type);
```

set El set.

type Tipo de objeto.

Retorna:

Puntero al elemento anterior o NULL si el iterador ha alcanzado el primero.

Observaciones:

Utiliza setpt_last para inicializar el iterador interno en recorridos revertidos *"IteradoresIteradores"* (Página 225).

setpt_prev_const

Versión **const** de setpt_prev.

```
const type*
setpt_prev_const(const SetPt(type) *set,
                 type);
```

set El set.

type Tipo de objeto.

Retorna:

Elemento.

setpt_foreach

Recorre todos los elementos del set. Utiliza setpt_fornext para cerrar el bucle.

```
void
setpt_foreach(type *elem,
              SetPt(type) *set,
              type);
```

```
setpt_foreach(product, set, Product)
    bstd_printf("Position: %d, Id: %d\n", product_i, product->id);
setpt_fornext(product, set, Product)
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo '_i' obtenemos el índice.

set El set.

type Tipo de objeto.

setpt_foreach_const

Versión **const** de setpt_foreach.

```
void
setpt_foreach_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setpt_fornext

Cierra el bucle abierto por `setpt_foreach`, incrementando el iterador interno.

```
void
setpt_fornext(type *elem,
              SetPt(type) *set,
              type);
```

elem Nombre de la variable 'elemento'. Debe ser el mismo que `setpt_foreach` .

set El set.

type Tipo de objeto.

setpt_fornext_const

Versión **const** de `setpt_fornext`.

```
void
setpt_fornext_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setpt_forback

Recorre todos los elementos del set en orden inverso. Utiliza `setpt_forprev` para cerrar el bucle.

```
void
setpt_forback(type *elem,
```

```
                SetPt(type) *set,
                type);
```

```
// Now in reverse order
setpt_forback(product, set, Product)
    bstd_printf("Position: %d, Id: %d\n", product_i, product->id);
setpt_forprev(product, set, Product)
```

elem Nombre de la variable 'elemento' dentro del bucle. Añadiendo el sufijo `'_i'` obtenemos el índice.

set El set.

type Tipo de objeto.

setpt_forback_const

Versión **const** de `setpt_forback`.

```
void
setpt_forback_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

setpt_forprev

Cierra el bucle abierto por `setpt_forback`, decrementando el iterador interno.

```
void
setpt_forprev(type *elem,
              SetPt(type) *set,
              type);
```

elem Nombre de la variable 'elemento'. Debe ser el mismo que `setpt_forback` .

set El set.

type Tipo de objeto.

setpt_forprev_const

Versión **const** de `setpt_forprev`.

```
void
setpt_forprev_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Elemento.

set El set.

type Tipo de objeto.

regex_create

Crea una expresión regular a partir de un patrón.

```
RegEx*
regex_create(const char_t *pattern);
```

pattern Patrón de búsqueda.

Retorna:

Expresión regular (autómata).

Observaciones:

Ver *"Definir patronesDefinir patrones"* (Página 228).

regex_destroy

Destruye una expresión regular.

```
void
regex_destroy(RegEx **regex);
```

regex Expresión regular. Será puesto a `NULL` tras la destrucción.

regex_match

Comprueba si una cadena cumple el patrón de búsqueda.

```
bool_t
regex_match(const RegEx *regex,
            const char_t *str);
```

regex Expresión regular.

str Cadena a evaluar.

Retorna:

TRUE si la cadena es aceptada por la expresión regular.

dbind

Añade un campo de una estructura/clase a su tabla interna dentro de dbind.

```
void
dbind(type,
      mtype,
      name);
```

type	Tipo de la estructura o clase.
mtype	Tipo del campo a registrar.
name	Nombre del campo dentro de la estructura.

Observaciones:

Se generarán errores en tiempo de compilación si el campo indicado no pertenece a la estructura. El método también sirve para clases en C++.

dbind_enum

Registra un valor tipo enum.

```
void
dbind_enum(type,
           value,
           const char_t *alias);
```

type	Tipo del enum.
value	Valor.
alias	Alias para el valor.

Observaciones:

dbind_enum(mode_t, ekIMAGE_ANALISYS, "Image Analisys"); utilizará la cadena “Image Analisys” en lugar de “ekIMAGE_ANALISYS” para aquellas operaciones E/S o de interfaz que requieran mostrar los literales del enumerado. Por ejemplo, para rellenar los campos de un PopUp vinculado con un campo de datos.

dbind_create

Crea un objeto de tipo registrado, inicializando sus campos con los valores por defecto.

```
type*
dbind_create(type);
```

type Tipo de objeto.

Retorna:

Objeto recién creado o `NULL` si *dbind* no reconoce el tipo de dato.

dbind_init

Inicializa los campos de un objeto de tipo registrado con los valores por defecto.

```
void
dbind_init(type *obj,
           type);
```

obj Objeto cuya memoria ha sido reservada, pero no inicializada.

type Tipo de objeto.

dbind_remove

Destruye la memoria reservada por los campos de un objeto de tipo registrado, pero no destruye el objeto en sí.

```
void
dbind_remove(type *obj,
             type);
```

obj Objeto.

type Tipo de objeto.

dbind_destroy

Destruye un objeto de tipo registrado. La memoria asignada a los campos y sub-objetos también será liberada de forma recursiva.

```
void
dbind_destroy(type **obj,
              type);
```

obj Objeto. Será puesto a `NULL` tras la destrucción.

type Tipo de objeto.

dbind_destopt

Destructor opcional. Igual que dbind_destroy, pero aceptando que el objeto sea NULL.

```
void
dbind_destopt(type **obj,
              type);
```

obj Objeto a destruir.

type Tipo de objeto.

dbind_read

Crea un objeto de tipo registrado a partir de los datos leídos desde un stream.

```
type*
dbind_read(Stream *stm,
           type);
```

stm Stream de lectura.

type Tipo del objeto a leer.

Retorna:

Objeto recién creado o NULL si ha habido algún error.

dbind_write

Escribe el contenido de un objeto de tipo registrado en un stream de escritura.

```
void
dbind_write(Stream *stm,
            const type *data,
            type);
```

stm Stream de escritura.

data Objeto a escribir.

type Tipo del objeto a escribir.

dbind_default

Establece el valor por defecto de un campo.

```
void
dbind_default(type,
              mtype,
```

```
                name,
                mtype value);
```

type	Tipo de la estructura o clase.
mtype	Tipo del campo.
name	Nombre del campo dentro de la estructura.
value	Valor por defecto a partir de ahora.

dbind_range

Establece el valor máximo y mínimo en campos numéricos.

```
void
dbind_range(type,
            mtype,
            name,
            mtype min,
            mtype max);
```

type	Tipo de la estructura o clase.
mtype	Tipo del campo.
name	Nombre del campo dentro de la estructura.
min	Valor mínimo.
max	Valor máximo.

Observaciones:

Dará error si se utiliza en campos no numéricos.

dbind_precision

Establece el salto entre dos valores numéricos consecutivos.

```
void
dbind_precision(type,
                mtype,
                name,
                mtype prec);
```

type Tipo de la estructura o clase.

mtype Tipo del campo.

name Nombre del campo dentro de la estructura.

prec Precisión (p.e. `.05f` en valores `real32_t`).

Observaciones:

Dará error si se utiliza en campos no numéricos.

dbind_increment

Establece el incremento de un valor numérico al pulsar en un control *"UpDown"* (Página 317).

```
void
dbind_increment(type,
                mtype,
                name,
                mtype incr);
```

type Tipo de la estructura o clase.

mtype Tipo del campo.

name Nombre del campo dentro de la estructura.

incr Incremento.

Observaciones:

Dará error si se utiliza en campos no numéricos.

dbind_suffix

Establece un sufijo que será añadido al valor numérico al convertirlo a texto.

```
void
dbind_suffix(type,
             mtype,
             name,
             const char_t *suffix);
```

type Tipo de la estructura o clase.

mtype Tipo del campo.

name Nombre del campo dentro de la estructura.

suffix Sufijo.

Observaciones:

Dará error si se utiliza en campos no numéricos.

listener

Crea un listener. Esta función vinculará un emisor de eventos con el receptor, normalmente el controlador de la aplicación. El objeto emisor *(sender)* es el responsable de destruir el listener.

```
Listener*
listener(type *obj,
         FPtr_event_handler func_event_handler,
         type);
```

obj	Objeto receptor que será pasado como primer parámetro a `func_event_handler`.
func_event_handler	Función *callback* que será llamada cuando se produzca el evento. También conocida como *event handler*.
type	El tipo de objeto receptor.

Retorna:

Objeto listener.

listen

Igual que `listener`, pero utilizado en C++ para definir *callbacks* de clase. *"Uso de C++"* (Página 45).

```
void
listen(void);
```

listener_destroy

Destruye un listener.

```
void
listener_destroy(Listener **listener);
```

listener	Listener. Será puesto a `NULL` tras la destrucción.

Observaciones:

El emisor es el responsable de destruir el listener.

listener_update

Actualiza el receptor y manejador del evento. Es equivalente a destruirlo crearlo de nuevo.

```
void
listener_update(Listener **listener,
                Listener *new_listener);
```

listener	El listener actual.
new_listener	El nuevo listener.

Observaciones:

Este método debe ser utilizado dentro del emisor.

listener_event

Lanza un evento desde el emisor *(sender)* hacia el receptor.

```
void
listener_event(Listener *listener,
               const uint32_t type,
               sender_type *sender,
               params_type *params,
               result_type *result,
               sender_type,
               params_type,
               result_type);
```

listener	Listener a través del cual se enviará el evento.
type	Código del evento.
sender	Generador del evento *(sender)*.
params	Parámetros del evento, o `NULL` si no tiene.
result	Resultado del evento, o `NULL` si no se espera.
sender_type	Tipo de objeto *sender*.
params_type	Tipo de objeto params, o `void` si no tiene.
result_type	Tipo de objeto result, o `void` si no tiene.

Observaciones:

Este método debe ser invocado dentro del emisor del evento.

listener_pass_event

Pasa el evento recibido a otro objeto, cambiando solo el emisor *(sender)*. Útil para no generar un nuevo objeto `Event`.

```
void
listener_pass_event(Listener *list,
                    Event *event,
                    sender_type *sender,
                    sender_type);
```

list	Listener a través del cual se re-enviará el evento.
event	Evento entrante.
sender	El nuevo generador del evento.
sender_type	El tipo de objeto.

Observaciones:

Este método debe ser invocado dentro del emisor del evento.

event_type

Obtiene el tipo de evento.

```
uint32_t
event_type(const Event *event);
```

event	Evento.

Retorna:

El tipo de evento. Normalmente asociado a un `enum`. Ejemplos en `core_event_t`, `gui_event_t`.

event_sender

Obtiene el emisor del evento *(sender)*.

```
type*
event_sender(Event *event,
             type);
```

event	Evento.
type	El tipo del emisor.

Retorna:

Sender.

event_params

Obtiene los parámetros del evento, encapsulados en una estructura, que será diferente en función del tipo de evento.

```
type*
event_params(Event *event,
             type);
```

event Evento.

type El tipo de parámetros.

Retorna:

Parámetros del evento.

event_result

Obtiene un objeto para escribir los resultados del evento. Algunos eventos requieren la devolución de datos por parte del receptor. El tipo de objeto resultado dependerá del tipo de evento.

```
type*
event_result(Event *event,
             type);
```

event Evento.

type El tipo de resultado.

Retorna:

Resultados del evento.

keybuf_create

Crea un búfer con el estado del teclado.

```
KeyBuf*
keybuf_create(void);
```

Retorna:

El búfer.

keybuf_destroy

Destruye el búfer.

```
void
keybuf_destroy(KeyBuf **bufer);
```

bufer El búfer. Será puesto a `NULL` tras la destrucción.

keybuf_OnUp

Establece el estado de una tecla como liberada.

```
void
keybuf_OnUp(KeyBuf *bufer,
            const vkey_t key);
```

bufer El búfer.

key El código de la tecla.

Observaciones:

Normalmente no será necesario llamar a esta función. Lo hará `View` o el módulo que capture los eventos de teclado.

keybuf_OnDown

Establece el estado de una tecla como pulsado.

```
void
keybuf_OnDown(KeyBuf *bufer,
              const vkey_t key);
```

bufer El búfer.

key El código de la tecla.

Observaciones:

Normalmente no será necesario llamar a esta función. Lo hará `View` o el módulo que capture los eventos de teclado.

keybuf_clear

Limpia el búfer. Establece todas las teclas como liberadas.

```
void
keybuf_clear(KeyBuf *bufer);
```

bufer El búfer.

Observaciones:

Normalmente no será necesario llamar a esta función. Lo hará `View` o el módulo que capture los eventos de teclado.

keybuf_pressed

Retorna el estado de una tecla.

```
bool_t
keybuf_pressed(const KeyBuf *bufer,
               const vkey_t key);
```

bufer El búfer.

key El código de la tecla.

Retorna:

Pulsada (`TRUE`) o liberada (`FALSE`).

keybuf_str

Retorna una cadena de texto asociada a una tecla.

```
void
keybuf_str(const vkey_t key);
```

key El código de la tecla.

keybuf_dump

Vuelca en el *"Log"* (Página 186) el estado del búfer.

```
void
keybuf_dump(const KeyBuf *bufer);
```

bufer El búfer.

hfile_dir

Comprueba si la ruta es un directorio.

```
bool_t
hfile_dir(const char_t *pathname);
```

pathname Nombre de la ruta a comprobar. *"Filename y pathnameFilename y pathname"* (Página 181).

Retorna:

`TRUE` si `pathname` es un directorio. Si no existe o es un archivo `FALSE`.

hfile_dir_create

Crea todos los subdirectorios intermedios de una ruta.

```
bool_t
hfile_dir_create(const char_t *pathname,
                 ferror_t *error);
```

```
// C:\dir1 doesn't exist.
bool_t ok = hfile_dir_create("C:\dir1\dir2\dir3\dir4\dir5");
ok = TRUE
```

pathname Nombre de la ruta a crear. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si la ruta completa ha sido creada, de lo contrario `FALSE`.

hfile_dir_destroy

Destruye recursivamente un directorio y todo su contenido.

```
bool_t
hfile_dir_destroy(const char_t *pathname,
                  ferror_t *error);
```

pathname Ruta del directorio a destruir. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el directorio ha sido destruido, o `FALSE` si ha habido algún error.

hfile_dir_list

Obtiene una lista del contenido de un directorio.

```
ArrSt(DirEntry)*
hfile_dir_list(const char_t *pathname,
               ferror_t *error);
```

pathname Ruta del directorio a listar. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

Array de `DirEntry` con el contenido. Debe ser destruido con `arrst_destroy(&array, hfile_dir_entry_remove, DirEntry)` cuando ya no sea necesario.

hfile_dir_entry_remove

Libera la memoria de un elemento del listado del directorio.

```
void
hfile_dir_entry_remove(DirEntry *entry);
```

entry Elemento.

Observaciones:

Ver `hfile_dir_list`.

hfile_date

Obtiene la fecha de modificación más reciente de un archivo o directorio.

```
Date
hfile_date(const char_t *pathname,
           const bool_t recursive);
```

pathname Ruta al archivo o directorio. *"Filename y pathnameFilename y pathname"* (Página 181).

recursive Si `pathname` es un directorio, indica si se realiza una exploración en profundidad por los subdirectorios.

Retorna:

La fecha de modificación. Si `pathname` no existe `kDATE_NULL`.

Observaciones:

Si `pathname` es un directorio, se considerarán las fechas de modificación de los archivos también, no solo del propio directorio.

hfile_dir_sync

Sincroniza el contenido de dos directorios.

```
bool_t
hfile_dir_sync(const char_t *src,
               const char_t *dest,
               const bool_t recursive,
               const bool_t remove_in_dest,
               const char_t **except,
               const uint32_t except_size,
               ferror_t *error);
```

src	Directorio origen.
dest	Directorio destino.
recursive	Si `TRUE` procesa recursivamente los subdirectorios.
remove_in_dest	Si `TRUE` elimina en `dest` aquellos archivos/directorios que no estén en `src`.
except	Lista de nombres de archivo/directorio que permanecerán intactos en `dest`.
except_size	Tamaño del array `except`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si todo ha ido bien, `FALSE` si ha habido algún error.

Observaciones:

Si un archivo está en `src` y no en `dest`, se copia en `dest`. Si un archivo es más reciente en `src` también se copia en `dest`. Si un archivo existe en `dest` pero no en `src` y `remove_in_dest` es `TRUE`, se eliminará de `dest`. Si el archivo existe en `except` no será tenido en cuenta ni para copiar ni borrar. Si `recursive` es `TRUE` se procesarán los subdirectorios de esta forma: Si ambos subdirs existen en `src` y `dest` se ejecutará la misma lógica descrita en ambos subdirs. Si el subdir existe en `src` pero no en `dest`, se copiará en su totalidad a `dest`. Si existe en `dest` y no en `src` y `remove_in_dest` es `TRUE` será eliminado completamente de `dest`.

hfile_exists

Comprueba si `pathname` existe en el sistema de archivos.

```
bool_t
```

```
hfile_exists(const char_t *pathname,
             file_type_t *file_type);
```

pathname Ruta del directorio o archivo a comprobar. *"Filename y pathnameFilename y pathname"* (Página 181).

file_type Tipo de archivo. Pueder ser `NULL`.

Retorna:

`TRUE` si `pathname` existe, `FALSE` si no.

hfile_is_uptodate

Comprueba si un archivo está actualizado. Considera que `dest` es una copia o depende de `src`.

```
bool_t
hfile_is_uptodate(const char_t *src,
                  const char_t *dest);
```

src Pathname del archivo de origen.

dest Pathname del archivo de destino.

Retorna:

`TRUE` si `dest` existe y es más reciente que `src`. De lo contrario `FALSE`.

hfile_copy

Copia un archivo de una ubicación a otra.

```
bool_t
hfile_copy(const char_t *src,
           const char_t *dest,
           ferror_t *error);
```

```
hfile_copy("/home/john/image.png", "/home/john/images", NULL); // image.png
hfile_copy("/home/john/image.png", "/home/john/images/party.png", NULL); //
    ↪ party.png
```

src Pathname del archivo a copiar. *"Filename y pathnameFilename y pathname"* (Página 181).

dest Destino de la copia. Si es un directorio tendrá el mismo *filename* que el origen. De lo contrario, la copia se hará con otro nombre de archivo.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

TRUE si la copia se ha realizado con éxito. De lo contrario FALSE.

hfile_buffer

Crea un búfer con el contenido de un archivo en disco.

```
Buffer*
hfile_buffer(const char_t *pathname,
             ferror_t *error);
```

pathname Ruta del archivo a cargar.

error Código de error si la función falla. Puede ser NULL.

Retorna:

El búfer con los datos del archivo o NULL si la función falla.

Observaciones:

No funciona con archivos de más de 4Gb (32-bits).

hfile_string

Crea un string con el contenido de un archivo en disco.

```
String*
hfile_string(const char_t *pathname,
             ferror_t *error);
```

pathname Ruta del archivo a cargar.

error Código de error si la función falla. Puede ser NULL.

Retorna:

El objeto string con los datos del archivo de texto o NULL si la función falla.

Observaciones:

No funciona con archivos de más de 4Gb (32-bits).

hfile_stream

Crea un *"Memory streamMemory stream"* (Página 200) y lo inicializa con el contenido de un archivo.

```
Stream*
hfile_stream(const char_t *pathname,
             ferror_t *error);
```

pathname Ruta del archivo a cargar.

error Código de error si la función falla. Puede ser NULL.

Retorna:

El stream inicializado con los datos del archivo o NULL si la función falla.

Observaciones:

No funciona con archivos de más de 4Gb (32-bits).

hfile_from_string

Crea un archivo en disco con el contenido de un *"Strings"* (Página 196).

```
bool_t
hfile_from_string(const char_t *pathname,
                  const String *str,
                  ferror_t *error);
```

pathname Ruta del archivo a guardar.

str String a guardar en el archivo.

error Código de error si la función falla. Puede ser NULL.

Retorna:

TRUE si el archivo ha sido creado con éxito. En caso contrario FALSE.

hfile_from_data

Crea un archivo en disco con el contenido de un bloque genérico de memoria.

```
bool_t
hfile_from_data(const char_t *pathname,
                const byte_t *data,
                const uint32_t size,
                ferror_t *error);
```

pathname	Ruta del archivo a guardar.
data	Bloque a guardar en el archivo.
size	Tamaño en bytes del bloque.
error	Código de error si la función falla. Puede ser NULL.

Retorna:

TRUE si el archivo ha sido creado con éxito. En caso contrario FALSE.

hfile_dir_loop

Recorre todos los archivos de un directorio.

```
bool_t
hfile_dir_loop(const char_t *pathname,
               Listener *listener,
               const bool_t subdirs,
               const bool_t hiddens,
               ferror_t *error);
```

```
static void i_OnEntry(App *app, Event *event)
{
    uint32_t type = event_type(event);
    const EvFileDir *p = event_params(event, EvFileDir);
    if (type == ekEFILE)
    {
        bstd_printf("File: %s\n", p->pathname);
        // Abort the directory loop
        if (app->more == FALSE)
        {
            bool_t *more = event_result(event, bool_t);
            *more = FALSE;
        }
    }
    else if (type == ekEENTRY)
    {
        if (app->direntry == TRUE)
        {
            bstd_printf("Entering: %s\n", params->pathname);
        }
        else
        {
            bool_t *entry = event_result(event, bool_t);
            *entry = FALSE;
        }
    }
    else if (type == ekEEXIT)
    {
```

```
        bstd_printf("Exiting: %s\n", params->pathname);
    }
}

hfile_dir_loop("/home/john/personal", listener(app, i_OnEntry, App), TRUE,
    ↪ FALSE, NULL);
```

pathname Ruta del directorio. *"Filename y pathnameFilename y pathname"* (Página 181).

listener Función *callback* que se llamará para cada archivo del directorio.

subdirs Si es `TRUE` el bucle procesará los subdirectorios.

hiddens Si es `TRUE` se procesarán los archivos ocultos.

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si el recorrido ha sido completado con éxito. `FALSE` si ha ocurrido algún error.

Observaciones:

Para cada archivo, se mandará un evento a `listener`. Será de tipo `ekEFILE` para archivos regulares, `ekEENTRY` cuando detecte en un subdirectorio y `ekEEXIT` cuando salga del mismo (si el subdirectorio se ha procesado). Los atributos del archivo se envían en el parámetro del evento como una estructura `EvFileDir`. El recorrido continuará hasta que todos los archivos/subdirectorios hayan sido recorridos o se retorne `FALSE` en `event_result`. Esta salida controlada no será considerada error y esta función retornará `TRUE`.

hfile_appdata

Obtiene la ruta completa de un archivo de datos o configuración de la aplicación.

```
String*
hfile_appdata(const char_t *pathname);
```

```
String *fname = hfile_appdata("gui/preferences.cfg");
fname = "C:\Users\USER\AppData\Roaming\MyApp\gui\preferences.cfg"
(in Windows operating system)
...
Stream *out = stm_to_file(tc(fname), NULL);
```

pathname Ruta relativa del archivo.

Retorna:

La ruta completa al archivo de configuración.

Observaciones:

En muchas ocasiones, las aplicaciones necesitan crear archivos de configuración para recordar preferencias del usuario u otro tipo de datos entre sesiones *"Home y AppDataHome y AppData"* (Página 181). Esta función añade una ruta relativa y el nombre del archivo y asegura que todos los directorios intermedios existirán.

hfile_home_dir

Obtiene la ruta completa a un archivo en el directorio del usuario **(home)**.

```
String*
hfile_home_dir(const char_t *path);
```

path Ruta relativa a partir del directorio **home**.

Retorna:

Ruta absoluta del archivo.

respack_destroy

Destruye un paquete de recursos.

```
void
respack_destroy(ResPack **pack);
```

pack Paquete de recursos. Será puesto a `NULL` tras la destrucción.

respack_text

Obtiene un texto de un paquete de recursos.

```
const char_t*
respack_text(const ResPack *pack,
             const ResId id);
```

pack Paquete de recursos.

id Identificador del recurso.

Retorna:

Cadena C UTF8 terminada en carácter nulo `'\0'`.

respack_file

Obtiene un puntero al contenido de un archivo, incluido en un paquete de recursos.

```
const byte_t*
respack_file(const ResPack *pack,
             const ResId id,
             uint32_t *size);
```

pack	Paquete de recursos.
id	Identificador del recurso.
size	Obtiene el tamaño del archivo en bytes.

Retorna:

Puntero al contenido del archivo (bytes en bruto).

date_system

Obtiene la fecha del sistema.

```
Date
date_system(void);
```

Retorna:

La fecha actual.

date_add_seconds

Calcula la fecha resultante de añadir una cantidad de segundos a otra fecha.

```
Date
date_add_seconds(const Date *date,
                 int32_t seconds);
```

date	La fecha base.
seconds	El número de segundos. Si es positivo obtendremos un fecha futura. Si es negativo una fecha pasada.

Retorna:

La fecha resultado.

date_add_minutes

Calcula la fecha resultante de añadir una cantidad de minutos a otra fecha.

```
Date
date_add_minutes(const Date *date,
                 int32_t minutes);
```

date	La fecha base.
minutes	El número de minutos. Si es positivo obtendremos un fecha futura. Si es negativo una fecha pasada.

Retorna:

La fecha resultado.

date_add_hours

Calcula la fecha resultante de añadir una cantidad de horas a otra fecha.

```
Date
date_add_hours(const Date *date,
               int32_t hours);
```

date	La fecha base.
hours	El número de horas. Si es positivo obtendremos un fecha futura. Si es negativo una fecha pasada.

Retorna:

La fecha resultado.

date_add_days

Calcula la fecha resultante de añadir una cantidad de días a otra fecha.

```
Date
date_add_days(const Date *date,
              int32_t days);
```

date	La fecha base.
days	El número de días. Si es positivo obtendremos un fecha futura. Si es negativo una fecha pasada.

Retorna:

La fecha resultado.

date_year

Obtiene el año actual.

```
int16_t
date_year(void);
```

Retorna:

El año actual.

date_cmp

Compara dos fechas. La fecha más reciente es considerada mayor.

```
int
date_cmp(const Date *date1,
         const Date *date2);
```

date1 Primera fecha a comparar.

date2 Segunda fecha a comparar.

Retorna:

Resultado de la comparación.

date_between

Comprueba si una fecha está dentro de un intervalo.

```
bool_t
date_between(const Date *date,
             const Date *from,
             const Date *to);
```

date Fecha a comprobar.

from Fecha de inicio.

to Fecha final.

Retorna:

`TRUE` si `date` está entre `from` y `to`.

date_is_null

Comprueba si una fecha es nula.

```
bool_t
date_is_null(const Date *date);
```

date Fecha a comprobar.

Retorna:

`TRUE` si `date` es nula.

date_DD_MM_YYYY_HH_MM_SS

Convierte una fecha a string, con el formato `DD/MM/YYYY-HH:MM:SS`.

```
String*
date_DD_MM_YYYY_HH_MM_SS(const Date *date);
```

date Fecha.

Retorna:

Objeto string con la conversión.

date_YYYY_MM_DD_HH_MM_SS

Convierte una fecha a string, con el formato `YYYY/MM/DD-HH:MM:SS`.

```
String*
date_YYYY_MM_DD_HH_MM_SS(const Date *date);
```

date Fecha.

Retorna:

Objeto string con la conversión.

date_month_en

Obtiene el nombre del mes, en Inglés.

```
const char_t*
date_month_en(const month_t month);
```

month El mes, normalmente obtenido con `btime_date`.

Retorna:

Cadena UTF8 con el nombre (January, February, ...).

date_month_es

Obtiene el nombre del mes, en Español.

```
const char_t*
date_month_es(const month_t month);
```

month El mes, normalmente obtenido con `btime_date`.

Retorna:

Cadena UTF8 con el nombre (Enero, Febrero, ...).

clock_create

Crea un reloj.

```
Clock*
clock_create(const real64_t interval);
```

interval Intervalo de tiempo para control de animaciones (en segundos).

Retorna:

El nuevo reloj.

clock_destroy

Destruye el reloj.

```
void
clock_destroy(Clock **clk);
```

clk Reloj. Será puesto a `NULL` tras la destrucción.

clock_frame

Detecta si ha vencido una nueva secuencia en una animación.

```
bool_t
clock_frame(Clock *clk,
            real64_t *prev_frame,
            real64_t *curr_frame);
```

clk Reloj.

prev_frame Marca temporal del instante anterior. Solo relevante si retorna `TRUE`.

curr_frame Marca temporal del instante actual. Solo relevante si retorna `TRUE`.

Retorna:

`TRUE` si ha llegado el momento para lanzar un nueva secuencia. `FALSE` si hay que esperar.

clock_reset

Pone a 0.0 el reloj.

```
void
clock_reset(Clock *clk);
```

clk Reloj.

clock_elapsed

Obtiene el tiempo transcurrido desde la creación del objeto o desde la última llamada a `clock_reset`.

```
real64_t
clock_elapsed(Clock *clk);
```

clk Reloj.

Retorna:

El número de segundos (con precisión de micro-segundos 0.000001).

Capítulo 38

Librería Geom2D

38.1. Tipos y Constantes

kZERO

El vector (0,0).

```
const V2Df kV2D_ZEROf;
const V2Dd kV2D_ZEROd;
const V2D V2D::kZERO;
```

kX

El vector (1,0).

```
const V2Df kV2D_Xf;
const V2Dd kV2D_Xd;
const V2D V2D::kX;
```

kY

El vector (0,1).

```
const V2Df kV2D_Yf;
const V2Dd kV2D_Yd;
const V2D V2D::kY;
```

kZERO

Valor `[0,0]`.

```
const S2Df kS2D_ZEROf;
const S2Dd kS2D_ZEROd;
const S2D S2D::kZERO;
```

kZERO

Valor `[0,0,0,0]`.

```
const R2Df kR2D_ZEROf;
const R2Dd kR2D_ZEROd;
const R2D R2D::kZERO;
```

kIDENT

Representa la transformación identidad.

```
const T2Df kT2D_IDENTf;
const T2Dd kT2D_IDENTd;
const T2D T2D::kIDENT;
```

kNULL

Representa un círculo nulo (sin geometría).

```
const Cir2Df kCIR2D_NULLf;
const Cir2Dd kCIR2D_NULLd;
const Cir2D Cir2D::kNULL;
```

kNULL

Representa un contenedor nulo (sin geometría).

```
const Box2Df kBOX2D_NULLf;
const Box2Dd kBOX2D_NULLd;
const Box2D Box2D::kNULL;
```

struct V2D

Representa un vector o punto 2d. *"Vectores 2D"* (Página 242).

```
struct V2Df
{
    real32_t x;
    real32_t y;
```

```
};

struct V2Dd
{
    real64_t x;
    real64_t y;
};

struct V2D
{
    real x;
    real y;
};
```

x Coordenada x.

y Coordenada y.

struct S2D

Representa un tamaño en 2d. *"Tamaños 2D"* (Página 245).

```
struct S2Df
{
    real32_t width;
    real32_t height;
};

struct S2Dd
{
    real64_t width;
    real64_t height;
};

struct S2D
{
    real width;
    real height;
};
```

`width` Ancho.

`height` Alto.

struct R2D

Rectángulo en 2d. *"Rectángulos 2D"* (Página 246).

```
struct R2Df
{
```

```
    V2Df pos;
    S2Df size;
};

struct R2Dd
{
    V2Dd pos;
    S2Dd size;
};

struct R2D
{
    V2D pos;
    S2D size;
};
```

`pos` Origen.

`size` Tamaño.

struct T2D

Transformación afín en 2d. *"Transformaciones 2D"* (Página 246).

```
struct T2Df
{
    V2Df i;
    V2Df j;
    V2Df p;
};

struct T2Dd
{
    V2Dd i;
    V2Dd j;
    V2Dd p;
};

struct T2D
{
    V2D i;
    V2D j;
    V2D p;
};
```

`i` Componente i de la transformación lineal.

`j` Componente j de la transformación lineal.

p Posición.

struct Seg2D

Segmento de recta en 2d. *"Segmentos 2D"* (Página 252).

```
struct Seg2Df
{
    V2Df p0;
    V2Df p1;
};

struct Seg2Dd
{
    V2Dd p0;
    V2Dd p1;
};

struct Seg2D
{
    V2D p0;
    V2D p1;
};
```

p0 Coordenada del primer punto del segmento.

p1 Coordenada del segundo punto del segmento.

struct Cir2D

Círculo en 2d. *"Círculos 2D"* (Página 253).

```
struct Cir2Df
{
    V2Df c;
    real32_t r;
};

struct Cir2Dd
{
    V2Dd c;
    real64_t r;
};

struct Cir2D
{
    V2D c;
    real r;
};
```

`c` Centro.

`r` Radio.

struct Box2D

Bounding Box en 2d. *"Cajas 2D"* (Página 254).

```
struct Box2Df
{
    V2Df min;
    V2Df max;
};

struct Box2Dd
{
    V2Dd min;
    V2Dd max;
};

struct Box2D
{
    V2D min;
    V2D max;
};
```

`min` Coordenada mínima del bounding.

`max` Coordenada máxima del bounding.

struct OBB2D

Caja (rectángulo) orientada en 2d. *"Cajas Orientadas 2D"* (Página 254).

```
struct OBB2Df;

struct OBB2Dd;

struct OBB2D;
```

struct Tri2D

Triángulo 2d. *"Triángulos 2D"* (Página 256).

```
struct Tri2Df
{
    V2Df p0;
    V2Df p1;
    V2Df p2;
```

```
};

struct Tri2Dd
{
    V2Dd p0;
    V2Dd p1;
    V2Dd p2;
};

struct Tri2D
{
    V2D p0;
    V2D p1;
    V2D p2;
};
```

`p0` Coordenada del primer punto del triángulo.

`p1` Coordenada del segundo punto del triángulo.

`p2` Coordenada del tercer punto del triángulo.

struct Pol2D

Polígono convexo en 2d. *"Polígonos 2D"* (Página 257).

```
struct Pol2Df;

struct Pol2Dd;

struct Pol2D;
```

struct Col2D

Datos de colisión en 2d. *"Colisiones 2D"* (Página 259).

```
struct Col2Df;

struct Col2Dd;

struct Col2D;
```

38.2. Funciones

v2d

Crea un vector 2d a partir de sus componentes.

```
V2Df
v2df(const real32_t x,
     const real32_t y);

V2Dd
v2dd(const real64_t x,
     const real64_t y);

V2D
V2D(const real x,
    const real y);
```

x Coordenada x.

y Coordenada y.

Retorna:

El vector 2d.

v2d_tof

Convierte un vector de double a float.

```
V2Df
v2d_tof(const V2Dd *v);
```

v El vector.

Retorna:

El vector 2d en simple precisión.

v2d_tod

Convierte un vector de float a double.

```
V2Dd
v2d_tod(const V2Df *v);
```

v El vector.

Retorna:

El vector 2d en doble precisión.

v2d_tofn

Convierte un array de vector de double a float.

```
void
v2d_tofn(V2Df *vf,
         const V2Dd *vd,
         const uint32_t n);
```

vf El array destino.

vd El array origen.

n Número de elementos.

v2d_todn

Convierte un array de vector de float a double.

```
void
v2d_todn(V2Dd *vd,
         const V2Df *vf,
         const uint32_t n);
```

vd El array destino.

vf El array origen.

n Número de elementos.

v2d_add

Suma dos vectores.

```
V2Df
v2d_addf(const V2Df *v1,
         const V2Df *v2);

V2Dd
v2d_addd(const V2Dd *v1,
         const V2Dd *v2);

V2D
V2D::add(const V2D *v1,
         const V2D *v2);
```

v1 Vector 1.

v2 Vector 2.

Retorna:

El vector resultado.

v2d_sub

Resta dos vectores.

```
V2Df
v2d_subf(const V2Df *v1,
         const V2Df *v2);

V2Dd
v2d_subd(const V2Dd *v1,
         const V2Dd *v2);

V2D
V2D::sub(const V2D *v1,
         const V2D *v2);
```

v1 Vector 1.

v2 Vector 2.

Retorna:

El vector resultado.

v2d_mul

Multiplica un vector por un escalar.

```
V2Df
v2d_mulf(const V2Df *v,
         const real32_t s);

V2Dd
v2d_muld(const V2Dd *v,
         const real64_t s);

V2D
V2D::mul(const V2D *v,
         const real s);
```

v Vector.

s Escalar.

Retorna:

El vector resultado.

v2d_from

Crea un vector a partir de un punto y una dirección.

```
V2Df
v2d_fromf(const V2Df *v,
          const V2Df *dir,
          const real32_t length);

V2Dd
v2d_fromd(const V2Dd *v,
          const V2Dd *dir,
          const real64_t length);

V2D
V2D::from(const V2D *v,
          const V2D *dir,
          const real length);
```

v	Vector inicial.
dir	Dirección.
length	Longitud.

Retorna:

El vector resultado.

Observaciones:

Realizará la operación `r = v + length * dir`. `dir` no es necesario que sea unitario, en cuyo caso `length` se comportará como un factor de escala.

v2d_mid

Retorna el punto medio de dos puntos.

```
V2Df
v2d_midf(const V2Df *v1,
         const V2Df *v2);

V2Dd
v2d_midd(const V2Dd *v1,
         const V2Dd *v2);

V2D
V2D::mid(const V2D *v1,
         const V2D *v2);
```

v1 Primer punto.

v2 Segundo punto.

Retorna:

El punto medio.

v2d_unit

Vector unitario (dirección) desde 1 a 2.

```
V2Df
v2d_unitf(const V2Df *v1,
          const V2Df *v2,
          real32_t *dist);

V2Dd
v2d_unitd(const V2Dd *v1,
          const V2Dd *v2,
          real64_t *dist);

V2D
V2D::unit(const V2D *v1,
          const V2D *v2,
          real *dist);
```

v1 Punto 1 (origen).

v2 Punto 2 (destino).

dist Distancia entre puntos. Puede ser NULL.

Retorna:

El vector unitario.

v2d_unit_xy

Vector unitario (dirección) desde 1 a 2.

```
V2Df
v2d_unit_xyf(const real32_t x1,
             const real32_t y1,
             const real32_t x2,
             const real32_t y2,
             real32_t *dist);

V2Dd
v2d_unit_xyd(const real64_t x1,
             const real64_t y1,
```

```
              const real64_t x2,
              const real64_t y2,
              real64_t *dist);

V2D
V2D::unit_xy(const real x1,
              const real y1,
              const real x2,
              const real y2,
              real *dist);
```

x1 Coordenada X del punto 1 (origen).

y1 Coordenada Y del punto 1 (origen).

x2 Coordenada X del punto 2 (destino).

y2 Coordenada Y del punto 2 (destino).

dist Distancia entre puntos. Puede ser NULL.

Retorna:

El vector unitario.

v2d_perp_pos

Obtiene el vector perpendicular positivo.

```
V2Df
v2d_perp_posf(const V2Df *v);

V2Dd
v2d_perp_posd(const V2Dd *v);

V2D
V2D::perp_pos(const V2D *v);
```

v Vector inicial.

Retorna:

El vector perpendicular.

Observaciones:

Es la perpendicular que se obtiene mediante ángulo positivo (+ /2).

v2d_perp_neg

Obtiene el vector perpendicular negativo.

```
V2Df
v2d_perp_negf(const V2Df *v);

V2Dd
v2d_perp_negd(const V2Dd *v);

V2D
V2D::perp_neg(const V2D *v);
```

v Vector inicial.

Retorna:

El vector perpendicular.

Observaciones:

Es la perpendicular que se obtiene mediante ángulo negativo (- /2).

v2d_from_angle

Obtiene el vector resultante de aplicar un giro al vector `[1,0]`.

```
V2Df
v2d_from_anglef(const real32_t a);

V2Dd
v2d_from_angled(const real64_t a);

V2D
V2D::from_angle(const real a);
```

a Ángulo.

Retorna:

El vector.

Observaciones:

Para `a=0` obtendremos `[1,0]`. Para `aπ=/2` `[0,1]`.

v2d_norm

Normaliza un vector, es decir, lo convierte en un vector de longitud = 1.

```
bool_t
v2d_normf(V2Df *v);
```

```
bool_t
v2d_normd(V2Dd *v);

bool_t
V2D::norm(V2D *v);
```

v El vector que será normalizado.

Retorna:

FALSE si el vector no se puede normalizar (el vector 0).

v2d_length

Calcula la longitud de un vector.

```
real32_t
v2d_lengthf(const V2Df *v);

real64_t
v2d_lengthd(const V2Dd *v);

real
V2D::length(const V2D *v);
```

v El vector.

Retorna:

El módulo del vector.

v2d_sqlength

Calcula el cuadrado de la longitud de un vector.

```
real32_t
v2d_sqlengthf(const V2Df *v);

real64_t
v2d_sqlengthd(const V2Dd *v);

real
V2D::sqlength(const V2D *v);
```

v El vector.

Retorna:

El cuadrado del módulo del vector.

Observaciones:

Evita utilizar la raíz cuadrada, por lo que es más eficiente que `v2d_lengthf`. Suele utilizarse para comparar distancias.

v2d_dot

Producto escalar de dos vectores.

```
real32_t
v2d_dotf(const V2Df *v1,
         const V2Df *v2);

real64_t
v2d_dotd(const V2Dd *v1,
         const V2Dd *v2);

real
V2D::dot(const V2D *v1,
         const V2D *v2);
```

v1 Vector 1.

v2 Vector 2.

Retorna:

Producto escalar.

v2d_dist

Calcula la distancia entre dos puntos.

```
real32_t
v2d_distf(const V2Df *v1,
          const V2Df *v2);

real64_t
v2d_distd(const V2Dd *v1,
          const V2Dd *v2);

real
V2D::dist(const V2D *v1,
          const V2D *v2);
```

v1 Punto 1.

v2 Punto 2.

Retorna:

La distancia.

v2d_sqdist

Calcula el cuadrado de la distancia entre dos puntos.

```
real32_t
v2d_sqdistf(const V2Df *v1,
            const V2Df *v2);

real64_t
v2d_sqdistd(const V2Dd *v1,
            const V2Dd *v2);

real
V2D::sqdist(const V2D *v1,
            const V2D *v2);
```

v1 Punto 1.

v2 Punto 2.

Retorna:

La distancia al cuadrado.

Observaciones:

Evita utilizar la raíz cuadrada, por lo que es más eficiente que `v2d_distf`. Suele utilizarse para comparar distancias.

v2d_angle

Calcula el ángulo formado por dos vectores.

```
real32_t
v2d_anglef(const V2Df *v1,
           const V2Df *v2);

real64_t
v2d_angled(const V2Dd *v1,
           const V2Dd *v2);

real
V2D::angle(const V2D *v1,
           const V2D *v2);
```

v1 Vector 1.

v2 Vector 2.

Retorna:

El ángulo en radianes (-Pi, Pi)

Observaciones:

Los ángulos positivos se expresan desde v1 a v2, en el mismo sentido que el giro del eje X hacia el eje Y. Para ángulos mayores de rad (180°) se devolverá sentido negativo.

v2d_rotate

Aplica un rotación a un vector.

```
void
v2d_rotatef(V2Df *v,
            const real32_t a);

void
v2d_rotated(V2Dd *v,
            const real64_t a);

void
V2D::rotate(V2D *v,
            const real a);
```

v Vector que será rotado (origen/destino).

a Ángulo en radianes.

Observaciones:

Esta función implica calcular el seno y coseno. Utiliza `t2d_vmultnf` si tienes que aplicar la misma rotación a varios vectores.

s2d

Crea una medida en 2d a partir de dos valores.

```
S2Df
s2df(const real32_t width,
     const real32_t height);

S2Dd
s2dd(const real64_t width,
     const real64_t height);

S2D
S2D(const real width,
    const real height);
```

width Ancho.

height Alto.

Retorna:

La medida.

r2d

Crea un rectángulo a partir de sus componentes.

```
R2Df
r2df(const real32_t x,
     const real32_t y,
     const real32_t width,
     const real32_t height);

R2Dd
r2dd(const real64_t x,
     const real64_t y,
     const real64_t width,
     const real64_t height);

R2D
R2D(const real x,
    const real y,
    const real width,
    const real height);
```

x Coordenada x del origen.

y Coordenada y del origen.

width Ancho.

height Alto.

Retorna:

El rectángulo.

r2d_center

Obtiene el punto central del rectángulo.

```
V2Df
r2d_centerf(const R2Df *r2d);

V2Dd
r2d_centerd(const R2Dd *r2d);
```

```
V2D
R2D::center(const R2D *r2d);
```

r2d Rectángulo.

Retorna:

El centro.

r2d_collide

Comprueba si dos rectángulos colisionan.

```
bool_t
r2d_collidef(const R2Df *r2d1,
             const R2Df *r2d2);

bool_t
r2d_collided(const R2Dd *r2d1,
             const R2Dd *r2d2);

bool_t
R2D::collide(const R2D *r2d1,
             const R2D *r2d2);
```

r2d1 Rectángulo 1.

r2d2 Rectángulo 2.

Retorna:

`TRUE` si hay colisión, `FALSE` si están separados.

r2d_contains

Comprueba si un punto está dentro del rectángulo.

```
bool_t
r2d_containsf(const R2Df *r2d,
              const real32_t x,
              const real32_t y);

bool_t
r2d_containsd(const R2Dd *r2d,
              const real64_t x,
              const real64_t y);

bool_t
R2D::contains(const R2D *r2d,
```

```
              const real x,
              const real y);
```

r2d Rectángulo.

x Coordenada x del punto.

y Coordenada y del punto.

Retorna:

TRUE si el punto está dentro.

r2d_clip

Comprueba si un rectángulo está totalmente fuera de un viewport.

```
bool_t
r2d_clipf(const R2Df *viewport,
          const R2Df *r2d);

bool_t
r2d_clipd(const R2Dd *viewport,
          const R2Dd *r2d);

bool_t
R2D::clip(const R2D *viewport,
          const R2D *r2d);
```

viewport Rectángulo contenedor.

r2d Rectángulo a comprobar.

Retorna:

TRUE si el rectángulo `r2d` se puede recortar.

Observaciones:

Útil para evitar procesar o dibujar objetos que estén totalmente fuera del área de visión.

r2d_join

Une dos rectángulos en uno.

```
void
r2d_joinf(R2Df *r2d,
          const R2Df *src);

void
```

```
r2d_joind(R2Dd *r2d,
          const R2Dd *src);

void
R2D::join(R2D *r2d,
          const R2D *src);
```

r2d Rectángulo destino. Su posición y tamaño será modificado para contener a `src`.

src Rectángulo que será añadido a `r2d`.

t2d_tof

Convierte una transformación de double a float.

```
void
t2d_tof(T2Df *dest,
        const T2Dd *src);
```

dest Transformación destino.

src Transformación origen.

t2d_tod

Convierte una transformación de float a double.

```
void
t2d_tod(T2Dd *dest,
        const T2Df *src);
```

dest Transformación destino.

src Transformación origen.

t2d_move

Multiplica una transformación por una traslación `dest = src * move(x,y)`.

```
void
t2d_movef(T2Df *dest,
          const T2Df *src,
          const real32_t x,
          const real32_t y);

void
t2d_moved(T2Dd *dest,
          const T2Dd *src,
```

```
          const real64_t x,
          const real64_t y);

void
T2D::move(T2D *dest,
          const T2D *src,
          const real x,
          const real y);
```

dest Transformación resultado.

src Transformación inicial.

x Coordenada x del desplazamiento.

y Coordenada y del desplazamiento.

Observaciones:

`dest` y `src` pueden apuntar a la misma matrix.

t2d_rotate

Multiplica una transformación por una rotación `dest = src * rotate(a)`.

```
void
t2d_rotatef(T2Df *dest,
            const T2Df *src,
            const real32_t a);

void
t2d_rotated(T2Dd *dest,
            const T2Dd *src,
            const real64_t a);

void
T2D::rotate(T2D *dest,
            const T2D *src,
            const real a);
```

dest Transformación resultado.

src Transformación inicial.

a Ángulo de rotación en radianes. Los ángulos positivos son los que giran desde el eje X al eje Y.

Observaciones:

`dest` y `src` pueden apuntar a la misma matrix.

t2d_scale

Multiplica una transformación por un escalado `dest = src * scale(sx,sy)`.

```
void
t2d_scalef(T2Df *dest,
           const T2Df *src,
           const real32_t sx,
           const real32_t sy);

void
t2d_scaled(T2Dd *dest,
           const T2Dd *src,
           const real64_t sx,
           const real64_t sy);

void
T2D::scale(T2D *dest,
           const T2D *src,
           const real sx,
           const real sy);
```

dest	Transformación resultado.
src	Transformación inicial.
sx	Escalado en el eje x.
sy	Escalado en el eje y.

Observaciones:

`dest` y `src` pueden apuntar a la misma matrix.

t2d_invfast

Calcula la transformación inversa, suponiendo que la entrada sea ortogonal.

```
void
t2d_invfastf(T2Df *dest,
             const T2Df *src);

void
t2d_invfastd(T2Dd *dest,
             const T2Dd *src);

void
T2D::invfast(T2D *dest,
             const T2D *src);
```

dest Transformación inversa.

src Transformación inicial.

Observaciones:

La transformación será ortogonal solo si contiene rotaciones y traslaciones, de lo contrario el resultado de aplicarla será impredecible. `dest` y `src` pueden apuntar a la misma matrix.

t2d_inverse

Calcula la transformación inversa.

```
void
t2d_inversef(T2Df *dest,
             const T2Df *src);

void
t2d_inversed(T2Dd *dest,
             const T2Dd *src);

void
T2D::inverse(T2D *dest,
             const T2D *src);
```

dest Transformación inversa.

src Transformación inicial.

Observaciones:

`dest` y `src` pueden apuntar a la misma matriz.

t2d_mult

Multiplica dos transformaciones `dest = src1 * src2`.

```
void
t2d_multf(T2Df *dest,
          const T2Df *src1,
          const T2Df *src2);

void
t2d_multd(T2Dd *dest,
          const T2Dd *src1,
          const T2Dd *src2);

void
T2D::mult(T2D *dest,
```

```
          const T2D *src1,
          const T2D *src2);
```

dest Transformación resultado.

src1 Primer operando.

src2 Segundo operando.

Observaciones:

`dest`, `src1` y `src2` pueden apuntar a la misma matrix.

t2d_vmult

Transforma un vector `dest = t2d * src`.

```
void
t2d_vmultf(V2Df *dest,
           const T2Df *t2d,
           const V2Df *src);

void
t2d_vmultd(V2Dd *dest,
           const T2Dd *t2d,
           const V2Dd *src);

void
T2D::vmult(V2D *dest,
           const T2D *t2d,
           const V2D *src);
```

dest Vector transformado.

t2d Transformación.

src Vector original.

Observaciones:

`dest` y `src` pueden apuntar al mismo vector.

t2d_vmultn

Transforma una lista de vectores `dest[i] = t2d * src[i]`.

```
void
t2d_vmultnf(V2Df *dest,
            const T2Df *t2d,
            const V2Df *src,
            const uint32_t n);
```

```
void
t2d_vmultnd(V2Dd *dest,
            const T2Dd *t2d,
            const V2Dd *src,
            const uint32_t n);

void
T2D::vmultn(V2D *dest,
            const T2D *t2d,
            const V2D *src,
            const uint32_t n);
```

dest	Array de vectores transformado.
t2d	Transformación.
src	Array de vectores original.
n	Número de vectores en `src`.

Observaciones:

`dest` y `src` pueden apuntar al mismo array.

t2d_decompose

Obtiene la posición, rotación y escalado de una transformación.

```
void
t2d_decomposef(const T2Df *t2d,
               V2Df *pos,
               real32_t *a,
               V2Df *sc);

void
t2d_decomposed(const T2Dd *t2d,
               V2Dd *pos,
               real64_t *a,
               V2Dd *sc);

void
T2D::decompose(const T2D *t2d,
               V2D *pos,
               real *a,
               V2D *sc);
```

t2d Transformación.

pos Posición. Puede ser `NULL`.

a Ángulo en radianes (- /2, /2). Puede ser `NULL`.

sc Escalado. Puede ser `NULL`.

Observaciones:

Si la transformación no está compuesta por una secuencia de traslaciones, rotaciones y escalados, el resultado no será válido.

seg2d

Crea un segmento 2d a partir de sus componentes.

```
Seg2Df
seg2df(const real32_t x0,
       const real32_t y0,
       const real32_t x1,
       const real32_t y1);

Seg2Dd
seg2dd(const real64_t x0,
       const real64_t y0,
       const real64_t x1,
       const real64_t y1);

Seg2D
Seg2D(const real x0,
      const real y0,
      const real x1,
      const real y1);
```

x0 Coordenada x del primer punto.

y0 Coordenada y del primer punto.

x1 Coordenada x del segundo punto.

y1 Coordenada y del segundo punto.

Retorna:

El segmento 2d.

seg2d_v

Crea un segmento 2d a partir de dos puntos.

```
Seg2Df
seg2d_vf(const V2Df *p0,
         const V2Df *p1);

Seg2Dd
seg2d_vd(const V2Dd *p0,
         const V2Dd *p1);

Seg2D
Seg2D::v(const V2D *p0,
         const V2D *p1);
```

p0 Primer punto.

p1 Segundo punto.

Retorna:

El segmento 2d.

seg2d_length

Obtiene la longitud del segmento.

```
real32_t
seg2d_lengthf(const Seg2Df *seg);

real64_t
seg2d_lengthd(const Seg2Dd *seg);

real
Seg2D::length(const Seg2D *seg);
```

seg Segmento.

Retorna:

Longitud.

seg2d_sqlength

Obtiene el cuadrado de la longitud del segmento.

```
real32_t
seg2d_sqlengthf(const Seg2Df *seg);

real64_t
seg2d_sqlengthd(const Seg2Dd *seg);
```

```
real
Seg2D::sqlength(const Seg2D *seg);
```

seg Segmento.

Retorna:

Cuadrado de la longitud.

Observaciones:

Evita calcular raíces cuadradas si solo nos interesa comparar medidas.

seg2d_eval

Obtiene el punto en el segmento según el parámetro.

```
V2Df
seg2d_evalf(const Seg2Df *seg,
            const real32_t t);

V2Dd
seg2d_evald(const Seg2Dd *seg,
            const real64_t t);

V2D
Seg2D::eval(const Seg2D *seg,
            const real t);
```

seg Segmento.

t Parámetro.

Retorna:

Punto en el segmento (o en la recta que lo contiene).

Observaciones:

Si `t=0` devuelve `p0`. Si `t=1` devuelve `p1`. Valores entre `(0,1)` puntos dentro del segmento. Otros valores, puntos en la recta que contiene al segmento.

seg2d_close_param

Obtiene el parámetro del segmento más cercano a un punto determinado.

```
real32_t
seg2d_close_paramf(const Seg2Df *seg,
                   const V2Df *pnt);
```

```
real64_t
seg2d_close_paramd(const Seg2Dd *seg,
                   const V2Dd *pnt);

real
Seg2D::close_param(const Seg2D *seg,
                   const V2D *pnt);
```

seg Segmento.

pnt Punto.

Retorna:

Parámetro. Ver `seg2d_evalf`.

seg2d_point_sqdist

Obtiene la distancia al cuadrado de un punto al segmento.

```
real32_t
seg2d_point_sqdistf(const Seg2Df *seg,
                    const V2Df *pnt,
                    real32_t *t);

real64_t
seg2d_point_sqdistd(const Seg2Dd *seg,
                    const V2Dd *pnt,
                    real64_t *t);

real
Seg2D::point_sqdist(const Seg2D *seg,
                    const V2D *pnt,
                    real *t);
```

seg Segmento.

pnt Punto.

t Parámetro en la recta que contiene al segmento. Ver `seg2d_close_paramf`. Puede ser `NULL` si no necesitamos este valor.

Retorna:

Cuadrado de la distancia.

seg2d_sqdist

Obtiene la distancia al cuadrado entre dos segmentos.

```
real32_t
seg2d_sqdistf(const Seg2Df *seg1,
              const Seg2Df *seg2,
              real32_t *t1,
              real32_t *t2);

real64_t
seg2d_sqdistd(const Seg2Dd *seg1,
              const Seg2Dd *seg2,
              real64_t *t1,
              real64_t *t2);

real
Seg2D::sqdist(const Seg2D *seg1,
              const Seg2D *seg2,
              real *t1,
              real *t2);
```

seg1 Primer segmento.

seg2 Segundo segmento.

t1 Parámetro más cercano en `seg1`. Puede ser `NULL` si no necesitamos este valor.

t2 Parámetro más cercano en `seg2`. Puede ser `NULL` si no necesitamos este valor.

Retorna:

Cuadrado de la distancia.

cir2d

Crea un círculo 2d a partir de sus componentes.

```
Cir2Df
cir2df(const real32_t x,
       const real32_t y,
       const real32_t r);

Cir2Dd
cir2dd(const real64_t x,
       const real64_t y,
       const real64_t r);

Cir2D
```

```
Cir2D(const real x,
      const real y,
      const real r);
```

x Coordenada x del centro.

y Coordenada y del centro.

r Radio.

Retorna:

El círculo 2d.

cir2d_from_box

Crea un círculo que contiene a una caja 2D.

```
Cir2Df
cir2d_from_boxf(const B2D *box);

Cir2Dd
cir2d_from_boxd(const B2D *box);

Cir2D
Cir2D::from_box(const B2D *box);
```

box La caja.

Retorna:

El círculo.

cir2d_from_points

Crea un círculo que contiene un conjunto de puntos.

```
Cir2Df
cir2d_from_pointsf(const V2Df *p,
                   const uint32_t n);

Cir2Dd
cir2d_from_pointsd(const V2Dd *p,
                   const uint32_t n);

Cir2D
Cir2D::from_points(const V2D *p,
                   const uint32_t n);
```

p El vector de puntos.

n El número de puntos.

Retorna:

El círculo.

Observaciones:

El centro será el punto medio del conjunto. El radio será la distancia al punto más alejado de dicho centro. Proporciona un buen ajuste con coste lineal.

cir2d_minimum

Calcula el círculo de radio mínimo que contiene a un conjunto de puntos.

```
Cir2Df
cir2d_minimumf(const V2Df *p,
               const uint32_t n);

Cir2Dd
cir2d_minimumd(const V2Dd *p,
               const uint32_t n);

Cir2D
Cir2D::minimum(const V2D *p,
               const uint32_t n);
```

p El vector de puntos.

n El número de puntos.

Retorna:

El círculo.

Observaciones:

Proporciona un ajuste óptimo en tiempo lineal. No obstante, es más lento que `cir2d_from_pointsf`.

cir2d_area

Obtiene el área del círculo.

```
real32_t
cir2d_areaf(const Cir2Df *cir);

real64_t
```

```
cir2d_aread(const Cir2Dd *cir);

real
Cir2D::area(const Cir2D *cir);
```

cir El círculo.

Retorna:

El área $\pi(r^2)$.

cir2d_is_null

Comprueba si un círculo es nulo (sin dimensión).

```
bool_t
cir2d_is_nullf(const Cir2Df *cir);

bool_t
cir2d_is_nulld(const Cir2Dd *cir);

bool_t
Cir2D::is_null(const Cir2D *cir);
```

cir El círculo.

Retorna:

`TRUE` si es nulo, `FALSE` si contiene algún punto.

Observaciones:

Un solo punto es un círculo válido con radio = 0.

box2d

Crea una nueva caja con los límites indicados.

```
Box2Df
box2df(const real32_t minX,
       const real32_t minY,
       const real32_t maxX,
       const real32_t maxY);

Box2Dd
box2dd(const real64_t minX,
       const real64_t minY,
       const real64_t maxX,
       const real64_t maxY);
```

```
Box2D
Box2D(const real minX,
      const real minY,
      const real maxX,
      const real maxY);
```

minX El límite inferior en X.

minY El límite inferior en Y.

maxX El límite superior en X.

maxY El límite superior en Y.

Retorna:

La caja recién creada.

box2d_from_points

Crea una nueva caja que contiene un conjunto de puntos.

```
Box2Df
box2d_from_pointsf(const V2Df *p,
                   const uint32_t n);

Box2Dd
box2d_from_pointsd(const V2Dd *p,
                   const uint32_t n);

Box2D
Box2D::from_points(const V2D *p,
                   const uint32_t n);
```

p Vector de puntos 2d.

n Número de puntos en el vector.

Retorna:

La caja recién creada.

box2d_center

Devuelve el punto central.

```
V2Df
box2d_centerf(const Box2Df *box);

V2Dd
```

```
box2d_centerd(const Box2Dd *box);

V2D
Box2D::center(const Box2D *box);
```

box El contenedor.

Retorna:

Las coordenadas del centro.

box2d_add

Amplia las dimensiones del contenedor para albergar al punto de entrada. Si el punto ya está dentro de su área, las dimensiones no se modificarán.

```
void
box2d_addf(Box2Df *box,
           const V2Df *p);

void
box2d_addd(Box2Dd *box,
           const V2Dd *p);

void
Box2D::add(Box2D *box,
           const V2D *p);
```

box El contenedor.

p El punto a incluir.

box2d_addn

Amplia las dimensiones del contenedor para albergar varios puntos. Es equivalente a llamar sucesivas veces al método `box2d_addf`.

```
void
box2d_addnf(Box2Df *box,
            const V2Df *p,
            const uint32_t n);

void
box2d_addnd(Box2Dd *box,
            const V2Dd *p,
            const uint32_t n);

void
Box2D::addn(Box2D *box,
```

```
            const V2D *p,
            const uint32_t n);
```

box El contenedor.

p Vector de puntos a incluir.

n Número de puntos.

box2d_add_circle

Amplia las dimensiones del contenedor para albergar un círculo.

```
void
box2d_add_circlef(Box2Df *box,
                  const Cir2Df *cir);

void
box2d_add_circled(Box2Dd *box,
                  const Cir2Dd *cir);

void
Box2D::add_circle(Box2D *box,
                  const Cir2D *cir);
```

box El contenedor.

cir Círculo.

box2d_merge

Amplia las dimensiones de la caja `dest` para contener a la caja `src`.

```
void
box2d_mergef(Box2Df *dest,
             const Box2Df *src);

void
box2d_merged(Box2Dd *dest,
             const Box2Dd *src);

void
Box2D::merge(Box2D *dest,
             const Box2D *src);
```

dest La caja destino.

src La caja origen.

box2d_segments

Obtiene los cuatro segmentos que forman la caja.

```
void
box2d_segmentsf(const Box2Df *box,
                Seg2Df *segs);

void
box2d_segmentsd(const Box2Dd *box,
                Seg2Dd *segs);

void
Box2D::segments(const Box2D *box,
                Seg2D *segs);
```

box El contenedor.

segs Array de al menos cuatro segmentos.

box2d_area

Obtiene el área de la caja.

```
real32_t
box2d_areaf(const Box2Df *box);

real64_t
box2d_aread(const Box2Dd *box);

real
Box2D::area(const Box2D *box);
```

box El contenedor.

Retorna:

El área `(width * height)`.

box2d_is_null

Comprueba si un contendor es nulo (sin dimensión).

```
bool_t
box2d_is_nullf(const Box2Df *box);

bool_t
box2d_is_nulld(const Box2Dd *box);

bool_t
Box2D::is_null(const Box2D *box);
```

box El contenedor.

Retorna:

`TRUE` si es nulo, `FALSE` si contiene alguna geometría.

obb2d_create

Crea una nueva caja orientada.

```
OBB2Df*
obb2d_createf(const V2Df *center,
              const real32_t width,
              const real32_t height,
              const real32_t angle);

OBB2Dd*
obb2d_created(const V2Dd *center,
              const real64_t width,
              const real64_t height,
              const real64_t angle);

OBB2D*
OBB2D::create(const V2D *center,
              const real width,
              const real height,
              const real angle);
```

center El punto central.

width El ancho de la caja.

height El alto de la caja.

angle El ángulo con respecto al eje X, en radianes.

Retorna:

La caja recién creada.

Observaciones:

Los ángulos positivos son los que giran desde el eje X al eje Y.

obb2d_from_line

Crea una caja a partir de un segmento.

```
OBB2Df*
obb2d_from_linef(const V2Df *p0,
                 const V2Df *p1,
```

```
                  const real32_t thickness);

OBB2Dd*
obb2d_from_lined(const V2Dd *p0,
                  const V2Dd *p1,
                  const real64_t thickness);

OBB2D*
OBB2D::from_line(const V2D *p0,
                  const V2D *p1,
                  const real thickness);
```

p0 El primer punto del segmento.

p1 El segundo punto del segmento.

thickness El "grosor" del segmento.

Retorna:

La caja recién creada.

Observaciones:

La anchura de la caja corresponderá con la longitud del segmento. La altura será `thickness` y el centro el punto medio del segmento.

obb2d_from_points

Crea una caja orientada a partir de un conjunto de puntos.

```
OBB2Df*
obb2d_from_pointsf(const V2Df *p,
                   const uint32_t n);

OBB2Dd*
obb2d_from_pointsd(const V2Dd *p,
                   const uint32_t n);

OBB2D*
OBB2D::from_points(const V2D *p,
                   const uint32_t n);
```

p Vector de puntos.

n Número de puntos.

Retorna:

La caja recién creada.

Observaciones:

Se producirá un buen ajuste en distribuciones de puntos "alargadas" calculando la matriz de covarianza y proyectando puntos en el vector director de dicha distribución. No obstante, no proporciona la caja de volumen mínimo.

obb2d_copy

Crea una copia de la caja.

```
OBB2Df*
obb2d_copyf(const OBB2Df obb);

OBB2Dd*
obb2d_copyd(const OBB2Dd obb);

OBB2D*
OBB2D::copy(const OBB2D obb);
```

obb La caja original.

Retorna:

La caja copiada.

obb2d_destroy

Destruye la caja.

```
void
obb2d_destroyf(OBB2Df **obb);

void
obb2d_destroyd(OBB2Dd **obb);

void
OBB2D::destroy(OBB2D **obb);
```

obb La caja. Será puesto a `NULL` tras la destrucción.

obb2d_update

Actualiza los parámetros de la caja.

```
void
obb2d_updatef(OBB2Df *obb,
              const V2Df *center,
              const real32_t width,
              const real32_t height,
```

```
                const real32_t angle);

void
obb2d_updated(OBB2Dd *obb,
                const V2Dd *center,
                const real64_t width,
                const real64_t height,
                const real64_t angle);

void
OBB2D::update(OBB2D *obb,
                const V2D *center,
                const real width,
                const real height,
                const real angle);
```

obb	La caja a actualizar.
center	El punto central.
width	La anchura.
height	La altura.
angle	El ángulo.

Observaciones:

Ver `obb2d_createf`.

obb2d_move

Desplaza la caja en el plano.

```
void
obb2d_movef(OBB2Df *obb,
              const real32_t offset_x,
              const real32_t offset_y);

void
obb2d_moved(OBB2Dd *obb,
              const real64_t offset_x,
              const real64_t offset_y);

void
OBB2D::move(OBB2D *obb,
              const real offset_x,
              const real offset_y);
```

obb La caja.

offset_x Desplazamiento en X.

offset_y Desplazamiento en Y.

obb2d_transform

Aplica una transformación a la caja.

```
void
obb2d_transformf(OBB2Df *obb,
                 const T2Df *t2d);

void
obb2d_transformd(OBB2Dd *obb,
                 const T2Dd *t2d);

void
OBB2D::transform(OBB2D *obb,
                 const T2D *t2d);
```

obb La caja.

t2d La transformación afín.

obb2d_corners

Obtiene los vértices que limitan la caja.

```
const V2Df*
obb2d_cornersf(const OBB2Df *obb);

const V2Dd*
obb2d_cornersd(const OBB2Dd *obb);

const V2D*
OBB2D::corners(const OBB2D *obb);
```

obb La caja.

Retorna:

Puntero a un array de 4 vértices.

Observaciones:

No modificar el array devuelto. Copiar si fuera necesario.

obb2d_center

Obtiene el punto central de la caja.

```
V2Df
obb2d_centerf(const OBB2Df *obb);

V2Dd
obb2d_centerd(const OBB2Dd *obb);

V2D
OBB2D::center(const OBB2D *obb);
```

obb La caja.

Retorna:

El centro.

obb2d_width

Obtiene la anchura de la caja.

```
real32_t
obb2d_widthf(const OBB2Df *obb);

real64_t
obb2d_widthd(const OBB2Dd *obb);

real
OBB2D::width(const OBB2D *obb);
```

obb La caja.

Retorna:

La anchura.

obb2d_height

Obtiene la altura de la caja.

```
real32_t
obb2d_heightf(const OBB2Df *obb);

real64_t
obb2d_heightd(const OBB2Dd *obb);

real
OBB2D::height(const OBB2D *obb);
```

obb La caja.

Retorna:

La altura.

obb2d_angle

Obtiene el ángulo de la caja.

```
real32_t
obb2d_anglef(const OBB2Df *obb);

real64_t
obb2d_angled(const OBB2Dd *obb);

real
OBB2D::angle(const OBB2D *obb);
```

obb La caja.

Retorna:

El ángulo en radianes con respecto al eje X.

obb2d_area

Obtiene el área de la caja.

```
real32_t
obb2d_areaf(const OBB2Df *obb);

real64_t
obb2d_aread(const OBB2Dd *obb);

real
OBB2D::area(const OBB2D *obb);
```

obb La caja.

Retorna:

El área (`width * height`).

obb2d_box

Obtiene los límites de la caja.

```
Box2Df
obb2d_boxf(const OBB2Df *obb);

Box2Dd
obb2d_boxd(const OBB2Dd *obb);

Box2D
OBB2D::box(const OBB2D *obb);
```

obb La caja.

Retorna:

Caja alineada con los ejes, definida por los vectores mínimo y máximo.

tri2d

Triángulo a partir de sus coordenadas.

```
Tri2Df
tri2df(const real32_t x0,
       const real32_t y0,
       const real32_t x1,
       const real32_t y1,
       const real32_t x2,
       const real32_t y2);

Tri2Dd
tri2dd(const real64_t x0,
       const real64_t y0,
       const real64_t x1,
       const real64_t y1,
       const real64_t x2,
       const real64_t y2);

Tri2D
Tri2D(const real x0,
      const real y0,
      const real x1,
      const real y1,
      const real x2,
      const real y2);
```

x0 Coordenada x del primer punto.

y0 Coordenada y del primer punto.

x1 Coordenada x del segundo punto.

y1 Coordenada y del segundo punto.

x2 Coordenada x del tercer punto.

y2 Coordenada y del tercer punto.

Retorna:

El triángulo.

tri2d_v

Triángulo a partir de tres puntos.

```
Tri2Df
tri2d_vf(const V2Df *p0,
         const V2Df *p1,
         const V2Df *p2);

Tri2Dd
tri2d_vd(const V2Dd *p0,
         const V2Dd *p1,
         const V2Dd *p2);

Tri2D
Tri2D::v(const V2D *p0,
         const V2D *p1,
         const V2D *p2);
```

p0 Primer punto.

p1 Segundo punto.

p2 Tercer punto.

Retorna:

El triángulo.

tri2d_transform

Aplica una transformación al triángulo.

```
void
tri2d_transformf(Tri2Df *tri,
                 const T2Df *t2d);
```

```
void
tri2d_transformd(Tri2Dd *tri,
                 const T2Dd *t2d);

void
Tri2D::transform(Tri2D *tri,
                 const T2D *t2d);
```

tri El triángulo.

t2d La transformación afín.

tri2d_area

Obtiene el área del triángulo.

```
real32_t
tri2d_areaf(const Tri2Df *tri);

real64_t
tri2d_aread(const Tri2Dd *tri);

real
Tri2D::area(const Tri2D *tri);
```

tri El triángulo.

Retorna:

El área.

tri2d_ccw

Obtiene el orden del recorrido de los puntos del triángulo.

```
bool_t
tri2d_ccwf(const Tri2Df *tri);

bool_t
tri2d_ccwd(const Tri2Dd *tri);

bool_t
Tri2D::ccw(const Tri2D *tri);
```

tri El triángulo.

Retorna:

`TRUE` sentido antihorario *counter-clockwise*. `FALSE` sentido horario *clockwise*.

Observaciones:

Ver *"Ángulos CW y CCWÁngulos CW y CCW"* (Página 244).

tri2d_centroid

Obtiene el centroide (centro de masas) del triángulo.

```
V2Df
tri2d_centroidf(const Tri2Df *tri);

V2Dd
tri2d_centroidd(const Tri2Dd *tri);

V2D
Tri2D::centroid(const Tri2D *tri);
```

tri El triángulo.

Retorna:

El centro de masas.

pol2d_create

Crea un nuevo polígono.

```
Pol2Df*
pol2d_createf(const V2Df *points,
              const uint32_t n);

Pol2Dd*
pol2d_created(const V2Dd *points,
              const uint32_t n);

Pol2D*
Pol2D::create(const V2D *points,
              const uint32_t n);
```

points Lista de puntos que forman el polígono.

n Número de puntos.

Retorna:

El polígono creado.

pol2d_convex_hull

Crea el polígono convexo mínimo que envuelve a un conjunto de puntos *(Convex Hull)*.

```
Pol2Df*
pol2d_convex_hullf(const V2Df *points,
                   const uint32_t n);

Pol2Dd*
pol2d_convex_hulld(const V2Dd *points,
                   const uint32_t n);

Pol2D*
Pol2D::convex_hull(const V2D *points,
                   const uint32_t n);
```

points Lista de puntos.

n Número de puntos.

Retorna:

El polígono.

pol2d_copy

Crea una copia del polígono.

```
Pol2Df*
pol2d_copyf(const Pol2Df *pol);

Pol2Dd*
pol2d_copyd(const Pol2Dd *pol);

Pol2D*
Pol2D::copy(const Pol2D *pol);
```

pol El polígono original.

Retorna:

El polígono copiado.

pol2d_destroy

Destruye el polígono.

```
void
pol2d_destroyf(Pol2Df **pol);

void
pol2d_destroyd(Pol2Dd **pol);
```

```
void
Pol2D::destroy(Pol2D **pol);
```

pol El polígono. Será puesto a NULL tras la destrucción.

pol2d_transform

Aplica una transformación 2D.

```
void
pol2d_transformf(Pol2Df *pol,
                 const T2Df *t2d);

void
pol2d_transformd(Pol2Dd *pol,
                 const T2Dd *t2d);

void
Pol2D::transform(Pol2D *pol,
                 const T2D *t2d);
```

pol El polígono.

t2d Transformación 2D.

Observaciones:

El polígono no guarda las coordenadas originales. Sucesivas transformaciones se irán acumulando.

pol2d_points

Obtiene los vértices que forman el polígono.

```
const V2Df*
pol2d_pointsf(const Pol2Df *pol);

const V2Dd*
pol2d_pointsd(const Pol2Dd *pol);

const V2D*
Pol2D::points(const Pol2D *pol);
```

pol El polígono.

Retorna:

Puntero a un array de vértices.

Observaciones:

No modificar el array devuelto. Copiar si fuera necesario.

pol2d_n

Obtiene el número de vértices que forman el polígono.

```
uint32_t
pol2d_nf(const Pol2Df *pol);

uint32_t
pol2d_nd(const Pol2Dd *pol);

uint32_t
Pol2D::n(const Pol2D *pol);
```

pol El polígono.

Retorna:

El número de vértices.

Observaciones:

Es el mismo valor que el utilizado en el constructor `pol2d_createf`.

pol2d_area

Obtiene el área del polígono.

```
real32_t
pol2d_areaf(const Pol2Df *pol);

real64_t
pol2d_aread(const Pol2Dd *pol);

real
Pol2D::area(const Pol2D *pol);
```

pol El polígono.

Retorna:

El área.

pol2d_box

Obtiene los límites geométricos del polígono.

```
Box2Df
pol2d_boxf(const Pol2Df *pol);

Box2Dd
pol2d_boxd(const Pol2Dd *pol);

Box2D
Pol2D::box(const Pol2D *pol);
```

pol El polígono.

Retorna:

Caja alineada con los ejes, definida por los vectores mínimo y máximo.

pol2d_ccw

Obtiene el orden del recorrido de los puntos del polígono.

```
bool_t
pol2d_ccwf(const Pol2Df *pol);

bool_t
pol2d_ccwd(const Pol2Dd *pol);

bool_t
Pol2D::ccw(const Pol2D *pol);
```

pol El polígono.

Retorna:

`TRUE` sentido antihorario *counter-clockwise*. `FALSE` sentido horario *clockwise*.

pol2d_convex

Obtiene si el polígono es o no convexo.

```
bool_t
pol2d_convexf(const Pol2Df *pol);

bool_t
pol2d_convexd(const Pol2Dd *pol);

bool_t
Pol2D::convex(const Pol2D *pol);
```

pol El polígono.

Retorna:

`TRUE` si es convexo. `FALSE` si no.

pol2d_centroid

Obtiene el centroide (centro de masas) del polígono.

```
V2Df
pol2d_centroidf(const Pol2Df *pol);

V2Dd
pol2d_centroidd(const Pol2Dd *pol);

V2D
Pol2D::centroid(const Pol2D *pol);
```

pol El polígono.

Retorna:

El centro de masas.

pol2d_visual_center

Obtiene el centro visual o punto de etiquetado.

```
V2Df
pol2d_visual_centerf(const Pol2Df *pol);

V2Dd
pol2d_visual_centerd(const Pol2Dd *pol);

V2D
Pol2D::visual_center(const Pol2D *pol);
```

pol El polígono.

Retorna:

El centro de etiquetado.

Observaciones:

Corresponde a un punto dentro del polígono situado a una distancia máxima de cualquier borde. En polígonos convexos coincidirá con el centroide. Implementa una adaptación del algoritmo **polylabel** del proyecto MapBox[1].

[1] https://github.com/mapbox/polylabel

pol2d_triangles

Obtiene una lista de triángulos que forman el polígono.

```
ArrSt(Tri2Df)*
pol2d_trianglesf(const Pol2Df *pol);

ArrSt(Tri2Df)*
pol2d_trianglesd(const Pol2Dd *pol);

ArrSt(Tri2Df)*
Pol2D::triangles(const Pol2D *pol);
```

pol El polígono.

Retorna:

Array de triángulos. Debe ser destruido con `arrst_destroy(&triangles, NULL, Tri2Df)`.

Observaciones:

La unión de todos los triángulos corresponde con el polígono original.

pol2d_convex_partition

Obtiene una lista de los polígonos convexos que forman el polígono.

```
ArrSt(Pol2Df)*
pol2d_convex_partitionf(const Pol2Df *pol);

ArrSt(Pol2Df)*
pol2d_convex_partitiond(const Pol2Dd *pol);

ArrSt(Pol2Df)*
Pol2D::convex_partition(const Pol2D *pol);
```

pol El polígono.

Retorna:

Array de polígonos convexos. Debe ser destruido con `arrst_destroy(&polys, pol2d_destroyf, Pol2Df)`.

Observaciones:

La unión de todos los polígonos corresponde con el polígono original.

col2d_point_point

Colisión punto-punto.

```
bool_t
col2d_point_pointf(const V2Df *pnt1,
                   const V2Df *pnt2,
                   const real32_t tol,
                   Col2Df *col);

bool_t
col2d_point_pointd(const V2Dd *pnt1,
                   const V2Dd *pnt2,
                   const real64_t tol,
                   Col2Dd *col);

bool_t
Col2D::point_point(const V2D *pnt1,
                   const V2D *pnt2,
                   const real tol,
                   Col2D *col);
```

pnt1 Primer punto.

pnt2 Segundo punto.

tol Tolerancia. Distancia mínima para que se considere colisión.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_segment_point

Colisión segmento-punto.

```
bool_t
col2d_segment_pointf(const Seg2Df *seg,
                     const V2Df *pnt,
                     const real32_t tol,
                     Col2Df *col);

bool_t
col2d_segment_pointd(const Seg2Dd *seg,
                     const V2Dd *pnt,
                     const real64_t tol,
                     Col2Dd *col);

bool_t
```

```
Col2D::segment_point(const Seg2D *seg,
                     const V2D *pnt,
                     const real tol,
                     Col2D *col);
```

seg Segmento.

pnt Punto.

tol Tolerancia. Distancia mínima para que se considere colisión.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_segment_segment

Colisión segmento-segmento.

```
bool_t
col2d_segment_segmentf(const Seg2Df *seg1,
                       const Seg2Df *seg2,
                       Col2Df *col);

bool_t
col2d_segment_segmentd(const Seg2Dd *seg1,
                       const Seg2Dd *seg2,
                       Col2Dd *col);

bool_t
Col2D::segment_segment(const Seg2D *seg1,
                       const Seg2D *seg2,
                       Col2D *col);
```

seg1 Primer segmento.

seg2 Segundo segmento.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_circle_point

Colisión círculo-punto.

```
bool_t
col2d_circle_pointf(const Cir2Df *cir,
                    const V2Df *pnt,
                    Col2Df *col);

bool_t
col2d_circle_pointd(const Cir2Dd *cir,
                    const V2Dd *pnt,
                    Col2Dd *col);

bool_t
Col2D::circle_point(const Cir2D *cir,
                    const V2D *pnt,
                    Col2D *col);
```

cir Círculo.

pnt Punto.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_circle_segment

Colisión círculo-segmento.

```
bool_t
col2d_circle_segmentf(const Cir2Df *cir,
                      const Seg2Df *seg,
                      Col2Df *col);

bool_t
col2d_circle_segmentd(const Cir2Dd *cir,
                      const Seg2Dd *seg,
                      Col2Dd *col);

bool_t
Col2D::circle_segment(const Cir2D *cir,
                      const Seg2D *seg,
                      Col2D *col);
```

cir Círculo.

seg Segmento.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_circle_circle

Colisión círculo-círculo.

```
bool_t
col2d_circle_circlef(const Cir2Df *cir1,
                     const Cir2Df *cir2,
                     Col2Df *col);

bool_t
col2d_circle_circled(const Cir2Dd *cir1,
                     const Cir2Dd *cir2,
                     Col2Dd *col);

bool_t
Col2D::circle_circle(const Cir2D *cir1,
                     const Cir2D *cir2,
                     Col2D *col);
```

cir1 Primer círculo.

cir2 Segundo círculo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_box_point

Colisión caja-punto.

```
bool_t
col2d_box_pointf(const Box2Df *box,
                 const V2Df *pnt,
                 Col2Df *col);

bool_t
```

```
col2d_box_pointd(const Box2Dd *box,
                 const V2Dd *pnt,
                 Col2Dd *col);

bool_t
Col2D::box_point(const Box2D *box,
                 const V2D *pnt,
                 Col2D *col);
```

box Caja.

pnt Punto.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_box_segment

Colisión caja-segmento.

```
bool_t
col2d_box_segmentf(const Box2Df *box,
                   const Seg2Df *seg,
                   Col2Df *col);

bool_t
col2d_box_segmentd(const Box2Dd *box,
                   const Seg2Dd *seg,
                   Col2Dd *col);

bool_t
Col2D::box_segment(const Box2D *box,
                   const Seg2D *seg,
                   Col2D *col);
```

box Caja.

seg Segmento.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_box_circle

Colisión caja-círculo.

```
bool_t
col2d_box_circlef(const Box2Df *box,
                  const Cir2Df *cir,
                  Col2Df *col);

bool_t
col2d_box_circled(const Box2Dd *box,
                  const Cir2Dd *cir,
                  Col2Dd *col);

bool_t
Col2D::box_circle(const Box2D *box,
                  const Cir2D *cir,
                  Col2D *col);
```

box Caja.

cir Círculo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_box_box

Colisión caja-caja.

```
bool_t
col2d_box_boxf(const Box2Df *box1,
               const Box2Df *box2,
               Col2Df *col);

bool_t
col2d_box_boxd(const Box2Dd *box1,
               const Box2Dd *box2,
               Col2Dd *col);

bool_t
Col2D::box_box(const Box2D *box1,
               const Box2D *box2,
               Col2D *col);
```

box1 Primera caja.

box2 Segunda caja.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_obb_point

Colisión caja orientada-punto.

```
bool_t
col2d_obb_pointf(const OBB2Df *obb,
                 const V2Df *pnt,
                 Col2Df *col);

bool_t
col2d_obb_pointd(const OBB2Dd *obb,
                 const V2Dd *pnt,
                 Col2Dd *col);

bool_t
Col2D::obb_point(const OBB2D *obb,
                 const V2D *pnt,
                 Col2D *col);
```

obb Caja orientada.

pnt Punto.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_obb_segment

Colisión caja orientada-segmento.

```
bool_t
col2d_obb_segmentf(const OBB2Df *obb,
                   const Seg2Df *seg,
                   Col2Df *col);

bool_t
```

```
col2d_obb_segmentd(const OBB2Dd *obb,
                   const Seg2Dd *seg,
                   Col2Dd *col);

bool_t
Col2D::obb_segment(const OBB2D *obb,
                   const Seg2D *seg,
                   Col2D *col);
```

obb Caja orientada.

seg Segmento.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_obb_circle

Colisión caja orientada-circle.

```
bool_t
col2d_obb_circlef(const OBB2Df *obb,
                  const Cir2Df *cir,
                  Col2Df *col);

bool_t
col2d_obb_circled(const OBB2Dd *obb,
                  const Cir2Dd *cir,
                  Col2Dd *col);

bool_t
Col2D::obb_circle(const OBB2D *obb,
                  const Cir2D *cir,
                  Col2D *col);
```

obb Caja orientada.

cir Círculo.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_obb_box

Colisión caja orientada-caja.

```
bool_t
col2d_obb_boxf(const OBB2Df *obb,
               const Box2Df *box,
               Col2Df *col);

bool_t
col2d_obb_boxd(const OBB2Dd *obb,
               const Box2Dd *box,
               Col2Dd *col);

bool_t
Col2D::obb_box(const OBB2D *obb,
               const Box2D *box,
               Col2D *col);
```

obb Caja orientada.

box Caja alineada.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_obb_obb

Colisión caja orientada-caja orientada.

```
bool_t
col2d_obb_obbf(const OBB2Df *obb1,
               const OBB2Df *obb2,
               Col2Df *col);

bool_t
col2d_obb_obbd(const OBB2Dd *obb1,
               const OBB2Dd *obb2,
               Col2Dd *col);

bool_t
Col2D::obb_obb(const OBB2D *obb1,
               const OBB2D *obb2,
               Col2D *col);
```

obb1 Primera caja orientada.

obb2 Segunda caja orientada.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_point

Colisión triángulo-punto.

```
bool_t
col2d_tri_pointf(const Tri2Df *tri,
                 const V2Df *pnt,
                 Col2Df *col);

bool_t
col2d_tri_pointd(const Tri2Dd *tri,
                 const V2Dd *pnt,
                 Col2Dd *col);

bool_t
Col2D::tri_point(const Tri2D *tri,
                 const V2D *pnt,
                 Col2D *col);
```

tri Triángulo.

pnt Punto.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_segment

Colisión triángulo-segmento.

```
bool_t
col2d_tri_segmentf(const Tri2Df *tri,
                   const Seg2Df *seg,
                   Col2Df *col);

bool_t
```

```
col2d_tri_segmentd(const Tri2Dd *tri,
                   const Seg2Dd *seg,
                   Col2Dd *col);

bool_t
Col2D::tri_segment(const Tri2D *tri,
                   const Seg2D *seg,
                   Col2D *col);
```

tri Triángulo.

seg Segmento.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_circle

Colisión triángulo-círculo.

```
bool_t
col2d_tri_circlef(const Tri2Df *tri,
                  const Cir2Df *cir,
                  Col2Df *col);

bool_t
col2d_tri_circled(const Tri2Dd *tri,
                  const Cir2Dd *cir,
                  Col2Dd *col);

bool_t
Col2D::tri_circle(const Tri2D *tri,
                  const Cir2D *cir,
                  Col2D *col);
```

tri Triángulo.

cir Círculo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_box

Colisión triángulo-caja.

```
bool_t
col2d_tri_boxf(const Tri2Df *tri,
               const Box2Df *box,
               Col2Df *col);

bool_t
col2d_tri_boxd(const Tri2Dd *tri,
               const Box2Dd *box,
               Col2Dd *col);

bool_t
Col2D::tri_box(const Tri2D *tri,
               const Box2D *box,
               Col2D *col);
```

tri Triángulo.

box Caja alineada.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_obb

Colisión triángulo-caja orientada.

```
bool_t
col2d_tri_obbf(const Tri2Df *tri,
               const OBB2Df *obb,
               Col2Df *col);

bool_t
col2d_tri_obbd(const Tri2Dd *tri,
               const OBB2Dd *obb,
               Col2Dd *col);

bool_t
Col2D::tri_obb(const Tri2D *tri,
               const OBB2D *obb,
               Col2D *col);
```

tri Triángulo.

obb Caja orientada.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_tri_tri

Colisión triángulo-triángulo.

```
bool_t
col2d_tri_trif(const Tri2Df *tri1,
               const Tri2Df *tri2,
               Col2Df *col);

bool_t
col2d_tri_trid(const Tri2Dd *tri1,
               const Tri2Dd *tri2,
               Col2Dd *col);

bool_t
Col2D::tri_tri(const Tri2D *tri1,
               const Tri2D *tri2,
               Col2D *col);
```

tri1 Primer triángulo.

tri2 Segundo triángulo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_poly_point

Colisión polígono-punto.

```
bool_t
col2d_poly_pointf(const Pol2Df *pol,
                  const V2Df *pnt,
                  Col2Df *col);

bool_t
```

```
col2d_poly_pointd(const Pol2Dd *pol,
                  const V2Dd *pnt,
                  Col2Dd *col);

bool_t
Col2D::poly_point(const Pol2D *pol,
                  const V2D *pnt,
                  Col2D *col);
```

pol Polígono.

pnt Punto.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_poly_segment

Colisión polígono-segmento.

```
bool_t
col2d_poly_segmentf(const Pol2Df *pol,
                    const Seg2Df *seg,
                    Col2Df *col);

bool_t
col2d_poly_segmentd(const Pol2Dd *pol,
                    const Seg2Dd *seg,
                    Col2Dd *col);

bool_t
Col2D::poly_segment(const Pol2D *pol,
                    const Seg2D *seg,
                    Col2D *col);
```

pol Polígono.

seg Segmento.

col Datos pormenorizados de la colisión. Puede ser NULL si no necesitamos información adicional.

Retorna:

TRUE si los objectos intersectan, FALSE si no.

col2d_poly_circle

Colisión polígono-círculo.

```
bool_t
col2d_poly_circlef(const Pol2Df *pol,
                   const Cir2Df *cir,
                   Col2Df *col);

bool_t
col2d_poly_circled(const Pol2Dd *pol,
                   const Cir2Dd *cir,
                   Col2Dd *col);

bool_t
Col2D::poly_circle(const Pol2D *pol,
                   const Cir2D *cir,
                   Col2D *col);
```

pol Polígono.

cir Círculo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_poly_box

Colisión polígono-caja.

```
bool_t
col2d_poly_boxf(const Pol2Df *pol,
                const Box2Df *cir,
                Col2Df *col);

bool_t
col2d_poly_boxd(const Pol2Dd *pol,
                const Box2Dd *cir,
                Col2Dd *col);

bool_t
Col2D::poly_box(const Pol2D *pol,
                const Box2D *cir,
                Col2D *col);
```

pol Polígono.

cir Caja.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_poly_obb

Colisión polígono-caja.

```
bool_t
col2d_poly_obbf(const Pol2Df *pol,
                const OBB2Df *cir,
                Col2Df *col);

bool_t
col2d_poly_obbd(const Pol2Dd *pol,
                const OBB2Dd *cir,
                Col2Dd *col);

bool_t
Col2D::poly_obb(const Pol2D *pol,
                const OBB2D *cir,
                Col2D *col);
```

pol Polígono.

cir Caja orientada.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_poly_tri

Colisión polígono-triángulo.

```
bool_t
col2d_poly_trif(const Pol2Df *pol,
                const Tri2Df *tri,
                Col2Df *col);

bool_t
```

```
col2d_poly_trid(const Pol2Dd *pol,
                const Tri2Dd *tri,
                Col2Dd *col);

bool_t
Col2D::poly_tri(const Pol2D *pol,
                const Tri2D *tri,
                Col2D *col);
```

pol Polígono.

tri Triángulo.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

col2d_poly_poly

Colisión polígono-polígono.

```
bool_t
col2d_poly_polyf(const Pol2Df *pol1,
                 const Pol2Df *pol2,
                 Col2Df *col);

bool_t
col2d_poly_polyd(const Pol2Dd *pol1,
                 const Pol2Dd *pol2,
                 Col2Dd *col);

bool_t
Col2D::poly_poly(const Pol2D *pol1,
                 const Pol2D *pol2,
                 Col2D *col);
```

pol1 Primer polígono.

pol2 Segundo polígono.

col Datos pormenorizados de la colisión. Puede ser `NULL` si no necesitamos información adicional.

Retorna:

`TRUE` si los objectos intersectan, `FALSE` si no.

Capítulo 39

Librería Draw2D

39.1. Tipos y Constantes

kCOLOR_TRANSPARENT

Color totalmente transparente o ausencia de color.

```
const color_t kCOLOR_TRANSPARENT;
```

kCOLOR_DEFAULT

Color por defecto.

```
const color_t kCOLOR_DEFAULT;
```

kCOLOR_BLACK

Color NEGRO rgb(0,0,0).

```
const color_t kCOLOR_BLACK;
```

kCOLOR_WHITE

Color BLANCO rgb(255,255,255).

```
const color_t kCOLOR_WHITE;
```

kCOLOR_RED

Color ROJO rgb(255,0,0).

```
const color_t kCOLOR_RED;
```

kCOLOR_GREEN

Color VERDE rgb(0,255,0).

```
const color_t kCOLOR_GREEN;
```

kCOLOR_BLUE

Color AZUL rgb(0,0,255).

```
const color_t kCOLOR_BLUE;
```

kCOLOR_YELLOW

Color AMARILLO rgb(255,255,0).

```
const color_t kCOLOR_YELLOW;
```

kCOLOR_CYAN

Color CIANO rgb(0,255,255).

```
const color_t kCOLOR_CYAN;
```

kCOLOR_MAGENTA

Color MAGENTA rgb(255,0,255).

```
const color_t kCOLOR_MAGENTA;
```

enum pixformat_t

"Formatos de píxelFormatos de píxel" (Página 287) en una imagen. Número de bits y modelo de color.

ekINDEX1	1 bit por pixel. 2 colores, indexado.
ekINDEX2	2 bits por pixel. 4 colores, indexado.
ekINDEX4	4 bits por pixel. 16 colores, indexado.
ekINDEX8	8 bits por pixel. 256 colores, indexado.
ekGRAY8	8 bits por pixel en escala de grises. 256 tonos de gris.

`ekRGB24` 24 bits por pixel RGB. 8 bits por canal (rojo, verde, azul). El byte de menor orden corresponde al rojo y el de mayor al azul.

`ekRGBA32` 32 bits por pixel RGBA. 8 bits por canal (rojo, verde, azul, alpha). El byte de menor orden corresponde al rojo y el de mayor al alpha (transparencia).

`ekFIMAGE` Representa el formato original de la imagen. Sólo válido en `image_pixels`.

enum codec_t

Formato de codificación y compresión de imágenes.

`ekJPG` *Joint Photographic Experts Group.*

`ekPNG` *Portable Network Graphics.*

`ekBMP` *BitMaP.*

`ekGIF` *Graphics Interchange Format.*

enum fstyle_t

Estilo en fuentes tipográficas. Pueden combinarse varios valores con el operator OR `('|')`.

`ekFNORMAL` Fuente normal, sin estilo. También llamada *Regular*.

`ekFBOLD` Fuente **en negrita**.

`ekFITALIC` Fuente *en itálica*.

`ekFSTRIKEOUT` Fuente ~~tachada~~.

`ekFUNDERLINE` Fuente subrayada.

`ekFSUBSCRIPT` Subíndice. Ver `textview_fstyle`.

`ekFSUPSCRIPT` Superíndice. Ver `textview_fstyle`.

`ekFPIXELS` Los tamaños de fuente se indicarán en píxeles.

`ekFPOINTS` Los tamaños de fuente se indicarán en puntos. *"Tamaño en puntosTamaño en puntos"* (Página 298).

enum linecap_t

Estilo en los extremos de línea.

`ekLCFLAT`	Terminación plana en el último punto de la línea.
`ekLCSQUARE`	Terminación en un cuadro, cuyo centro es el último punto de la línea.
`ekLCROUND`	Terminación en círculo, cuyo centro es el último punto de la línea.

enum linejoin_t

Estilo en las uniones de línea.

`ekLJMITER`	Unión en ángulo. En ángulos muy cerrados se recorta.
`ekLJROUND`	Unión redondeada.
`ekLJBEVEL`	Unión biselada.

enum fillwrap_t

Comportamiento del patrón de relleno en los límites.

`ekFCLAMP`	Se utiliza el último valor del límite para rellenar el área exterior.
`ekFTILE`	Se repite el patrón.
`ekFFLIP`	Se repite el patrón, invirtiendo el orden.

enum drawop_t

Operación a realizar sobre primitivas gráficas.

`ekSTROKE`	Dibuja el contorno de la figura con el estilo de línea predeterminado.
`ekFILL`	Rellena el interior de la figura con el color o patrón predeterminado.
`ekSKFILL`	Primero dibuja el contorno y después rellena.
`ekFILLSK`	Primero rellena y después dibuja el contorno.

enum align_t

Valores de alineación.

`ekLEFT`	Alineación al margen izquierdo.

`ekTOP`	Alineación al margen superior.
`ekCENTER`	Alineación centrada.
`ekRIGHT`	Alineación al margen derecho.
`ekBOTTOM`	Alineación al margen inferior.
`ekJUSTIFY`	Justificación o expansión del contenido.

enum ellipsis_t

Posición de la elipsis (...) al cortar un texto.

`ekELLIPNONE`	Sin elipsis.
`ekELLIPBEGIN`	Elipsis al principio del texto.
`ekELLIPMIDDLE`	Elipsis en el centro del texto.
`ekELLIPEND`	Elipsis al final del texto.
`ekELLIPMLINE`	Texto multilínea (sin elipsis).

struct color_t

Entero de 32 bits que representa un color RGBA. El byte de menor orden corresponde al canal rojo (Red) y el de mayor orden al canal Alpha (transparencia). *"Colores"* (Página 283).

```
struct color_t;
```

struct DCtx

Contexto de dibujo 2D, destinatario para los comandos de dibujo. También se conoce como lienzo *(canvas)* o superficie *(surface)*. *"Contextos 2D"* (Página 263).

```
struct DCtx;
```

struct Draw

Dibujo de entidades geométricas.

```
struct Drawf;

struct Drawd;

struct Draw;
```

struct Palette

Paleta de colores, normalmente relacionada con `Pixbuf` indexados. *"Paletas"* (Página 285).

```
struct Palette;
```

struct Pixbuf

Búfer en memoria con información de píxeles. *"Pixel Buffer"* (Página 286).

```
struct Pixbuf;
```

struct Image

Contiene una imagen de mapa de bits, compuesta por píxeles. *"Imágenes"* (Página 290).

```
struct Image;
```

struct Font

Contiene la familia tipográfica, tamaño y estilo con el que se dibujarán los textos. *"Fuentes tipográficas"* (Página 295).

```
struct Font;
```

39.2. Funciones

draw2d_start

Inicia la librería *draw2d*, reservando espacio para las estructuras internas globales. Internamente llama a `core_start`. En aplicaciones de escritorio, `osmain` llama a esta función al iniciar el programa.

```
void
draw2d_start(void);
```

draw2d_finish

Finaliza la librería *draw2d*, liberando el espacio de las estructuras internas globales. Internamente llama a `core_finish`. En aplicaciones de escritorio, `osmain` llama a esta función al salir del programa.

```
void
draw2d_finish(void);
```

resid_image

Realiza un casting ResId-Image.

```
const Image*
resid_image(const ResId id);
```

id El identificador del recurso.

Retorna:

El identificador de la imagen.

dctx_bitmap

Crea un contexto en memoria, con el fin de generar una imagen.

```
DCtx*
dctx_bitmap(const uint32_t width,
            const uint32_t height,
            const pixformat_t format);
```

width Ancho de la imagen en píxeles.

height Alto de la imagen en píxeles.

format Formato de píxel de la imagen generada.

Retorna:

El contexto de dibujo.

Observaciones:

Al terminar de dibujar, debemos llamar a `dctx_image` para obtener la imagen.

dctx_image

Obtiene la imagen resultado tras dibujar en el contexto creado con `dctx_bitmap`.

```
Image*
dctx_image(DCtx **ctx);
```

ctx El contexto, que será destruido tras generar la imagen.

Retorna:

La imagen.

draw_clear

Borra todo el área del contexto, utilizando un color plano.

```
void
draw_clear(DCtx *ctx,
           const color_t color);
```

ctx Contexto de dibujo.

color Color de fondo.

draw_matrix

Establece el sistema de referencia del contexto (transformación afín).

```
void
draw_matrixf(DCtx *ctx,
             const T2Df *t2d);

void
draw_matrixd(DCtx *ctx,
             const T2Dd *t2d);

void
Draw::matrix(DCtx *ctx,
             const T2D *t2d);
```

ctx Contexto de dibujo.

t2d Transformación.

Observaciones:

El origen de coordenadas está en la esquina superior izquierda. El eje Y aumenta hacia abajo.

draw_matrix_cartesian

Establece el sistema de referencia en coordenadas cartesianas.

```
void
draw_matrix_cartesianf(DCtx *ctx,
                       const T2Df *t2d);

void
draw_matrix_cartesiand(DCtx *ctx,
                       const T2Dd *t2d);

void
```

```
Draw::matrix_cartesian(DCtx *ctx,
                       const T2D *t2d);
```

ctx Contexto de dibujo.

t2d Transformación.

Observaciones:

El origen de coordenadas está en la esquina inferior izquierda. El eje Y aumenta hacia arriba. Ver *"Sistemas cartesianosSistemas cartesianos"* (Página 268).

draw_antialias

Activa o desactiva el antialiasing.

```
void
draw_antialias(DCtx *ctx,
               const bool_t on);
```

ctx Contexto de dibujo.

on `TRUE` activo, `FALSE` inactivo.

Observaciones:

El antialias puede cambiar en cada primitiva. No es necesario establecer una política para todo el dibujo. Ver *"AntialiasingAntialiasing"* (Página 269).

draw_line

Dibuja una línea.

```
void
draw_line(DCtx *ctx,
          const real32_t x0,
          const real32_t y0,
          const real32_t x1,
          const real32_t y1);
```

ctx Contexto de dibujo.

x0 Coordenada x del primer punto.

y0 Coordenada y del primer punto.

x1 Coordenada x del segundo punto.

y1 Coordenada y del segundo punto.

draw_polyline

Dibuja varias líneas unidas.

```
void
draw_polyline(DCtx *ctx,
              const bool_t closed,
              const V2Df *points,
              const uint32_t n);
```

ctx	Contexto de dibujo.
closed	`TRUE` para unir el último punto con el primero.
points	Array de puntos que forman la polilínea.
n	Número de puntos.

draw_arc

Dibuja un arco (segmento circular).

```
void
draw_arc(DCtx *ctx,
         const real32_t x,
         const real32_t y,
         const real32_t radius,
         const real32_t start,
         const real32_t sweep);
```

ctx	Contexto de dibujo.
x	Coordenada x del centro del arco.
y	Coordenada y del centro del arco.
radius	Radio del arco.
start	Ángulo inicial con respecto al vector `X=[1,0]` en radianes.
sweep	Ángulo de barrido o tamaño del arco en radianes.

Observaciones:

Los ángulos positivos son los que giran desde el vector X al vector Y. Ver *"Vectores 2D"* (Página 242).

draw_bezier

Dibuja una curva de Bézier cúbica (grado 3) utilizando dos extremos (x0,y0)-(x3,y3) y dos puntos de control intermedios (x1,y1)-(x2,y2).

```
void
draw_bezier(DCtx *ctx,
            const real32_t x0,
            const real32_t y0,
            const real32_t x1,
            const real32_t y1,
            const real32_t x2,
            const real32_t y2,
            const real32_t x3,
            const real32_t y3);
```

ctx Contexto de dibujo.

x0 Coordenada x del punto inicial.

y0 Coordenada y del punto inicial.

x1 Coordenada x del primer punto intermedio.

y1 Coordenada y del primer punto intermedio.

x2 Coordenada x del segundo punto intermedio.

y2 Coordenada y del segundo punto intermedio.

x3 Coordenada x del punto final.

y3 Coordenada y del punto final.

draw_line_color

Establece el color de dibujo de líneas y contornos.

```
void
draw_line_color(DCtx *ctx,
                const color_t color);
```

ctx Contexto de dibujo.

color Color de línea.

draw_line_fill

Establece el patrón de relleno actual para el dibujo de líneas.

```
void
draw_line_fill(DCtx *ctx);
```

ctx Contexto de dibujo.

Observaciones:

El patrón de relleno debe haber sido establecido previamente mediante `draw_fill_linear`. Ver *"Gradientes en líneasGradientes en líneas"* (Página 276).

draw_line_width

Establece el grosor de línea.

```
void
draw_line_width(DCtx *ctx,
                const real32_t width);
```

ctx	Contexto de dibujo.
width	Grosor de línea.

draw_line_cap

Establece el estilo de los extremos de línea.

```
void
draw_line_cap(DCtx *ctx,
              const linecap_t cap);
```

ctx	Contexto de dibujo.
cap	Estilo.

draw_line_join

Establece el estilo de las uniones de línea.

```
void
draw_line_join(DCtx *ctx,
               const linejoin_t join);
```

ctx	Contexto de dibujo.
join	Estilo de la unión.

draw_line_dash

Establece un patrón para el dibujo de líneas.

```
void
draw_line_dash(DCtx *ctx,
               const real32_t *pattern,
               const uint32_t n);
```

ctx	Contexto de dibujo.
pattern	Array de valores que definen el patrón.
n	Número de valores.

Observaciones:

El primer elemento de `pattern` define la longitud del primer trazo y el segundo la del primer hueco. Así sucesivamente. Las longitudes se escalan por el grosor de línea `draw_line_width`, es decir, un trazo de longitud 1 dibujará un cuadrado de lado *line_width*. Longitudes de valor 2 equivalen al doble del grosor de línea, etc. El patrón escalará proporcionalmente al cambiar el grosor o hacer zoom mediante transformaciones.

draw_rect

Dibuja un rectángulo.

```
void
draw_rect(DCtx *ctx,
          const drawop_t op,
          const real32_t x,
          const real32_t y,
          const real32_t width,
          const real32_t height);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
x	Coordenada x de la esquina superior izquierda del rectángulo.
y	Coordenada y de la esquina superior izquierda del rectángulo.
width	Ancho del rectángulo.
height	Alto del rectángulo.

Observaciones:

En *"Sistemas cartesianosSistemas cartesianos"* (Página 268) (x,y) indican el origen de la esquina inferior izquierda.

draw_rndrect

Dibuja un rectángulo con los bordes redondeados.

```
void
draw_rndrect(DCtx *ctx,
             const drawop_t op,
             const real32_t x,
```

```
                const real32_t y,
                const real32_t width,
                const real32_t height,
                const real32_t radius);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
x	Coordenada x de la esquina superior izquierda.
y	Coordenada y de la esquina superior izquierda.
width	Ancho del rectángulo.
height	Alto del rectángulo.
radius	Radio de curvatura de la esquina.

Observaciones:

En *"Sistemas cartesianosSistemas cartesianos"* (Página 268) (x,y) indican el origen de la esquina inferior izquierda.

draw_circle

Dibuja un círculo.

```
void
draw_circle(DCtx *ctx,
            const drawop_t op,
            const real32_t x,
            const real32_t y,
            const real32_t radius);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
x	Coordenada x del centro.
y	Coordenada y del centro.
radius	Radio.

draw_ellipse

Dibuja una elipse.

```
void
draw_ellipse(DCtx *ctx,
             const drawop_t op,
```

```
              const real32_t x,
              const real32_t y,
              const real32_t radx,
              const real32_t rady);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
x	Coordenada x del centro.
y	Coordenada y del centro.
radx	Radio en el eje X.
rady	Radio en el eje Y.

draw_polygon

Dibuja un polígono.

```
void
draw_polygon(DCtx *ctx,
             const drawop_t op,
             const V2Df *points,
             const uint32_t n);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
points	Array de puntos que forman el polígono.
n	Número de puntos.

draw_fill_color

Establece un color sólido para el relleno de figuras.

```
void
draw_fill_color(DCtx *ctx,
                const color_t color);
```

ctx	Contexto de dibujo.
color	Color de relleno.

draw_fill_linear

Establece un gradiente (o degradado) para el relleno de figuras.

```
void
draw_fill_linear(DCtx *ctx,
                 const color_t *color,
                 const real32_t *stop,
                 const uint32_t n,
                 const real32_t x0,
                 const real32_t y0,
                 const real32_t x1,
                 const real32_t y1);
```

ctx	Contexto de dibujo.
color	Array de colores.
stop	Posiciones de los colores.
n	Número de posiciones/colores.
x0	Coordenada x del punto inicial.
y0	Coordenada y del punto inicial.
x1	Coordenada x del punto final.
y1	Coordenada y del punto final.

Observaciones:

Las posiciones deben ir del valor 0 al 1. Ver *"GradientesGradientes"* (Página 274).

draw_fill_matrix

Establece la matriz de transformación del patrón de relleno.

```
void
draw_fill_matrix(DCtx *ctx,
                 const T2Df *t2d);
```

ctx	Contexto de dibujo.
t2d	Transformación.

Observaciones:

Solo será efectiva en rellenos no sólidos. Ver *"GradientesGradientes"* (Página 274).

draw_fill_wrap

Establece el comportamiento del gradiente o patrón de relleno en los límites.

```
void
draw_fill_wrap(DCtx *ctx,
               const fillwrap_t wrap);
```

ctx Contexto de dibujo.

wrap Comportamiento en el borde del relleno.

Observaciones:

Solo será efectiva en rellenos no sólidos. Ver *"GradientesGradientes"* (Página 274).

draw_font

Establece la fuente para el dibujado de textos.

```
void
draw_font(DCtx *ctx,
          const Font *font);
```

ctx Contexto de dibujo.

font Fuente tipográfica.

Observaciones:

Tendrá efecto a partir del siguiente texto dibujado. Ver *"Fuentes tipográficas"* (Página 295).

draw_text_color

Establece el color del texto.

```
void
draw_text_color(DCtx *ctx,
                const color_t color);
```

ctx Contexto de dibujo.

color Color.

draw_text

Dibuja un bloque de texto.

```
void
draw_text(DCtx *ctx,
          const char_t *text,
          const real32_t x,
```

```
          const real32_t y);
```

ctx Contexto de dibujo.

text Cadena UTF8, terminada en carácter nulo `'\0'`.

x Coordenada x en el lienzo del origen del texto.

y Coordenada y en el lienzo del origen del texto.

Observaciones:

El texto se dibujará con la fuente y color preestablecido y será sensible a la transformación del contexto (rotaciones, escalados, etc). Ver *"Dibujar textoDibujar texto"* (Página 278).

draw_text_path

Dibuja un bloque de texto como un área geométrica. Similar a `draw_text`, pero permite utilizar gradientes o dibujar sólo el borde del texto.

```
void
draw_text_path(DCtx *ctx,
               const drawop_t op,
               const char_t *text,
               const real32_t x,
               const real32_t y);
```

ctx Contexto de dibujo.

op Operación de dibujo.

text Cadena UTF8, terminada en carácter nulo `'\0'`.

x Coordenada x en el lienzo del origen del texto.

y Coordenada y en el lienzo del origen del texto.

Observaciones:

El texto se dibujará con la fuente y estilo (relleno y línea) preestablecido y será sensible a la transformación del contexto. Ver *"Dibujar textoDibujar texto"* (Página 278).

draw_text_width

Establece el ancho máximo de los bloques de texto.

```
void
draw_text_width(DCtx *ctx,
                const real32_t width);
```

ctx	Contexto de dibujo.
width	Anchura máxima.

Observaciones:

Si el texto a dibujar con `draw_text` es más ancho que `width`, se fragmentará en varias líneas. Pasar -1 para dibujar todo el bloque en una sola línea. No se tiene en cuenta el escalado del contexto. La medición se realiza en función del tamaño de la fuente preestablecida. Ver *"Dibujar textoDibujar texto"* (Página 278).

draw_text_trim

Establece como el texto será recortado cuando sea más ancho que el valor de `draw_text_width` .

```
void
draw_text_trim(DCtx *ctx,
               const ellipsis_t ellipsis);
```

ctx	Contexto de dibujo.
ellipsis	Estilo de recorte.

draw_text_align

Establece la alineación del texto con respecto al punto de inserción.

```
void
draw_text_align(DCtx *ctx,
                const align_t halign,
                const align_t valign);
```

ctx	Contexto de dibujo.
halign	Alineación horizontal.
valign	Alineación vertical.

Observaciones:

El punto de inserción es la coordenada `(x,y)` de `draw_text`. Ver *"Dibujar textoDibujar texto"* (Página 278).

draw_text_halign

Establece la alineación horizontal interna del texto, dentro de un bloque multilínea.

```
void
draw_text_halign(DCtx *ctx,
                 const align_t halign);
```

ctx	Contexto de dibujo.
halign	Alineación horizontal.

Observaciones:

En textos de una sola línea, no tiene efecto. Ver *"Dibujar textoDibujar texto"* (Página 278).

draw_text_extents

Calcula el tamaño que ocupa un bloque de texto.

```
void
draw_text_extents(DCtx *ctx,
                  const char_t *text,
                  const real32_t refwidth,
                  real32_t *width,
                  real32_t *height);
```

ctx	Contexto de dibujo.
text	Texto.
refwidth	Ancho máximo de referencia.
width	Ancho del bloque.
height	Alto del bloque.

Observaciones:

Si `refwidth` es mayor de `0`, `width` estará acotado por este valor y `height` se expandirá hasta dar cabida a todo el texto. Tiene en cuenta los posible saltos de línea `'\n'` de `text`.

draw_image

Dibuja una imagen.

```
void
draw_image(DCtx *ctx,
           const real32_t x,
           const real32_t y);
```

ctx Contexto de dibujo.

x Coordenada x en el lienzo del origen de la imagen.

y Coordenada y en el lienzo del origen de la imagen.

Observaciones:

La imagen se dibujará a su tamaño natural y en la posición indicada. Utilizar `draw_matrixf` para realizar escalados y rotaciones. Ver *"Dibujar imágenesDibujar imágenes"* (Página 281).

draw_image_frame

Igual que `draw_image`, pero indicando el número de secuencia de una animación.

```
void
draw_image_frame(DCtx *ctx,
                 const uint32_t frame,
                 const real32_t x,
                 const real32_t y);
```

ctx Contexto de dibujo.

frame Índice de la secuencia *(frame)* de la animación.

x Coordenada x en el lienzo del origen de la imagen.

y Coordenada y en el lienzo del origen de la imagen.

Observaciones:

Únicamente imágenes creadas a partir de un archivo **GIF** soportan múltiples frames (animaciones). Ver `image_num_frames`.

draw_image_align

Establece la alineación de la imagen con respecto al punto de inserción.

```
void
draw_image_align(DCtx *ctx,
                 const align_t halign,
                 const align_t valign);
```

ctx Contexto de dibujo.

halign Alineación horizontal.

valign Alineación vertical.

Observaciones:

El punto de inserción es la coordenada `(x,y)` de `draw_image`. Ver *"Dibujar imágenes-Dibujar imágenes"* (Página 281).

draw_v2d

Dibuja un punto 2D.

```
void
draw_v2df(DCtx *ctx,
          const drawop_t op,
          const V2Df *v2d,
          const real32_t radius);

void
draw_v2dd(DCtx *ctx,
          const drawop_t op,
          const V2Dd *v2d,
          const real64_t radius);

void
Draw::v2d(DCtx *ctx,
          const drawop_t op,
          const V2D *v2d,
          const real radius);
```

ctx	Contexto de dibujo.
op	Operación de dibujo.
v2d	Punto.
radius	Radio.

draw_seg2d

Dibuja un segmento 2D.

```
void
draw_seg2df(DCtx *ctx,
            const Seg2Df *seg);

void
draw_seg2dd(DCtx *ctx,
            const Seg2Dd *seg);

void
Draw::seg2d(DCtx *ctx,
            const Seg2D *seg);
```

ctx Contexto de dibujo.

seg Segmento.

draw_cir2d

Dibuja un círculo 2D.

```
void
draw_cir2df(DCtx *ctx,
            const drawop_t op,
            const Cir2Df *cir);

void
draw_cir2dd(DCtx *ctx,
            const drawop_t op,
            const Cir2Dd *cir);

void
Draw::cir2d(DCtx *ctx,
            const drawop_t op,
            const Cir2D *cir);
```

ctx Contexto de dibujo.

op Operación de dibujo.

cir Círculo.

draw_box2d

Dibuja una caja 2D.

```
void
draw_box2df(DCtx *ctx,
            const drawop_t op,
            const Box2Df *box);

void
draw_box2dd(DCtx *ctx,
            const drawop_t op,
            const Box2Dd *box);

void
Draw::box2d(DCtx *ctx,
            const drawop_t op,
            const Box2D *box);
```

ctx Contexto de dibujo.

op Operación de dibujo.

box Caja alineada.

draw_obb2d

Dibuja una caja 2D orientada.

```
void
draw_obb2df(DCtx *ctx,
            const drawop_t op,
            const OBB2Df *obb);

void
draw_obb2dd(DCtx *ctx,
            const drawop_t op,
            const OBB2Dd *obb);

void
Draw::obb2d(DCtx *ctx,
            const drawop_t op,
            const OBB2D *obb);
```

ctx Contexto de dibujo.

op Operación de dibujo.

obb Caja orientada.

draw_tri2d

Dibuja un triángulo 2D.

```
void
draw_tri2df(DCtx *ctx,
            const drawop_t op,
            const Tri2Df *tri);

void
draw_tri2dd(DCtx *ctx,
            const drawop_t op,
            const Tri2Dd *tri);

void
Draw::tri2d(DCtx *ctx,
            const drawop_t op,
            const Tri2D *tri);
```

ctx Contexto de dibujo.

op Operación de dibujo.

tri Triángulo.

draw_pol2d

Dibuja un polígono 2D.

```
void
draw_pol2df(DCtx *ctx,
            const drawop_t op,
            const Pol2Df *pol);

void
draw_pol2dd(DCtx *ctx,
            const drawop_t op,
            const Pol2Dd *pol);

void
Draw::pol2d(DCtx *ctx,
            const drawop_t op,
            const Pol2D *pol);
```

ctx Contexto de dibujo.

op Operación de dibujo.

pol Polígono.

color_rgb

Crea un color a partir de los canales **R** (red), **G** (green) y **B** (blue).

```
color_t
color_rgb(const uint8_t r,
          const uint8_t g,
          const uint8_t b);
```

r El canal rojo.

g El canal verde.

b El canal azul.

Retorna:

Color.

Observaciones:

El canal alpha se establece a 255 (totalmente opaco).

color_rgba

Crea un color a partir de los canales **R** (red), **G** (green), **B** (blue) y **A** (alpha).

```
color_t
color_rgba(const uint8_t r,
           const uint8_t g,
           const uint8_t b,
           const uint8_t a);
```

r El canal rojo.

g El canal verde.

b El canal azul.

a El canal alpha (transparencia).

Retorna:

Color.

Observaciones:

No se admite `a=0`, utilizar `kCOLOR_TRANSPARENT` en esos casos.

color_rgbaf

Crea un color a partir de los canales RGBA normalizados de 0 a 1.

```
color_t
color_rgbaf(const real32_t r,
            const real32_t g,
            const real32_t b,
            const real32_t a);
```

r El canal rojo.

g El canal verde.

b El canal azul.

a El canal alpha (transparencia).

Retorna:

Color.

Observaciones:

No se admite `a=0`, utilizar `kCOLOR_TRANSPARENT` en esos casos.

color_hsbf

Crea un color (rgb) a partir de sus componentes Hue-Saturation-Brightness (Tono-Saturación-Brillo).

```
color_t
color_hsbf(const real32_t hue,
           const real32_t sat,
           const real32_t bright);
```

hue	Componente Hue (brillo).
sat	Componente Saturation (saturación).
bright	Componente Brightness (brillo).

Retorna:

Color.

color_red

Crea un color RGB utilizando solo el canal rojo.

```
color_t
color_red(const uint8_t r);
```

r	Canal rojo.

Retorna:

Color.

Observaciones:

Equivalente a `color_rgb(r, 0, 0)`.

color_green

Crea un color RGB utilizando solo el canal verde.

```
color_t
color_green(const uint8_t g);
```

g	Canal verde.

Retorna:

Color.

Observaciones:

Equivalente a `color_rgb(0, g, 0)`.

color_blue

Crea un color RGB utilizando solo el canal azul.

```
color_t
color_blue(const uint8_t b);
```

b Canal azul.

Retorna:

Color.

Observaciones:

Equivalente a `color_rgb(0, 0, b)`.

color_gray

Crea un color RGB gris a partir del valor de intensidad.

```
color_t
color_gray(const uint8_t l);
```

l Intensidad (luminancia).

Retorna:

Color.

Observaciones:

Equivalente a `color_rgb(l, l, l)`.

color_bgr

Crea un color a partir de un valor de 32 bits BGR. El byte 0 corresponde al canal **B**, el 1 al **G** y el 2 al **R**. El byte de mayor orden se ignora (se fija a 255).

```
color_t
color_bgr(const uint32_t bgr);
```

bgr El valor bgr de 32bits.

Retorna:

Color.

Observaciones:

Este orden de bytes es el típico en colores Web.

color_html

Crea un color a partir de una cadena en formato HTML o CSS.

```
color_t
color_html(const char_t *html);
```

```
color_t c1 = color_html("#FF0000"); // Red
color_t c2 = color_html("#000080"); // Navy
```

html La cadena de texto con el color HTML.

Retorna:

El color transformado a RGB.

color_to_hsbf

Convierte un color (rgb) al espacio HSB (hue, saturation, brightness).

```
void
color_to_hsbf(const color_t color,
              real32_t *hue,
              real32_t *sat,
              real32_t *sat);
```

color Color.

hue Componente Hue (tono).

sat Componente Saturación.

sat Componente Brightness (brillo).

color_to_html

Convierte un color al formato HTML o CSS (#RRGGBB).

```
void
color_to_html(const color_t color,
              char_t *html,
              const uint32_t size);
```

color	El color a convertir.
html	Búfer donde escribir el resultado.
size	Tamaño del búfer resultado.

color_get_rgb

Devuelve los valores RGB del color.

```
void
color_get_rgb(const color_t color,
              uint8_t *r,
              uint8_t *g,
              uint8_t *b);
```

color	Color.
r	El canal rojo.
g	El canal verde.
b	El canal azul.

Observaciones:

En colores de sistema o indexados, hace efectivo el valor RGB.

color_get_rgbf

Devuelve los valores RGB del color, normalizados de 0 a 1.

```
void
color_get_rgbf(const color_t color,
               real32_t *r,
               real32_t *g,
               real32_t *b);
```

color Color.

r El canal rojo.

g El canal verde.

b El canal azul.

Observaciones:

En colores de sistema o indexados, hace efectivo el valor RGB.

color_get_rgba

Devuelve los valores RGBA del color.

```
void
color_get_rgba(const color_t color,
               uint8_t *r,
               uint8_t *g,
               uint8_t *b,
               uint8_t *a);
```

color Color.

r El canal rojo.

g El canal verde.

b El canal azul.

a El canal alfa (transparencia).

Observaciones:

En colores de sistema o indexados, hace efectivo el valor RGBA.

color_get_rgbaf

Devuelve los valores RGBA del color, normalizados de 0 a 1.

```
void
color_get_rgbaf(const color_t color,
                real32_t *r,
                real32_t *g,
                real32_t *b,
                real32_t *a);
```

color Color.

r El canal rojo.

g El canal verde.

b El canal azul.

a El canal alfa (transparencia).

Observaciones:

En colores de sistema o indexados, hace efectivo el valor RGBA.

color_get_alpha

Extrae el componente alfa (transparencia) del color.

```
uint8_t
color_get_alpha(const color_t color);
```

color Color.

Retorna:

El componente alfa. Si es igual 0 significa que el color es indexado (no contiene valores RGB).

color_set_alpha

Cambia el valor alfa (transparencia) de un color.

```
color_t
color_set_alpha(const color_t color,
                const uint8_t alpha);
```

color Color.

alpha Componente alfa.

Retorna:

El nuevo color, con la componente alfa alterada.

palette_create

Crea una paleta.

```
Palette*
palette_create(const uint32_t size);
```

size El número de colores.

Retorna:

La paleta. El contenido inicial es indeterminado. Editar con `palette_colors`.

palette_cga2

Crea la paleta de 4 colores (2 bits) de las tarjetas CGA.

```
Palette*
palette_cga2(const bool_t mode,
             const bool_t intense);
```

mode `TRUE` para el modo 1 de CGA, `FALSE` modo 0.

intense `TRUE` para colores brillantes.

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_ega4

Crea la paleta por defecto de las tarjetas EGA (16 colores, 4 bits).

```
Palette*
palette_ega4(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_rgb8

Crea la paleta de por defecto RGB de 8 bits. Los colores combinan 8 tonos de rojo, 8 de verde y 4 de azul.

```
Palette*
palette_rgb8(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_gray1

Crea una paleta de 2 tonos de gris (1 bit). Negro (0) y blanco (1).

```
Palette*
palette_gray1(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_gray2

Crea una paleta de 4 tonos de gris (2 bit). Negro (0), blanco (3).

```
Palette*
palette_gray2(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_gray4

Crea una paleta de 16 tonos de gris (4 bit). Negro (0), blanco (15).

```
Palette*
palette_gray4(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_gray8

Crea una paleta de 256 tonos de gris (8 bit). Negro (0), blanco (255).

```
Palette*
palette_gray8(void);
```

Retorna:

La paleta.

Observaciones:

"Paleta predefinidaPaleta predefinida" (Página 286)

palette_binary

Crea una paleta de dos colores.

```
Palette*
palette_binary(const color_t zero,
               const color_t one);
```

zero Color asociado al valor `0`.

one Color asociado al valor `1`.

Retorna:

La paleta.

palette_destroy

Destruye la paleta.

```
void
palette_destroy(Palette **palette);
```

palette La paleta. Será puesto a `NULL` tras la destrucción.

palette_size

Retorna el número de colores de la paleta.

```
uint32_t
palette_size(const Palette *palette);
```

palette La paleta.

Retorna:

El número de colores.

palette_colors

Obtiene la lista de colores.

```
color_t*
palette_colors(Palette *palette);
```

palette La paleta.

Retorna:

Colores. El tamaño del array viene dado por `palette_size`.

Observaciones:

El búfer es de lectura/escritura.

palette_colors_const

Obtiene la lista de colores.

```
const color_t*
palette_colors_const(const Palette *palette);
```

palette La paleta.

Retorna:

Colores. El tamaño del array viene dado por `palette_size`.

pixbuf_create

Crea un nuevo pixel búfer.

```
Pixbuf*
pixbuf_create(const uint32_t width,
              const uint32_t height,
              const pixformat_t format);
```

width	Anchura.
height	Altura.
format	Formato de píxel.

Retorna:

El píxel búfer.

Observaciones:

El contenido inicial estará indeterminado.

pixbuf_copy

Crea una copia del pixel búfer.

```
Pixbuf*
pixbuf_copy(const Pixbuf *pixbuf);
```

pixbuf	El búfer original.

Retorna:

La copia.

pixbuf_trim

Recorta un pixel búfer.

```
Pixbuf*
pixbuf_trim(const Pixbuf *pixbuf,
            const uint32_t x,
            const uint32_t y,
            const uint32_t width,
            const uint32_t height);
```

pixbuf	El búfer original.
x	Coordenada x del pixel superior-izquierda.
y	Coordenada y del pixel superior-izquierda.
width	Número de píxeles de ancho.
height	Número de píxeles de alto.

Retorna:

Un nuevo pixel búfer con el recorte.

Observaciones:

La función no comprueba que los límites sean válidos. Obtendrás un error de segmentación en dichos casos.

pixbuf_convert

Cambia el formato de un píxel búfer.

```
Pixbuf*
pixbuf_convert(const Pixbuf *pixbuf,
               const Palette *palette,
               const pixformat_t oformat);
```

pixbuf	El búfer original.
palette	Paleta de colores necesaria para determinadas conversiones.
oformat	Formato del búfer resultado.

Retorna:

El búfer convertido.

Observaciones:

Ver *"Copia y conversiónCopia y conversión"* (Página 289).

pixbuf_destroy

Destruye el búfer.

```
void
pixbuf_destroy(Pixbuf **pixbuf);
```

pixbuf	El búfer. Será puesto a `NULL` tras la destrucción.

pixbuf_format

Obtiene el formato de píxel.

```
pixformat_t
pixbuf_format(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

El formato.

Observaciones:

Ver *"Formatos de píxelFormatos de píxel"* (Página 287).

pixbuf_width

Obtiene la anchura del búfer.

```
uint32_t
pixbuf_width(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Anchura.

pixbuf_height

Obtiene la altura del búfer.

```
uint32_t
pixbuf_height(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Altura.

pixbuf_size

Obtiene el tamaño del búfer (en píxeles).

```
uint32_t
pixbuf_size(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Anchura x altura.

pixbuf_dsize

Obtiene el tamaño del búfer (en bytes).

```
uint32_t
pixbuf_dsize(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Número de bytes totales del búfer.

pixbuf_cdata

Obtiene un puntero de solo lectura al contenido del búfer.

```
const byte_t*
pixbuf_cdata(const Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Puntero al primer elemento.

Observaciones:

Manipular correctamente el búfer requiere conocer los *"Formatos de píxelFormatos de píxel"* (Página 287) y, en ocasiones, utilizar los operadores a nivel de bit. Utilizar `pixbuf_get` para leer correctamente un píxel.

pixbuf_data

Obtiene un puntero de lectura/escritura al contenido del búfer.

```
byte_t*
pixbuf_data(Pixbuf *pixbuf);
```

pixbuf El búfer.

Retorna:

Puntero al primer elemento.

Observaciones:

Manipular correctamente el búfer requiere conocer los *"Formatos de píxelFormatos de píxel"* (Página 287) y, en ocasiones, utilizar los operadores a nivel de bit. Utilizar `pixbuf_get` y `pixbuf_set` para leer/escribir correctamente un píxel.

pixbuf_format_bpp

Obtiene los bits por píxel según el formato.

```
uint32_t
pixbuf_format_bpp(const pixformat_t format);
```

format El formato.

Retorna:

Bits por píxel.

Observaciones:

Ver *"Formatos de píxelFormatos de píxel"* (Página 287).

pixbuf_get

Obtiene el valor de un píxel.

```
uint32_t
pixbuf_get(const Pixbuf *pixbuf,
           const uint32_t x,
           const uint32_t y);
```

pixbuf El búfer.

x Coordenada x del píxel.

y Coordenada y del píxel.

Retorna:

El valor.

Observaciones:

Ver *"Formatos de píxelFormatos de píxel"* (Página 287) para interpretar correctamente el valor.

pixbuf_set

Establece el valor de un píxel.

```
void
pixbuf_set(Pixbuf *pixbuf,
           const uint32_t x,
           const uint32_t y,
           const uint32_t value);
```

pixbuf El búfer.

x Coordenada x del píxel.

y Coordenada y del píxel.

value El valor.

Observaciones:

Ver *"Formatos de píxelFormatos de píxel"* (Página 287) para interpretar correctamente el valor.

image_from_pixels

Crea una imagen a partir de un array de píxeles.

```
Image*
image_from_pixels(const uint32_t width,
                  const uint32_t height,
                  const pixformat_t format,
                  const byte_t *data,
                  const color_t *palette,
                  const uint32_t palsize);
```

width El ancho de la imagen (en píxeles).

height El alto de la imagen (en píxeles).

format El formato de pixel.

data Búfer que contiene el valor de color de cada pixel. Dependerá de la resolución y del formato.

palette Paleta de color necesaria para representar imágenes indexadas. Si es `NULL` y el formato es indexado, se utilizará una *"Paleta predefinidaPaleta predefinida"* (Página 286).

palsize Número de colores en la paleta.

Retorna:

La imagen recién creada.

Observaciones:

Ver *"Acceso a píxelesAcceso a píxeles"* (Página 292).

image_from_pixbuf

Crea una imagen a partir de un píxel buffer.

```
Image*
image_from_pixbuf(const Pixbuf *pixbuf,
                  const Palette *palette);
```

pixbuf El buffer.

palette La paleta.

Retorna:

La imagen recién creada.

Observaciones:

Igual a `image_from_pixels` evitando indicar los parámetros por separado.

image_from_file

Crea una imagen a partir de un archivo en disco.

```
Image*
image_from_file(const char_t *pathname,
                ferror_t *error);
```

pathname La ruta del archivo. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

La imagen recién creada.

Observaciones:

Solo los formatos *jpg*, *png*, *bmp* y *gif* son aceptados.

image_from_data

Crea una imagen a partir de un búfer que contiene los datos codificados.

```
Image*
image_from_data(const byte_t *data,
                const uint32_t size);
```

data El búfer con los datos de la imagen.

size El tamaño del búfer en bytes.

Retorna:

La imagen recién creada.

Observaciones:

El búfer representa datos codificados en *jpg*, *png*, *bmp* o *gif*. Para crear la imagen directamente desde píxeles utiliza `image_from_pixels`.

image_from_resource

Obtiene una imagen de un paquete de recursos.

```
const Image*
image_from_resource(const ResPack *pack,
                    const ResId id);
```

pack El paquete de recursos.

id El identificador del recurso.

Retorna:

La imagen.

Observaciones:

La imagen no debe ser destruida con `image_destroy` ya que forma parte del propio paquete (es constante). Hacer una copia con `image_copy` en caso de que deba conservarse tras la destrucción de los recursos. Ver *"Recursos"* (Página 131).

image_copy

Crea una copia de la imagen.

```
Image*
image_copy(const Image *image);
```

image La imagen original.

Retorna:

La imagen copiada.

Observaciones:

Las imágenes son objetos inmutables. Copiar realmente significa incrementar un contador interno sin llegar a clonar el objeto. No obstante, la aplicación debe destruir la copia con `image_destroy` al igual que las creadas con cualquier otro constructor. Cuando todas las copias sean destruídas se eliminará realmente de memoria.

image_trim

Crea una imagen recortando otra imagen.

```
Image*
image_trim(const uint32_t x,
           const uint32_t y,
           const uint32_t width,
           const uint32_t height);
```

x Coordenada X del origen de la sub-imagen.

y Coordenada Y del origen de la sub-imagen.

width Ancho en pixels de la sub-imagen.

height Alto en pixels de la sub-imagen.

Retorna:

La nueva imagen.

image_rotate

Crea una nueva imagen rotando otra ya existente.

```
Image*
image_rotate(const Image *image,
             const real32_t angle,
             const bool_t nsize,
             const color_t background,
             T2Df *t2d);
```

image	La imagen original.
angle	Ángulo en radianes.
nsize	`TRUE` la imagen resultado se redimensionará para que quepa toda la original. `FALSE` la imagen resultado tendrá las mismas dimensiones que la original, recortando parte del contenido (clipping).
background	Color de fondo. La nueva imagen tendrá zonas "en blanco" debido a la rotación.
t2d	Guarda la transformación aplicada a la imagen. Pueden ser `NULL` si no necesitamos este valor.

Retorna:

La imagen recién creada.

image_scale

Crea una copia de la imagen, con un nuevo tamaño.

```
Image*
image_scale(const Image *image,
            const uint32_t nwidth,
            const uint32_t nheight);
```

image	La imagen original.
nwidth	El ancho de la nueva imagen. Pasar `UINT32_MAX` para que se mantenga la relación de aspecto con respecto a `nheight`.
nheight	El alto de la nueva imagen. Pasar `UINT32_MAX` para que se mantenga la relación de aspecto con respecto a `nwidth`.

Retorna:

La imagen recién creada.

Observaciones:

Si ambos valores `nwidth`, `nheight` son `UINT32_MAX` o las nuevas dimensiones son idénticas a las actuales se incrementará el contador de referencias interno, al igual que ocurre en `image_copy`.

image_read

Crea una imagen a partir de los datos leídos desde un *"Streams"* (Página 197).

```
Image*
image_read(Stream *stm);
```

stm Stream de lectura. Se esperan datos codificados en *jpg*, *png*, *bmp* o *gif*. La función detecta el formato automáticamente.

Retorna:

La imagen recién creada.

image_to_file

Guarda una imagen en disco, utilizado el codec asociado a la misma.

```
bool_t
image_to_file(const Image *image,
              const char_t *pathname,
              ferror_t *error);
```

image La imagen.

pathname La ruta del archivo destino. *"Filename y pathnameFilename y pathname"* (Página 181).

error Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si se ha guardado correctamente o `FALSE` y ha ocurrido algún error.

Observaciones:

Utiliza `image_codec` para cambiar el codec por defecto.

image_write

Escribe una imagen en un stream de salida, utilizado el codec asociado a la misma.

```
void
image_write(Stream *stm,
            const Image *image);
```

stm Stream de escritura. Se escribirán datos codificados en *jpg*, *png*, *bmp* o *gif*.

image La imagen.

Observaciones:

Utiliza `image_codec` para cambiar el codec por defecto.

image_destroy

Destruye la imagen.

```
void
image_destroy(Image **image);
```

image La imagen. Será puesto a `NULL` tras la destrucción.

image_format

Obtiene el formato de pixel de la imagen.

```
pixformat_t
image_format(const Image *image);
```

image La imagen.

Retorna:

El formato de pixel.

image_width

Obtiene el ancho de la imagen en pixels.

```
uint32_t
image_width(const Image *image);
```

image La imagen.

Retorna:

Número de pixeles de anchura.

image_height

Obtiene el alto de la imagen en pixels.

```
uint32_t
image_height(const Image *image);
```

image La imagen.

Retorna:

Número de pixeles de altura.

image_pixels

Obtiene un búfer con los píxeles que forman la imagen decodificada.

```
Pixbuf*
image_pixels(const Image *image,
             const pixformat_t format);
```

image	La imagen.
format	El formato de pixel requerido.

Retorna:

Píxel búfer con el contenido de la imagen.

Observaciones:

Si en `pixformat` indicamos `ekFIMAGE` devolverá el búfer con el formato original de la imagen. Podemos indicar `ekRGB24`, `ekRGBA32` o `ekGRAY8` si necesitamos un formato específico. No es posible utilizar formatos indexados.

image_codec

Cambia el codec por defecto asociado con la imagen.

```
bool
image_codec(const Image *image,
            const codec_t codec);
```

```
Image *img = image_from_file("lenna.jpg", NULL);
Stream *stm = stm_socket(ip, port, NULL, NULL);
image_codec(img, ekPNG);
image_write(socket, img);
```

image	La imagen.
codec	El nuevo codec.

Retorna:

`TRUE` si el API gráfico soporta el codec seleccionado. `FALSE` en caso contrario.

Observaciones:

El cambio tendrá efecto la próxima vez que guardemos o escribamos la imagen. Por defecto, la imagen conserva el codec con el que fue leída. Cuando la creamos con `image_from_pixels` se le asigna el codec `ekJPG` por defecto. Para imágenes desde contextos 2d `dctx_image`, el codec por defecto es `ekPNG`. Todos los codecs son soportados por todas las APIs gráficas, salvo `ekGIF` en algunas versiones de Linux. Chequea en valor de retorno si es imperativo que tu aplicación exporte imágenes en GIF.

image_get_codec

Obtiene el codec asociado con la imagen.

```
codec_t
image_get_codec(const Image *image);
```

image La imagen.

Retorna:

El codec.

Observaciones:

Ver `image_codec`.

image_num_frames

Obtiene el número de secuencias en imágenes animadas.

```
uint32_t
image_num_frames(const Image *image);
```

image La imagen.

Retorna:

El número de secuencias o *frames*.

Observaciones:

Únicamente el formato *gif* soporta animaciones. Para el resto siempre se devolverá 1.

image_frame_length

Obtiene el tiempo que dura una secuencia de animación.

```
real32_t
image_frame_length(const Image *image,
                   const uint32_t findex);
```

image La imagen.

findex El índice del frame.

Retorna:

El tiempo de la secuencia en segundos.

Observaciones:

Únicamente el formato *gif* soporta animaciones.

image_data

Vincula datos de usuario con la imagen.

```
void
image_data(Image *image,
           type *data,
           FPtr_destroy func_destroy_data,
           type );
```

image La imagen.

data Los datos de usuario.

func_destroy_data Destructor de los datos de usuario.

Tipo de datos de usuario.

image_get_data

Obtiene los datos de usuario de la imagen.

```
type*
image_get_data(const Image *image,
               type );
```

image La imagen.

Tipo de datos de usuario.

Retorna:

Los datos de usuario.

image_native

Obtiene la imagen en el formato nativo de cada plataforma.

```
void*
image_native(const Image *image);
```

image La imagen.

Retorna:

La imagen nativa. `Gdiplus::Bitmap` en Windows, `GdkPixbuf` en Linux y `NSImage` en macOS.

font_create

Crea una fuente tipográfica.

```
Font*
font_create(const char_t *family,
            const real32_t size,
            const uint32_t style);
```

family Familia tipográfica. Pe: "Arial", "Times New Roman", etc.

size Tamaño de la fuente. Por defecto en píxeles. Utiliza `ekFPOINTS` en `style` para cambiar la unidad.

style Operación `OR |` sobre los campos de la estructura `fstyle_t`. Pe: `ekFBOLD | ekFITALIC`.

Retorna:

La fuente recién creada.

font_system

Crea una fuente tipográfica, con la familia por defecto del sistema.

```
Font*
font_system(const real32_t size,
            const uint32_t style);
```

size Tamaño de la fuente. Por defecto en píxeles. Utiliza `ekFPOINTS` en `style` para cambiar la unidad.

style Operación `OR |` sobre los campos de la estructura `fstyle_t`. Pe: `ekFBOLD | ekFITALIC`.

Retorna:

La fuente recién creada.

font_monospace

Crea una fuente tipográfica, con la familia mono-espacio por defecto del sistema.

```
Font*
font_monospace(const real32_t size,
               const uint32_t style);
```

size Tamaño de la fuente. Por defecto en píxeles. Utiliza `ekFPOINTS` en `style` para cambiar la unidad.

style Operación `OR` `|` sobre los campos de la estructura `fstyle_t`. Pe: `ekFBOLD | ekFITALIC`.

Retorna:

La fuente recién creada.

font_with_style

Crea una copia de una fuente existente, cambiando el estilo.

```
Font*
font_with_style(const Font *font,
                const uint32_t style);
```

font Fuente original.

style Operación `OR` `|` sobre los campos de la estructura `fstyle_t`. Pe: `ekFBOLD | ekFITALIC`.

Retorna:

Una copia de `font` con otro estilo.

font_copy

Crea una copia exacta de una fuente tipográfica.

```
Font*
font_copy(const Font *font);
```

font Fuente original.

Retorna:

La copia de `font`.

Observaciones:

Las fuentes son objetos inmutables. Copiar realmente significa incrementar un contador interno sin llegar a clonar el objeto. No obstante, la aplicación debe destruir la copia con `font_destroy` al igual que las creadas con cualquier otro constructor.

font_destroy

Destruye la fuente.

```
void
font_destroy(Font **font);
```

font La fuente. Será puesto a `NULL` tras la destrucción.

font_equals

Compara dos fuentes. Se consideran iguales si tienen la misma familia, tamaño y estilo.

```
bool_t
font_equals(const Font *font1,
            const Font *font2);
```

font1 Primera fuente a comparar.

font2 Segunda fuente a comparar.

Retorna:

`TRUE` si son iguales, `FALSE` si no.

font_regular_size

Obtiene el tamaño de la fuente por defecto para los controles de interfaz.

```
real32_t
font_regular_size(void);
```

Retorna:

El tamaño por defecto en píxeles.

font_small_size

Obtiene el tamaño *pequeño* de la fuente por defecto para los controles de interfaz.

```
real32_t
font_small_size(void);
```

Retorna:

El tamaño en píxeles.

Observaciones:

Este tamaño es ligeramente menor que el obtenido por `font_regular_size`.

font_mini_size

Obtiene el tamaño *mini* de la fuente por defecto para los controles de interfaz.

```
real32_t
font_mini_size(void);
```

Retorna:

El tamaño en píxeles.

Observaciones:

Este tamaño es ligeramente menor que el obtenido por `font_small_size`.

font_family

Obtiene la familia tipográfica de la fuente.

```
const char_t*
font_family(const Font *font);
```

font La fuente.

Retorna:

La familia tipográfica en UTF8.

font_size

Obtiene el tamaño de la fuente.

```
real32_t
font_size(const Font *font);
```

font La fuente.

Retorna:

El tamaño. Las unidades depende el parámetro `style`.

font_height

Obtiene la altura de la celda o línea de texto con esta fuente.

```
real32_t
font_height(const Font *font);
```

font La fuente.

Retorna:

Altura de la celda.

font_style

Obtiene el estilo de la fuente.

```
uint32_t
font_style(const Font *font);
```

font La fuente.

Retorna:

El estilo. Combinación de valores de la estructura `fstyle_t`. Pe: `ekFBOLD | ekFITALIC` .

font_extents

Obtiene el tamaño en píxeles de una cadena de texto, en función de la fuente.

```
void
font_extents(const Font *font,
             const char_t *text,
             const real32_t refwidth,
             real32_t *width,
             real32_t *height);
```

font La fuente.

text La cadena de texto a dimensionar.

refwidth Anchura máxima del cuadro de texto.

width Ancho del cuadro de texto.

height Alto del cuadro de texto.

font_exists_family

Comprueba si una familia tipográfica está instalada en el sistema operativo.

```
bool_t
font_exists_family(const char_t *family);
```

family Cadena UTF8 con el nombre de la familia, terminada en carácter nulo `'\0'`.

Retorna:

`TRUE` si existe la familia, `FALSE` si no.

font_installed_families

Obtiene una lista con los nombres de todas las familias tipográficas instaladas en el sistema operativo.

```
ArrPt(String)*
font_installed_families(void);
```

```
ArrPt(String) *families = font_installed_families();
...
arrpt_destroy(&families, str_destroy, String);
```

Retorna:

Array de `String` con los nombres de las familias, ordenadas alfabéticamente. Debe ser destruido con `arrpt_destroy`.

font_native

Obtiene la fuente en el formato nativo de cada plataforma.

```
void*
font_native(const Font *font);
```

font La fuente.

Retorna:

La fuente nativa. `HFONT` en Windows, `PangoFontDescription` en Linux y `NSFont` en macOS.

Capítulo 40

Librería Gui

40.1. Tipos y Constantes

enum gui_orient_t

Orientación.

ekGUI_HORIZONTAL	Horizontal.
ekGUI_VERTICAL	Vertical.

enum gui_state_t

Valores de estado.

ekGUI_OFF	Apagado/desactivado.
ekGUI_ON	Encendido/activado.
ekGUI_MIXED	Medio/indeterminado.

enum gui_mouse_t

Botones del ratón.

ekGUI_MOUSE_LEFT	Izquierdo.
ekGUI_MOUSE_RIGHT	Derecho.
ekGUI_MOUSE_MIDDLE	Centro.

enum gui_cursor_t

Cursores. Ver `window_cursor`.

`ekGUI_CURSOR_ARROW`	Flecha (por defecto).
`ekGUI_CURSOR_HAND`	Mano.
`ekGUI_CURSOR_IBEAM`	Barra vertical (edición de texto).
`ekGUI_CURSOR_CROSS`	Cruz.
`ekGUI_CURSOR_SIZEWE`	Redimensionado horizontal (izquierda-derecha).
`ekGUI_CURSOR_SIZENS`	Redimensionado vertical (arriba-abajo).
`ekGUI_CURSOR_USER`	Creado a partir de una imagen.

enum gui_close_t

Motivo del cierre de una ventana.

`ekGUI_CLOSE_ESC`	Se ha pulsado la tecla `[ESC]` (cancelar).
`ekGUI_CLOSE_INTRO`	Se ha pulsado la tecla `[ENTER]` (aceptar).
`ekGUI_CLOSE_BUTTON`	Se ha pulsado el botón de cerrar `[X]` en la barra de título.
`ekGUI_CLOSE_DEACT`	Se ha ocultado la ventana padre.

enum gui_scale_t

Formas de escalado.

`ekGUI_SCALE_AUTO`	Escalado automático, la proporción puede cambiar.
`ekGUI_SCALE_NONE`	Sin escalado.
`ekGUI_SCALE_ASPECT`	Escalado automático, pero manteniendo la proporción *(aspect ratio)*.
`ekGUI_SCALE_ASPECTDW`	Igual que el anterior, pero no aumenta el tamaño original, solo lo reduce si procede.

enum gui_event_t

Tipo de evento. Ver *"Eventos GUIEventos GUI"* (Página 306).

`ekGUI_EVENT_LABEL`	Se ha hecho clic sobre un control `Label`.
`ekGUI_EVENT_BUTTON`	Se ha hecho clic sobre un control `Button`.
`ekGUI_EVENT_POPUP`	Se ha cambiado la selección de un control `PopUp`.

ekGUI_EVENT_LISTBOX	Se ha cambiado la selección de un control `ListBox`.
ekGUI_EVENT_SLIDER	Se está deslizando un control `Slider`.
ekGUI_EVENT_UPDOWN	Se ha hecho clic sobre un control `UpDown`.
ekGUI_EVENT_TXTFILTER	Se está editando el texto de un control `Edit` o `Combo`.
ekGUI_EVENT_TXTCHANGE	Se ha terminado de editar el texto de un control `Edit` o `Combo`.
ekGUI_EVENT_FOCUS	Un control ha recibido el foco del teclado.
ekGUI_EVENT_MENU	Se ha hecho clic sobre un menú.
ekGUI_EVENT_DRAW	Hay que volver a dibujar el contenido de la vista.
ekGUI_EVENT_RESIZE	Ha cambiado el tamaño de una vista.
ekGUI_EVENT_ENTER	El ratón ha entrado en el área de la vista.
ekGUI_EVENT_EXIT	El ratón ha salido del área de la vista.
ekGUI_EVENT_MOVED	El ratón se está moviendo sobre la superficie de la vista.
ekGUI_EVENT_DOWN	Se ha pulsado un botón del ratón.
ekGUI_EVENT_UP	Se ha liberado un botón del ratón.
ekGUI_EVENT_CLICK	Se ha hecho clic sobre una vista.
ekGUI_EVENT_DRAG	Se está realizando *dragging* dentro de la vista.
ekGUI_EVENT_WHEEL	Se ha movido la rueda del ratón.
ekGUI_EVENT_KEYDOWN	Se ha pulsado una tecla.
ekGUI_EVENT_KEYUP	Se ha liberado una tecla.
ekGUI_EVENT_WND_MOVED	Se está moviendo la ventana por el escritorio.
ekGUI_EVENT_WND_SIZING	Se está re-dimensionando la ventana.
ekGUI_EVENT_WND_SIZE	Se ha re-dimensionando la ventana.
ekGUI_EVENT_WND_CLOSE	Se ha cerrado la ventana.
ekGUI_EVENT_COLOR	Se está actualizando un color de `comwin_color`.
ekGUI_EVENT_THEME	Ha cambiado el tema del escritorio.
ekGUI_EVENT_OBJCHANGE	Se ha editado un objeto vinculado con un layout. *"Notificaciones y campos calculadosNotificaciones y campos calculados"* (Página 364).

ekGUI_EVENT_TBL_NROWS	Una tabla necesita saber el número de filas. tableview_OnData.
ekGUI_EVENT_TBL_BEGIN	Una tabla va a comenzar a dibujar la parte visible de los datos. tableview_OnData.tableview_OnData.
ekGUI_EVENT_TBL_END	Una tabla ha terminado de dibujar. tableview_OnData.
ekGUI_EVENT_TBL_CELL	Una tabla necesita los datos de una celda. tableview_OnData.
ekGUI_EVENT_TBL_SEL	Han cambiado las filas seleccionadas en una tabla. tableview_OnSelect.
ekGUI_EVENT_TBL_HEADCLICK	Se ha hecho clic en una cabecera de la tabla. tableview_OnHeaderClick.

enum window_flag_t

Atributos de creación de una ventana.

ekWINDOW_FLAG	Atributos por defecto.
ekWINDOW_EDGE	La ventana dibuja un borde exterior.
ekWINDOW_TITLE	La ventana tiene barra de título.
ekWINDOW_MAX	La ventana muestra el botón de maximizar.
ekWINDOW_MIN	La ventana muestra el botón de minimizar.
ekWINDOW_CLOSE	La ventana muestra el botón de cerrar.
ekWINDOW_RESIZE	La ventana tiene bordes redimensionables.
ekWINDOW_RETURN	La ventana procesará la pulsación de la tecla [RETURN] como un posible evento de cierre, enviando el mensaje OnClose.
ekWINDOW_ESC	La ventana procesará la pulsación de la tecla [ESC] como un posible evento de cierre, enviando el mensaje OnClose.
ekWINDOW_STD	Combinación ekWINDOW_TITLE | ekWINDOW_MIN | ekWINDOW_CLOSE.
ekWINDOW_STDRES	Combinación ekWINDOW_STD | ekWINDOW_MAX | ekWINDOW_RESIZE.

enum gui_notif_t

Notificaciones enviadas por la librería gui.

ekGUI_NOTIF_LANGUAGE	Se ha cambiado el idioma de la interfaz.
ekGUI_NOTIF_WIN_DESTROY	Se ha destruido una ventana.
ekGUI_NOTIF_MENU_DESTROY	Se ha destruido un menú.

struct Control

Control de interfaz (abstracto).

```
struct Control;
```

struct Label

Control de interfaz que contiene un texto estático, normalmente limitado a una sola línea. *"Label"* (Página 309).

```
struct Label;
```

struct Button

Control de interfaz que representa un botón. *"Button"* (Página 310).

```
struct Button;
```

struct PopUp

Control botón con lista desplegable. *"PopUp"* (Página 313).

```
struct PopUp;
```

struct Edit

Control de edición de texto *"Edit"* (Página 313).

```
struct Edit;
```

struct Combo

Control que combina un cuadro de edición con una lista desplegable. *"Combo"* (Página 315).

```
struct Combo;
```

struct ListBox

Control de lista. *"ListBox"* (Página 316).

```
struct ListBox;
```

struct UpDown

Control que muestra dos pequeños botones de incremento y decremento. *"UpDown"* (Página 317).

```
struct UpDown;
```

struct Slider

Control que muestra una barra con un deslizador. *"Slider"* (Página 317).

```
struct Slider;
```

struct Progress

Barra de progreso. *"Progress"* (Página 318).

```
struct Progress;
```

struct View

Vista personalizada. Permite crear nuestro propios controles, dibujando lo que queramos. *"View"* (Página 319)

```
struct View;
```

struct TextView

Vista de texto con varios párrafos y diferentes atributos. *"TextView"* (Página 323).

```
struct TextView;
```

struct ImageView

Visor de imágenes. *"ImageView"* (Página 325).

```
struct ImageView;
```

struct TableView

Vista de tabla con varias filas y columnas. *"TableView"* (Página 326).

```
struct TableView;
```

struct SplitView

Vista partida redimensionable en horizontal o vertical. *"SplitView"* (Página 332).

```
struct SplitView;
```

struct Layout

Rejilla invisible donde se organizan los controles de un `Panel`. *"Layout"* (Página 335).

```
struct Layout;
```

struct Cell

Cada una de las celdas que forman un `Layout`. *"Cell"* (Página 344).

```
struct Cell;
```

struct Panel

Área interna de una ventana, que permite agrupar diferentes controles. *"Panel"* (Página 344).

```
struct Panel;
```

struct Window

Ventana de interfaz. *"Window"* (Página 350).

```
struct Window;
```

struct Menu

Menu o submenu. *"Menu"* (Página 366).

```
struct Menu;
```

struct MenuItem

Elemento dentro de un menú. *"MenuItem"* (Página 367).

```
struct MenuItem;
```

struct EvButton

Parámetros del evento *OnClick* de un botón o *OnSelect* de un popup.

```
struct EvButton
{
    uint32_t index;
    gui_state_t state;
    const char_t* text;
};
```

`index`	Índice del botón o del elemento.
`state`	Estado.
`text`	Texto.

struct EvSlider

Parámetros del evento *OnMoved* de un slider.

```
struct EvSlider
{
    real32_t pos;
    real32_t incr;
    uint32_t step;
};
```

`pos`	Posición normalizada del slider (0, 1).
`incr`	Incremento con respecto a la posición anterior.
`step`	Índice del intervalo (solo para rangos discretos).

struct EvText

Parámetros del evento *OnChange* de los cuadros de texto.

```
struct EvText
{
    const char_t* text;
    uint32_t cpos;
};
```

`text` Texto.

`cpos` Posición del cursor *(caret)*.

struct EvTextFilter

Resultado del evento *OnFilter* de los cuadros de texto.

```
struct EvTextFilter
{
    bool_t apply;
    char_t* text;
    uint32_t cpos;
};
```

`apply` `TRUE` si se debe cambiar el texto original del control.

`text` Nuevo texto del control, que es una revisión (filtro) del texto original.

`cpos` Posición del cursor *(caret)*.

struct EvDraw

Parámetros del evento *OnDraw*.

```
struct EvDraw
{
    DCtx* ctx;
    real32_t x;
    real32_t y;
    real32_t width;
    real32_t height;
};
```

`ctx` Contexto de dibujo 2D.

`x` Coordenada x del area de dibujo (viewport).

`y` Coordenada y del area de dibujo.

`width` Ancho del área de dibujo.

`height` Alto del área de dibujo.

struct EvMouse

Parámetros de eventos de ratón.

```
struct EvMouse
```

```
{
    real32_t x;
    real32_t y;
    gui_mouse_t button;
    uint32_t count;
};
```

`x`	Coordenada x del puntero.
`y`	Coordenada y del puntero.
`button`	Botón activo.
`count`	Número de clics.

struct EvWheel

Parámetros del evento *OnWheel*.

```
struct EvWheel
{
    real32_t x;
    real32_t y;
    real32_t dx;
    real32_t dy;
    real32_t dz;
};
```

`x`	Coordenada x del puntero.
`y`	Coordenada y del puntero.
`dx`	Incremento en x de la rueda o *trackpad*.
`dy`	Incremento en y de la rueda o *trackpad*.
`dz`	Incremento en z de la rueda o *trackpad*.

struct EvKey

Parámetros de eventos de teclado.

```
struct EvKey
{
    vkey_t key;
};
```

`key`	Tecla pulsada o liberada.

struct EvPos

Parámetros de eventos de cambio de posición.

```
struct EvPos
{
    real32_t x;
    real32_t y;
};
```

x	Coordenada x.
y	Coordenada y.

struct EvSize

Parámetros de eventos de cambio de tamaño.

```
struct EvSize
{
    real32_t width;
    real32_t height;
};
```

width	Anchura (tamaño en x).
height	Altura (tamaño en y).

struct EvWinClose

Parámetros de evento de cierre de ventana.

```
struct EvWinClose
{
    gui_close_t origin;
};
```

origin	Origen del cierre.

struct EvMenu

Parámetros de evento de menú.

```
struct EvMenu
{
    uint32_t index;
```

```
    gui_state_t state;
    const char_t* str;
};
```

`index` Índice del *item* pulsado.

`state` Estado del *item* pulsado.

`str` Texto del *item* pulsado.

struct EvTbPos

Localización de una celda en una tabla.

```
struct EvTbPos
{
    uint32_t col;
    uint32_t row;
};
```

`col` Índice de la columna.

`row` Índice de la fila.

struct EvTbRect

Grupo de celdas en una tabla.

```
struct EvTbRect
{
    uint32_t stcol;
    uint32_t edcol;
    uint32_t strow;
    uint32_t edrow;
};
```

`stcol` Índice de la columna inicial.

`edcol` Índice de la columna final.

`strow` Índice de la fila inicial.

`edrow` Índice de la fila final.

struct EvTbSel

Selección en una tabla.

```
struct EvTbSel
{
```

```
    ArrSt(uint32_t)* sel;
};
```

`sel` Índices de fila.

struct EvTbCell

Datos de una celda en una tabla.

```
struct EvTbCell
{
    const char_t* text;
    align_t align;
};
```

`text` Texto de la celda.

`align` Alineación del texto.

40.2. Funciones

gui_start

Inicia la librería *Gui*, reservando espacio para las estructuras internas globales. Internamente llama a `draw2d_start`. Es llamada automáticamente por `osmain`.

```
void
gui_start(void);
```

gui_finish

Finaliza la librería *Gui*, liberando el espacio de las estructuras internas globales. Internamente llama a `draw2d_finish`. Es llamada automáticamente por `osmain`.

```
void
gui_finish(void);
```

gui_respack

Registra un paquete de recursos.

```
void
gui_respack(FPtr_respack func_respack);
```

func_respack Constructor de recursos.

Observaciones:

Ver *"Recursos"* (Página 131).

gui_language

Establece el idioma de los recursos registrados con `gui_respack`.

```
void
gui_language(const char_t *lang);
```

lang El idioma.

Observaciones:

Ver *"Recursos"* (Página 131).

gui_text

Obtiene una cadena de texto a través de su identificador de recurso.

```
const char_t*
gui_text(const ResId id);
```

id Identificador del recurso.

Retorna:

La cadena de texto o `NULL` si no se encuentra.

Observaciones:

El recurso debe pertenecer a un paquete registrado con `gui_respack`.

gui_image

Obtiene una imagen a través de su identificador de recurso.

```
const Image*
gui_image(const ResId id);
```

id Identificador del recurso.

Retorna:

La imagen o `NULL` si no se encuentra.

Observaciones:

El recurso debe pertenecer a un paquete registrado con `gui_respack`. No hay que destruir la imagen ya que está gestionada por Gui.

gui_file

Obtiene el contenido de un archivo a través de su identificador de recurso.

```
const byte_t*
gui_file(const ResId id,
         uint32_t *size);
```

id	Identificador del recurso.
size	Tamaño en bytes del búfer.

Retorna:

Los datos del archivo o `NULL` si no se encuentra.

Observaciones:

El recurso debe pertenecer a un paquete registrado con `gui_respack`. Los datos están gestionados por Gui, por lo que no hay que liberar memoria.

gui_dark_mode

Determina si el entorno de ventanas tiene un tema claro u oscuro.

```
bool_t
gui_dark_mode(void);
```

Retorna:

`TRUE` para *Dark mode*, `FALSE` para *light mode*.

gui_alt_color

Crea un color con dos versiones alternativas.

```
color_t
gui_alt_color(const color_t light_color,
              const color_t dark_color);
```

light_color	Color para temas CLAROS de escritorio.
dark_color	Color para temas OSCUROS de escritorio.

Retorna:

El color.

Observaciones:

El sistema establecerá el color final en función de la "claridad" de los colores del gestor de ventanas (Light/Dark). NO SE PERMITEN colores alternativos anidados. Los valores `light` y `dark` deben ser colores RGB o de sistema.

gui_label_color

Retorna el color por defecto de las etiquetas de texto `Label`.

```
color_t
gui_label_color(void);
```

Retorna:

El color.

gui_view_color

Retorna el color de fondo en controles `View`.

```
color_t
gui_view_color(void);
```

Retorna:

El color.

gui_line_color

Retorna el color de líneas en tablas o elementos separadores de ventanas.

```
color_t
gui_line_color(void);
```

Retorna:

El color.

gui_link_color

Retorna el color del texto en hiperenlaces.

```
color_t
gui_link_color(void);
```

Retorna:

El color.

gui_border_color

Retorna el color del borde en controles tipo botón, popups, etc.

```
color_t
gui_border_color(void);
```

Retorna:

El color.

gui_resolution

Retorna la resolución de pantalla.

```
S2Df
gui_resolution(void);
```

Retorna:

Resolución.

gui_mouse_pos

Retorna la posición del cursor del ratón.

```
V2Df
gui_mouse_pos(void);
```

Retorna:

Posición.

gui_update

Actualiza todas las ventanas de la aplicación, después de un cambio de tema.

```
void
gui_update(void);
```

Observaciones:

Normalmente no es necesario llamar a este método. Se invoca automáticamente desde `osapp`.

gui_OnThemeChanged

Establece un manejador para detectar el cambio del tema visual del entorno de ventanas.

```
void
gui_OnThemeChanged(Listener *listener);
```

listener El manejador del evento.

gui_update_transitions

Actualiza las animaciones automáticas de la interfaz.

```
void
gui_update_transitions(const real64_t prtime,
                       const real64_t crtime);
```

prtime Tiempo del instante previo.

crtime Tiempo del instante actual.

Observaciones:

Normalmente no es necesario llamar a este método. Se invoca automáticamente desde `osapp`.

gui_OnNotification

Establece un manejador para recibir notificaciones por parte de `gui`.

```
void
gui_OnNotification(Listener *listener);
```

listener El manejador del evento.

Observaciones:

Ver `gui_notif_t`.

evbind_object

Obtiene el objeto vinculado a un layout dentro de una función `callback`.

```
type*
evbind_object(Event *e,
              type);
```

e	El evento.
type	El tipo de objeto.

Retorna:

El objeto.

Observaciones:

Ver *"Notificaciones y campos calculadosNotificaciones y campos calculados"* (Página 364).

evbind_modify

Comprueba, dentro de una función `callback`, si el campo del objeto se modificado.

```
bool_t
evbind_modify(Event *e,
              type,
              mtype,
              mname);
```

e	El evento.
type	El tipo de objeto.
mtype	El tipo del campo a comprobar.
mname	El nombre del campo a comprobar.

Retorna:

`TRUE` si el campo ha sido modificado.

Observaciones:

Ver *"Notificaciones y campos calculadosNotificaciones y campos calculados"* (Página 364).

label_create

Crea un control de texto.

```
Label*
label_create(void);
```

Retorna:

El nuevo Label.

label_multiline

Crea un control de texto multilínea.

```
Label*
label_multiline(void);
```

Retorna:

El nuevo Label.

label_OnClick

Establece el manejador del evento `OnClick`.

```
void
label_OnClick(Label *label,
              Listener *listener);
```

label	El Label.
listener	Manejador del evento.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

label_text

Establece el texto que mostrará la etiqueta.

```
void
label_text(Label *label,
           const char_t *text);
```

label	El Label.
text	Cadena C UTF8 terminada en carácter nulo `'\0'`.

label_font

Establece la fuente del texto.

```
void
label_font(Label *label,
           const Font *font);
```

label El Label.

font Fuente tipográfica.

label_style_over

Establece los modificadores de fuente, cuando el ratón está sobre el control.

```
void
label_style_over(Label *label,
                 const uint32_t style);
```

label El Label.

style Combinación de valores `fstyle_t`.

label_align

Establece la alineación horizontal del texto con respecto al tamaño del control.

```
void
label_align(Label *label,
            const align_t align);
```

label El Label.

align La alineación.

label_color

Establece el color del texto.

```
void
label_color(Label *label,
            const color_t color);
```

label El Label.

color El color.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

label_color_over

Establece el color del texto, cuando el ratón está sobre el control.

```
void
label_color_over(Label *label,
                 const color_t color);
```

label El Label.

color El color.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

label_bgcolor

Establece el color de fondo del texto.

```
void
label_bgcolor(Label *label,
              const color_t color);
```

label El Label.

color El color.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

label_bgcolor_over

Establece el color de fondo del texto, cuando el ratón está sobre el control.

```
void
label_bgcolor_over(Label *label,
                   const color_t color);
```

label El Label.

color El color.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

button_push

Crea un botón de pulsación, el típico `[Aceptar]`, `[Cancelar]`, etc.

```
Button*
button_push(void);
```

Retorna:

El botón.

button_check

Crea una casilla de verificación (checkbox).

```
Button*
button_check(void);
```

Retorna:

El botón.

button_check3

Crea una casilla de verificación (checkbox) con tres estados.

```
Button*
button_check3(void);
```

Retorna:

El botón.

button_radio

Crea un botón de radio.(radiobutton).

```
Button*
button_radio(void);
```

Retorna:

El botón.

button_flat

Crea un botón plano, al que puede asignarse una imagen. Es el típico botón de las barras de herramientas.

```
Button*
button_flat(void);
```

Retorna:

El botón.

button_flatgle

Crea un botón plano con estado. El botón alternará entre pulsado/liberado cada vez que se haga clic sobre él.

```
Button*
button_flatgle(void);
```

Retorna:

El botón.

button_OnClick

Establece un manejador para la pulsación del botón.

```
void
button_OnClick(Button *button,
               Listener *listener);
```

button	El botón.
listener	Función *callback* que se llamará tras hacer clic.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

button_text

Establece el texto que mostrará el botón.

```
void
button_text(Button *button,
            const char_t *text);
```

button El botón.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

Observaciones:

En botones planos, el texto se mostrará como *tooltip*.

button_text_alt

Establece un texto alternativo.

```
void
button_text_alt(Button *button,
                const char_t *text);
```

button El botón.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

Observaciones:

Solo aplicable en botones planos con estado `button_flatgle`. Se mostrará cuando el botón esté en estado `ekGUI_ON`.

button_tooltip

Establece un tooltip para el botón. Es un pequeño texto explicativo que aparecerá cuando el ratón esté sobre el control.

```
void
button_tooltip(Button *button,
               const char_t *text);
```

button El botón.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

button_font

Establece la fuente del botón.

```
void
button_font(Button *button,
            const Font *font);
```

button El botón.

font Fuente tipográfica.

button_image

Establece el icono que mostrará el botón.

```
void
button_image(Button *button,
             const Image *image);
```

button El botón.

image Imagen.

Observaciones:

No aplicable en checkbox ni radiobutton. En botones planos, el tamaño del control se ajustará a la imagen.

button_image_alt

Establece una imagen alternativa para el botón.

```
void
button_image_alt(Button *button,
                 const Image *image);
```

button El botón.

image Imagen.

Observaciones:

Solo aplicable en botones planos con estado `button_flatgle`. Se mostrará cuando el botón esté en estado `ekGUI_ON`.

button_state

Establece el estado del botón.

```
void
button_state(Button *button,
             const gui_state_t state);
```

button El botón.

state Estado.

Observaciones:

No aplicable en botones de pulsación `button_push`.

button_get_state

Obtiene el estado del botón.

```
gui_state_t
button_get_state(Button *button);
```

button El botón.

Retorna:

El estado.

Observaciones:

No aplicable en botones de pulsación `button_push`.

button_tag

Establece una etiqueta para el botón.

```
void
button_tag(Button *button,
           const uint32_t tag);
```

button El botón.

tag La etiqueta.

button_get_tag

Obtiene la etiqueta del botón.

```
uint32_t
button_get_tag(const Button *button);
```

button El botón.

Retorna:

La etiqueta.

popup_create

Crea un nuevo control de lista desplegable *(PopUp button)*.

```
PopUp*
popup_create(void);
```

Retorna:

El popup recién creado.

popup_OnSelect

Establece un manejador para la selección de un nuevo elemento.

```
void
popup_OnSelect(PopUp *popup,
               Listener *listener);
```

popup — El popup.

listener — Función *callback* que se llamará tras seleccionar un nuevo elemento de la lista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

popup_tooltip

Asigna un tooltip al control popup.

```
void
popup_tooltip(PopUp *popup,
              const char_t *text);
```

popup — El popup.

text — Cadena C UTF8 terminada en carácter nulo `'\0'`.

popup_add_elem

Añade un nuevo elemento a la lista desplegable.

```
void
popup_add_elem(PopUp *popup,
               const char_t *text,
               const Image *image);
```

popup — El popup.

text — El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image — Icono asociado al elemento o el identificador del recurso. Por cuestión de espacio, se escalará a un alto máximo de 16 píxeles.

popup_set_elem

Edita un elemento de la lista desplegable.

```
void
popup_set_elem(PopUp *popup,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

popup — El popup.

index — El índice del elemento a sustituir.

text — El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image — Icono asociado al elemento o el identificador del recurso. Por cuestión de espacio, se escalará a un alto máximo de 16 píxeles.

popup_clear

Elimina todos los elementos de la lista desplegable.

```
void
popup_clear(PopUp *popup);
```

popup — El popup.

popup_count

Obtiene el número de elementos de la lista.

```
uint32_t
popup_count(const PopUp *popup);
```

popup — El popup.

Retorna:

El número de elementos.

popup_list_height

Establece el tamaño de la lista desplegable.

```
void
popup_list_height(PopUp *popup,
                  const uint32_t elems);
```

popup El popup.

elems Número de elementos visibles. Si el control tiene más, aparecerá una barra de scroll.

popup_selected

Establece el elemento seleccionado del popup.

```
void
popup_selected(PopUp *popup,
               const uint32_t index);
```

popup El popup.

index El elemento a seleccionar. Si pasamos `UINT32_MAX` se quita la selección.

popup_get_selected

Obtiene el elemento seleccionado del popup.

```
uint32_t
popup_get_selected(PopUp *popup);
```

popup El popup.

Retorna:

El elemento seleccionado.

edit_create

Crea un control de edición de texto.

```
Edit*
edit_create(void);
```

Retorna:

El edit.

edit_multiline

Crea un control de edición de texto que permite múltiples líneas.

```
Edit*
edit_multiline(void);
```

Retorna:

El edit.

edit_OnFilter

Establece un manejador para filtrar el texto mientras se edita.

```
void
edit_OnFilter(Edit *edit,
              Listener *listener);
```

edit — El edit.

listener — Función *callback* que se llamará tras cada pulsación de tecla. En `EvTextFilter` de `event_result` se devolverá el texto filtrado.

Observaciones:

Ver *"Filtrar textosFiltrar textos"* (Página 313) y *"Eventos GUIEventos GUI"* (Página 306).

edit_OnChange

Establece un manejador para detectar cuando el texto ha cambiado.

```
void
edit_OnChange(Edit *edit,
              Listener *listener);
```

edit — El edit.

listener — Función *callback* que se llamará cuando el control pierda el foco del teclado, lo que indicará el final de la edición.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

edit_text

Establece el texto del control de edición.

```
void
edit_text(Edit *edit,
          const char_t *text);
```

edit — El edit.

text — Cadena C UTF8 terminada en carácter nulo `'\0'`.

edit_font

Establece la fuente del control edit.

```
void
edit_font(Edit *edit,
          const Font *font);
```

edit El edit.

font Fuente tipográfica.

edit_align

Establece la alineación del texto.

```
void
edit_align(Edit *edit,
           const align_t align);
```

edit El edit.

align La alineación.

edit_passmode

Activa el modo contraseña, que ocultará los caracteres tecleados.

```
void
edit_passmode(Edit *edit,
              const bool_t passmode);
```

edit El edit.

passmode Activa o desactiva el modo contraseña.

edit_editable

Activa o desactiva la edición en el control.

```
void
edit_editable(Edit *edit,
              const bool_t is_editable);
```

edit El edit.

is_editable `TRUE` permitirá editar el texto (por defecto).

edit_autoselect

Activa o desactiva la autoselección del texto.

```
void
edit_autoselect(Edit *edit,
                const bool_t autoselect);
```

edit El edit.

autoselect `TRUE` el texto del control será totalmente seleccionado cuando reciba el foco.

Observaciones:

Por defecto es `FALSE`.

edit_tooltip

Asigna un tooltip al control edit.

```
void
edit_tooltip(Edit *edit,
             const char_t *text);
```

edit El edit.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

edit_color

Establece el color del texto.

```
void
edit_color(Edit *edit,
           const color_t color);
```

edit El edit.

color El color del texto.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

edit_color_focus

Establece el color del texto, cuando el control tiene el foco del teclado.

```
void
edit_color_focus(Edit *edit,
                 const color_t color);
```

edit El edit.

color El color del texto.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

edit_bgcolor

Establece el color de fondo.

```
void
edit_bgcolor(Edit *edit,
             const color_t color);
```

edit El edit.

color El color de fondo.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

edit_bgcolor_focus

Establece el color de fondo, cuando el control tiene el foco del teclado.

```
void
edit_bgcolor_focus(Edit *edit,
                   const color_t color);
```

edit El edit.

color El color de fondo.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

edit_phtext

Establece un texto explicativo para cuando el control está en blanco *(placeholder)*.

```
void
edit_phtext(Edit *edit,
            const char_t *text);
```

edit El edit.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

edit_phcolor

Establece el color del texto placeholder.

```
void
edit_phcolor(Edit *edit,
             const color_t color);
```

edit El edit.

color El color del texto.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

edit_phstyle

Establece el estilo de la fuente para el placeholder.

```
void
edit_phstyle(Edit *edit,
             const uint32_t fstyle);
```

edit El edit.

fstyle Combinación de valores de `fstyle_t`.

edit_get_text

Obtiene el texto del control.

```
const char_t*
edit_get_text(const Edit *edit);
```

edit El edit.

Retorna:

Cadena C UTF8 terminada en carácter nulo `'\0'`.

combo_create

Crea un control combo.

```
Combo*
combo_create(void);
```

Retorna:

El combo.

combo_OnFilter

Establece un manejador para filtrar el texto mientras se edita.

```
void
combo_OnFilter(Combo *combo,
               Listener *listener);
```

combo	El combo.
listener	Función *callback* que se llamará tras cada pulsación de tecla. En `EvTextFilter` de `event_result` se devolverá el texto filtrado.

Observaciones:

Ver *"Filtrar textosFiltrar textos"* (Página 313) y *"Eventos GUIEventos GUI"* (Página 306).

combo_OnChange

Establece un manejador para detectar cuando el texto ha cambiado.

```
void
combo_OnChange(Combo *combo,
               Listener *listener);
```

combo	El combo.
listener	Función *callback* que se llamará cuando el control pierda el foco del teclado, lo que indicará el final de la edición.

Observaciones:

Este evento se lanzará también al seleccionar un elemento de la lista, señal que el texto ha cambiado en la caja de edición. Ver *"Eventos GUIEventos GUI"* (Página 306).

combo_text

Establece el texto de edición del combo.

```
void
combo_text(Combo *combo,
           const char_t *text);
```

combo El combo.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

combo_align

Establece la alineación del texto.

```
void
combo_align(Combo *combo,
            const align_t align);
```

combo El combo.

align La alineación.

combo_tooltip

Asigna un tooltip al control combo.

```
void
combo_tooltip(Combo *combo,
              const char_t *text);
```

combo El combo.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

combo_color

Establece el color del texto del combo.

```
void
combo_color(Combo *combo,
            const color_t color);
```

combo El combo.

color El color del texto.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

combo_color_focus

Establece el color del texto, cuando el control tiene el foco del teclado.

```
void
combo_color_focus(Combo *combo,
                  const color_t color);
```

combo El combo.

color El color del texto.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

combo_bgcolor

Establece el color de fondo.

```
void
combo_bgcolor(Combo *combo,
              const color_t color);
```

combo El combo.

color El color de fondo.

Observaciones:

Los valores RGB puede que no sean del todo portables. Ver *"Colores"* (Página 283).

combo_bgcolor_focus

Establece el color de fondo cuando el control tiene el foco del teclado.

```
void
combo_bgcolor_focus(Combo *combo,
                    const color_t color);
```

combo El combo.

color El color de fondo.

combo_phtext

Establece un texto explicativo para cuando el control está en blanco.

```
void
combo_phtext(Combo *combo,
             const char_t *text);
```

combo El combo.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

combo_phcolor

Establece el color del texto placeholder.

```
void
combo_phcolor(Combo *combo,
              const color_t color);
```

combo El combo.

color El color del texto.

combo_phstyle

Establece el estilo de la fuente para el placeholder.

```
void
combo_phstyle(Combo *combo,
              const uint32_t fstyle);
```

combo El combo.

fstyle Combinación de valores de `fstyle_t`.

combo_get_text

Obtiene el texto del control.

```
const char_t*
combo_get_text(const Combo *combo);
```

combo El combo.

Retorna:

Cadena C UTF8 terminada en carácter nulo `'\0'`.

combo_count

Obtiene el número de elementos en la lista desplegable.

```
uint32_t
combo_count(const Combo *combo);
```

combo El combo.

Retorna:

El número de elementos.

combo_add_elem

Añade un nuevo elemento a la lista desplegable.

```
void
combo_add_elem(Combo *combo,
               const char_t *text,
               const Image *image);
```

combo El combo.

text El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image Icono asociado al elemento o el identificador del recurso. Por cuestión de espacio, se escalará a un alto máximo de 16 píxeles.

combo_set_elem

Edita un elemento de la lista desplegable.

```
void
combo_set_elem(Combo *combo,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

combo El combo.

index El índice del elemento a sustituir.

text El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image Icono asociado al elemento o el identificador del recurso. Por cuestión de espacio, se escalará a un alto máximo de 16 píxeles.

combo_ins_elem

Inserta un elemento en la lista desplegable.

```
void
combo_ins_elem(Combo *combo,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

combo — El combo.

index — La posición de inserción.

text — El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image — Icono asociado al elemento o el identificador del recurso. Por cuestión de espacio, se escalará a un alto máximo de 16 píxeles.

combo_del_elem

Elimina un elemento de la lista desplegable.

```
void
combo_del_elem(Combo *combo,
               const uint32_t index);
```

combo — El combo.

index — El índice del elemento a eliminar.

combo_duplicates

Impide que hayan textos duplicados en la lista desplegable.

```
void
combo_duplicates(Combo *combo,
                 const bool_t duplicates);
```

combo — El combo.

duplicates — `TRUE` para permitir textos duplicados.

listbox_create

Crea un nuevo control de lista.

```
ListBox*
listbox_create(void);
```

Retorna:

El ListBox recién creado.

listbox_OnSelect

Establece un manejador para la selección de un nuevo elemento.

```
void
listbox_OnSelect(ListBox *listbox,
                 Listener *listener);
```

listbox El ListBox.

listener Función *callback* que se llamará tras seleccionar un nuevo elemento de la lista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

listbox_size

Establece el tamaño por defecto de la lista.

```
void
listbox_size(ListBox *listbox,
             const S2Df size);
```

listbox El ListBox.

size El tamaño.

Observaciones:

Corresponde al *"Dimensionado naturalDimensionado natural"* (Página 336) del control. Por defecto 128x128.

listbox_checkbox

Muestra u oculta los checkboxes a la izquierda de los elementos.

```
void
listbox_checkbox(ListBox *listbox,
                 const bool_t show);
```

listbox El ListBox.

show `TRUE` para visualizarlos.

listbox_multisel

Habilita la selección múltiple.

```
void
listbox_multisel(ListBox *listbox,
                 const bool_t multisel);
```

listbox El ListBox.

multisel `TRUE` para permitir varios elementos seleccionados al mismo tiempo.

listbox_add_elem

Añade un nuevo elemento.

```
void
listbox_add_elem(ListBox *listbox,
                 const char_t *text,
                 const Image *image);
```

listbox El ListBox.

text El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image Icono asociado al elemento o el identificador del recurso.

listbox_set_elem

Edita un elemento de la lista.

```
void
listbox_set_elem(ListBox *listbox,
                 const uint32_t index,
                 const char_t *text,
                 const Image *image);
```

listbox El ListBox.

index El índice del elemento a sustituir.

text El texto del elemento en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

image Icono asociado al elemento o el identificador del recurso.

listbox_clear

Elimina todos los elementos de la lista.

```
void
listbox_clear(ListBox *listbox);
```

listbox El ListBox.

listbox_color

Establece el color del texto de un elemento.

```
void
listbox_color(ListBox *listbox,
              const color_t color);
```

listbox El ListBox.

color El color. Por defecto kCOLOR_DEFAULT.

listbox_select

Selecciona un elemento desde el código del programa.

```
void
listbox_select(ListBox *listbox,
               const uint32_t index,
               const bool_t select);
```

listbox El ListBox.

index El índice del elemento a seleccionar.

select Seleccionar o de-seleccionar.

Observaciones:

Si la selección múltiple no está habilitada, seleccionar un elemento implica de-seleccionar todos los demás.

listbox_check

Marca o desmarca el check del elemento desde el código del programa.

```
void
listbox_check(ListBox *listbox,
              const uint32_t index,
              const bool_t check);
```

listbox El ListBox.

index El índice del elemento.

check Marcar o desmarcar.

Observaciones:

Marcar un elemento es independiente de seleccionarlo. Se pueden marcar los elementos aunque los checkboxes no estén visibles. Ver `listbox_checkbox`.

listbox_count

Devuelve el número de elementos de la lista.

```
uint32_t
listbox_count(const ListBox *listbox);
```

listbox El ListBox.

Retorna:

El número de elementos.

listbox_text

Devuelve el texto de un elemento.

```
const char_t*
listbox_text(const ListBox *listbox);
```

listbox El ListBox.

Retorna:

El texto en UTF-8 terminado en carácter nulo '\0'.

listbox_selected

Devuelve si un elemento está o no seleccionado.

```
bool_t
listbox_selected(const ListBox *listbox);
```

listbox El ListBox.

Retorna:

El estado de la selección.

listbox_checked

Devuelve si un elemento está o no marcado.

```
bool_t
listbox_checked(const ListBox *listbox);
```

listbox El ListBox.

Retorna:

El estado del checkbox.

Observaciones:

Marcar un elemento es independiente de seleccionarlo. Se pueden marcar los elementos aunque los checkboxes no estén visibles. Ver `listbox_checkbox`.

updown_create

Crea un control updown.

```
UpDown*
updown_create(void);
```

Retorna:

El updown.

updown_OnClick

Establece un manejador para la pulsación del botón.

```
void
updown_OnClick(UpDown *updown,
               Listener *listener);
```

updown El updown.

listener Función *callback* que se llamará tras hacer clic.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

updown_tooltip

Establece un tooltip para el botón. Es un pequeño texto explicativo que aparecerá cuando el ratón esté sobre el control.

```
void
updown_tooltip(UpDown *updown,
               const char_t *text);
```

updown El updown.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

slider_create

Crea un nuevo control deslizador.

```
Slider*
slider_create(void);
```

Retorna:

El slider.

slider_vertical

Crea un nuevo slider en vertical.

```
Slider*
slider_vertical(void);
```

Retorna:

El slider.

slider_OnMoved

Establece un manejador para el movimiento del slider.

```
void
slider_OnMoved(Slider *slider,
               Listener *listener);
```

slider El slider.

listener Función *callback* que se llamará continuamente mientras el usuario mueva un slider.

Observaciones:

`EvSlider` contiene los parámetros del evento, ver *"Eventos GUIEventos GUI"* (Página 306).

slider_tooltip

Establece un tooltip para el slider. Es un pequeño texto explicativo que aparecerá cuando el ratón esté sobre el control.

```
void
slider_tooltip(Slider *slider,
               const char_t *text);
```

slider El slider.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

slider_steps

Cambia el slider de rango continuo a intervalos discretos.

```
void
slider_steps(Slider *slider,
             const uint32_t steps);
```

slider El slider.

steps Número de intervalos. Utiliza `UINT32_MAX` para volver al rango continuo.

slider_value

Establece la posición del slider.

```
void
slider_value(Slider *slider,
             const real32_t value);
```

slider El slider.

value La posición entre 0.0 y 1.0.

slider_get_value

Obtiene la posición del slider.

```
real32_t
slider_get_value(const Slider *slider);
```

slider El slider.

Retorna:

La posición normalizada entre 0.0 y 1.0.

progress_create

Crea una barra de progreso.

```
Progress*
progress_create(void);
```

Retorna:

El progress.

progress_undefined

Establece la barra de progreso como indefinida.

```
void
progress_undefined(Progress *progress,
                   const bool_t running);
```

progress El progress.

running `TRUE` para activar la animación.

progress_value

Establece la posición del progress.

```
void
progress_value(Progress *progress,
               const real32_t value);
```

progress El progress.

value La posición entre 0.0 y 1.0.

view_create

Crea una nueva vista personalizada.

```
View*
view_create(void);
```

Retorna:

El control vista.

view_scroll

Crea una nueva vista con barras de scroll.

```
View*
view_scroll(void);
```

Retorna:

El control vista.

view_data

Asocia datos de usuario con la vista.

```
void
view_data(View *view,
          type **data,
          FPtr_destroy func_destroy_data,
          type);
```

view	La vista.
data	Datos de usuario.
func_destroy_data	Destructor de los datos de usuario. Será llamado al destruir la vista.
type	Tipo de datos de usuario.

view_get_data

Obtiene los datos de usuario asociados con la vista.

```
type*
view_get_data(const View *view,
              type);
```

view	La vista.
type	Tipo de datos de usuario.

Retorna:

Los datos de usuario.

view_size

Establece el tamaño por defecto de la vista.

```
void
view_size(View *view,
          const S2Df size);
```

view La vista.

size El tamaño.

Observaciones:

Corresponde al *"Dimensionado naturalDimensionado natural"* (Página 336) del control. Por defecto 128x128.

view_OnDraw

Establece un manejador para dibujar en la vista.

```
void
view_OnDraw(View *view,
            Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cada vez que deba refrescarse el dibujo.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnSize

Establece un manejador para el cambio de tamaño.

```
void
view_OnSize(View *view,
            Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cada vez que la vista cambie de tamaño.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnEnter

Establece un manejador para la entrada del ratón.

```
void
view_OnEnter(View *view,
             Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cuando el cursor del ratón entre en el área de la vista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnExit

Establece un manejador para la salida del ratón.

```
void
view_OnExit(View *view,
            Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cuando el cursor del ratón salga del área de la vista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnMove

Establece un manejador para el movimiento del ratón.

```
void
view_OnMove(View *view,
            Listener *listener);
```

view La vista.

listener Función *callback* que se llamará a medida que el cursor del ratón se desplace por encima de la vista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnDown

Establece un manejador para la pulsación de un botón del ratón.

```
void
view_OnDown(View *view,
            Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cuando se pulse un botón.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnUp

Establece un manejador para la liberación de un botón del ratón.

```
void
view_OnUp(View *view,
          Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cuando se libere un botón.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnClick

Establece un manejador para el clic del ratón.

```
void
view_OnClick(View *view,
             Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cada vez que se haga clic sobre la vista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnDrag

Establece un manejador para el arrastre del ratón.

```
void
view_OnDrag(View *view,
            Listener *listener);
```

view — La vista.

listener — Función *callback* que se llamará mientras se esté arrastrando el cursor del ratón sobre la vista.

Observaciones:

"Arrastrar" es desplazar el ratón con uno de los botones pulsados. Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnWheel

Establece un manejador para la ruedecilla del ratón.

```
void
view_OnWheel(View *view,
             Listener *listener);
```

view — La vista.

listener — Función *callback* que se llamará cuando se mueva la ruedecilla del ratón sobre la vista.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnKeyDown

Establece un manejador para la pulsación de una tecla.

```
void
view_OnKeyDown(View *view,
               Listener *listener);
```

view — La vista.

listener — Función *callback* que se llamará cuando se pulse una tecla y la vista tenga el foco del teclado.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnKeyUp

Establece un manejador para la liberación de una tecla.

```
void
view_OnKeyUp(View *view,
             Listener *listener);
```

view — La vista.

listener — Función *callback* que se llamará cuando se libere una tecla y la vista tenga el foco del teclado.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

view_OnFocus

Establece un manejador para el foco del teclado.

```
void
view_OnFocus(View *view,
             Listener *listener);
```

view — La vista.

listener — Función *callback* que se llamará cuando se reciba o se pierda el foco del teclado.

Observaciones:

Ver *"Uso del tecladoUso del teclado"* (Página 322) y *"Eventos GUIEventos GUI"* (Página 306).

view_keybuf

Establece un búfer de teclado para la consulta síncrona o asíncrona del estado de las teclas.

```
void
view_keybuf(View *view,
            Keybuf *buffer);
```

view — La vista.

buffer — Bufer de teclado que será mantenido por la vista, capturando los eventos `OnKeyDown` y `OnKeyUp`.

Observaciones:

Solo mantiene una referencia al búfer, que deberá ser destruido por el objeto que lo creó. Ver *"Búfer de teclado"* (Página 236). La aplicación podrá seguir recibiendo eventos de teclado a través de `view_OnKeyDown` y `view_OnKeyUp`.

view_get_size

Obtiene el tamaño actual de la vista.

```
void
view_get_size(const View *view,
              S2Df *size);
```

view La vista.

size El tamaño.

view_content_size

Establece el tamaño del área de dibujo cuando existen barras de scroll.

```
void
view_content_size(View *view,
                  const S2Df size);
```

view La vista.

size El tamaño interno del área de dibujo.

Observaciones:

Cuando se crea una vista con scroll, este método indica la totalidad del área de dibujo. El control lo utilizará para dimensionar y posicionar las barras de scroll.

view_scroll_x

Desplaza la barra de scroll horizontal.

```
void
view_scroll_x(View *view,
              const real32_t pos);
```

view La vista.

pos Nueva posición de la barra horizontal.

view_scroll_y

Desplaza la barra de scroll vertical.

```
void
view_scroll_y(View *view,
              const real32_t pos);
```

view La vista.

pos Nueva posición de la barra vertical.

view_viewport

Obtiene las dimensiones del área visible de la vista.

```
void
view_viewport(const View *view,
              V2Df *pos,
              S2Df *size);
```

view La vista.

pos La posición del viewport. Puede ser `NULL`.

size El tamaño del viewport. Puede ser `NULL`.

Observaciones:

Si la vista no tiene barras de scroll, `pos` será `(0,0)`.

view_point_scale

Obtiene el escalado del punto.

```
void
view_point_scale(const View *view,
                 real32_t *scale);
```

view La vista.

scale El escalado.

Observaciones:

El tamaño de la vista y coordenadas de dibujo se expresan en puntos, que normalmente corresponden con píxeles (1pt = 1px). En *"Pantallas retinaPantallas retina"* (Página 270) puede ocurrir que (1pt = 2px). Si bien los *"Contextos 2D"* (Página 263) gestionan esto automáticamente, es posible que necesitemos saber la cantidad de píxeles para crear otro tipo de *framebuffers* (OpenGL, DirectX, etc). Pixels = `view_get_size` * `view_point_scale`.

view_update

Manda una orden al sistema operativo que la vista debe ser refrescada.

```
void
view_update(View *view);
```

view La vista.

textview_create

Crea una vista de texto.

```
TextView*
textview_create(void);
```

Retorna:

La vista de texto.

textview_size

Establece el tamaño por defecto de la vista.

```
void
textview_size(TextView *view,
              const S2Df size);
```

view La vista.

size El tamaño.

Observaciones:

Corresponde al *"Dimensionado naturalDimensionado natural"* (Página 336) del control. Por defecto 245x144.

textview_clear

Borra todo el contenido de la vista.

```
void
textview_clear(TextView *view);
```

view La vista.

textview_printf

Escribe texto en la vista, utilizando el formato del `printf`.

```
uint32_t
textview_printf(TextView *view,
                const char_t *format,
                ...);
```

```
textview_printf(view, Code: %-10s Price %5.2f\n", code, price);
```

view	La vista.
format	Cadena con el formato tipo-printf con un número variable de parámetros.
...	Argumentos o variables del printf.

Retorna:

El número de bytes escritos.

textview_writef

Escribe una cadena C UTF8 en la vista.

```
void
textview_writef(TextView *view,
                const char_t *str);
```

view	La vista.
str	Cadena C UTF8 terminada en carácter nulo `'\0'`.

textview_rtf

Inserta texto en el formato **RTF** de Microsoft.

```
void
textview_rtf(TextView *view,
             Stream *rtf_in);
```

view	La vista.
rtf_in	Stream de lectura con el contenido RTF.

textview_units

Establece las unidades del texto.

```
void
textview_units(TextView *view,
               const uint32_t units);
```

view La vista.

units Unidades `ekFPIXELS` ó `ekFPOINTS`.

Observaciones:

`ekFPOINTS` es el valor por defecto y el utilizado normalmente por los procesadores de texto. Ver *"Tamaño en puntosTamaño en puntos"* (Página 298).

textview_family

Establece la familia tipográfica del texto (*"Arial", "Times New Roman", "Helvetica", etc*).

```
void
textview_family(TextView *view,
                const char_t *family);
```

view La vista.

family La familia tipográfica.

Observaciones:

No todas las familias estarán presentes en todas las plataformas. Utiliza `font_exists_family` o `font_installed_families` para comprobarlo.

textview_fsize

Establece el tamaño del texto.

```
void
textview_fsize(TextView *view,
               const real32_t size);
```

view La vista.

size El tamaño.

Observaciones:

El valor está condicionado a las unidades establecidas en `textview_units`.

textview_fstyle

Establece el estilo del texto.

```
void
textview_fstyle(TextView *view,
                const uint32_t fstyle);
```

view La vista.

fstyle Combinación de ekFBOLD, ekFITALIC, ekFSTRIKEOUT, ekFUNDERLINE ekFSUBSCRIPT, ekFSUPSCRIPT. Para anular cualquier estilo previo utiliza ekFNORMAL.

textview_color

Establece el color del texto.

```
void
textview_color(TextView *view,
               const color_t color);
```

view La vista.

color El color. Utiliza kCOLOR_DEFAULT para restablecer el color por defecto.

textview_bgcolor

Establece el color de fondo del texto.

```
void
textview_bgcolor(TextView *view,
                 const color_t color);
```

view La vista.

color El color. Utiliza kCOLOR_DEFAULT para restablecer el color por defecto.

textview_pgcolor

Establece el color de fondo del control.

```
void
textview_pgcolor(TextView *view,
                 const color_t color);
```

view La vista.

color El color. Utiliza kCOLOR_DEFAULT para restablecer el color por defecto.

textview_halign

Establece la alineación del texto en un párrafo.

```
void
textview_halign(TextView *view,
                const align_t align);
```

view La vista.

align La alineación. Por defecto `ekLEFT`.

textview_lspacing

Establece el espaciado inter-lineal del párrafo.

```
void
textview_lspacing(TextView *view,
                  const real32_t scale);
```

view La vista.

scale Factor de escala en la altura de la fuente. `1` es el valor por defecto, `2` el doble de dicha altura, `3` el triple, etc. Valores intermedios también son válidos (pe. `1.25`).

textview_bfspace

Establece un espacio vertical antes del párrafo.

```
void
textview_bfspace(TextView *view,
                 const real32_t space);
```

view La vista.

space El espacio en las unidades preestablecidas.

textview_afspace

Establece un espacio vertical después del párrafo.

```
void
textview_afspace(TextView *view,
                 const real32_t space);
```

view La vista.

space El espacio en las unidades preestablecidas.

textview_scroll_down

Fuerza un scroll hasta el límite inferior del contenido de la vista.

```
void
textview_scroll_down(TextView *view);
```

view La vista.

Observaciones:

Hará visible el último contenido insertado.

textview_editable

Establece si el texto del control es o no editable.

```
void
textview_editable(TextView *view,
                  const bool_t is_editable);
```

view La vista.

is_editable `TRUE` permitirá editar el texto. Por defecto `FALSE`.

imageview_create

Crea un control de vista de imágenes.

```
ImageView*
imageview_create(void);
```

Retorna:

La vista de imagen.

imageview_size

Establece el tamaño por defecto del control.

```
void
imageview_size(ImageView *view,
               const S2Df size);
```

view La vista.

size El tamaño.

imageview_scale

Establece el escalado a aplicar a la imagen.

```
void
imageview_scale(ImageView *view,
                const gui_scale_t scale);
```

view La vista.

scale El escalado.

imageview_image

Establece la imagen que se mostrará en el control.

```
void
imageview_image(ImageView *view,
                const Image *image);
```

view La vista.

image La imagen a mostrar.

imageview_OnClick

Establece un manejador para el evento clic sobre la imagen.

```
void
imageview_OnClick(ImageView *view,
                  Listener *listener);
```

view La vista.

listener Función *callback* que se llamará tras hacer clic.

imageview_OnOverDraw

Permite dibujar un *overlay* sobre la imagen cuando el ratón esté encima de ella.

```
void
imageview_OnOverDraw(ImageView *view,
                     Listener *listener);
```

view La vista.

listener Función *callback* que se llamará cuando el ratón esté sobre la imagen. Aquí incluiremos el código de dibujo adicional.

tableview_create

Crea un nuevo control de tabla.

```
TableView*
tableview_create(void);
```

Retorna:

El TableView.

tableview_OnData

Establece un manejador para leer datos desde la aplicación.

```
void
tableview_OnData(TableView *view,
                 Listener *listener);
```

view	El TableView.
listener	Función *callback* que se llamará cada vez que la tabla deba actualizar el contenido.

Observaciones:

Ver *"Conexión de datosConexión de datos"* (Página 326).

tableview_OnSelect

Notifica que la selección ha cambiado.

```
void
tableview_OnSelect(TableView *view,
                   Listener *listener);
```

view	El TableView.
listener	Función *callback* que se llamará cada vez que cambie la selección en la tabla.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

tableview_OnHeaderClick

Notifica cada vez que se pulse sobre una cabecera.

```
void
tableview_OnHeaderClick(TableView *view,
                        Listener *listener);
```

view El TableView.

listener Función *callback* que se llamará cada vez que se haga clic sobre una cabecera de la tabla.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_font

Establece la fuente general para toda la tabla.

```
void
tableview_font(TableView *view,
               const Font *font);
```

view El TableView.

font Fuente tipográfica.

tableview_size

Establece el tamaño por defecto del control tabla.

```
void
tableview_size(TableView *view,
               const S2Df size);
```

view El TableView.

size El tamaño.

Observaciones:

Corresponde al *"Dimensionado naturalDimensionado natural"* (Página 336) del control. Por defecto 256x128.

tableview_new_column_text

Añade una nueva columna a la tabla.

```
uint32_t
tableview_new_column_text(TableView *view);
```

view El TableView.

Retorna:

El identificador (índice) de columna.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_column_width

Establece el ancho de una columna.

```
void
tableview_column_width(TableView *view,
                       const uint32_t column_id,
                       const real32_t width);
```

view El TableView.

column_id El identificador de columna.

width El ancho de columna.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_column_limits

Establece los límites de tamaño de una columna.

```
void
tableview_column_limits(TableView *view,
                        const uint32_t column_id,
                        const real32_t min,
                        const real32_t max);
```

view El TableView.

column_id El identificador de columna.

min El ancho mínimo.

max El ancho máximo.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_column_resizable

Estable si una columna es redimensionable o no.

```
void
tableview_column_resizable(TableView *view,
                           const uint32_t column_id,
                           const bool_t resizable);
```

view	El TableView.
column_id	El identificador de columna.
resizable	TRUE si se puede redimensionar.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_column_freeze

Permite fijar las primeras columnas de la tabla.

```
void
tableview_column_freeze(TableView *view,
                        const uint32_t last_column_id);
```

view	El TableView.
last_column_id	El identificador de la última columna fijada.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_header_title

Establece el texto de la cabecera de una columna.

```
void
tableview_header_title(TableView *view,
                       const uint32_t column_id,
                       const char_t *text);
```

view	El TableView.
column_id	El identificador de columna.
text	El texto en UTF-8 o el identificador del recurso. *"Recursos"* (Página 131).

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_header_align

Establece la alineación del texto de la cabecera.

```
void
tableview_header_align(TableView *view,
                       const uint32_t column_id,
                       const align_t align);
```

view	El TableView.
column_id	El identificador de columna.
align	La alineación.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_header_visible

Establece si la cabecera de la tabla es visible o no.

```
void
tableview_header_visible(TableView *view,
                         const bool_t visible);
```

view	El TableView.
visible	`TRUE` para mostrar la cabecera.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_header_clickable

Establece si la cabecera de la tabla se puede pulsar como un botón.

```
void
tableview_header_clickable(TableView *view,
                           const bool_t clickable);
```

view	El TableView.
clickable	`TRUE` para permitir pulsaciones.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_header_resizable

Establece si la cabecera permite el redimensionado de columnas.

```
void
tableview_header_resizable(TableView *view,
                           const bool_t resizable);
```

view	El TableView.
resizable	TRUE si se puede redimensionar.

Observaciones:

Ver *"Configurar columnasConfigurar columnas"* (Página 332).

tableview_multisel

Establece el modo de selección de filas.

```
void
tableview_multisel(TableView *view,
                   const bool_t multisel,
                   const bool_t preserve);
```

view	El TableView.
multisel	TRUE para permitir selección múltiple.
preserve	TRUE para preservar la selección mientras navegamos.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

tableview_grid

Establece el dibujo de las líneas interiores.

```
void
tableview_grid(TableView *view,
               const bool_t hlines,
               const bool_t vlines);
```

view	El TableView.
hlines	`TRUE` para dibujar líneas horizontales.
vlines	`TRUE` para dibujar líneas verticales.

Observaciones:

Ver *"Dibujo de rejillaDibujo de rejilla"* (Página 332).

tableview_update

Sincroniza la tabla con el origen de datos.

```
void
tableview_update(TableView *view);
```

view	El TableView.

Observaciones:

Ver *"Conexión de datosConexión de datos"* (Página 326). Deberemos llamar a esta función desde la aplicación siempre que cambien los datos vinculados con la tabla, con el fin de actualizar la vista.

tableview_select

Selecciona filas en la tabla.

```
void
tableview_select(TableView *view,
                 const uint32_t *rows,
                 const uint32_t n);
```

view	El TableView.
rows	Vector de índices de línea.
n	Número de elementos del vector.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

tableview_deselect

De-selecciona filas en la tabla.

```
void
tableview_deselect(TableView *view,
                   const uint32_t *rows,
                   const uint32_t n);
```

view El TableView.

rows Vector de índices de línea.

n Número de elementos del vector.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

tableview_deselect_all

De-selecciona todas las filas de la tabla.

```
void
tableview_deselect_all(TableView *view);
```

view El TableView.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

tableview_selected

Devuelve las filas actualmente seleccionadas.

```
const ArrSt(uint32_t)*
tableview_selected(const TableView *view);
```

view El TableView.

Retorna:

Array con los índices de las filas seleccionadas.

Observaciones:

Ver *"Selección múltipleSelección múltiple"* (Página 331).

splitview_horizontal

Crea un splitview con división horizontal.

```
SplitView*
splitview_horizontal(void);
```

Retorna:

La vista dividida recién creada.

splitview_vertical

Crea un splitview con división vertical.

```
SplitView*
splitview_vertical(void);
```

Retorna:

La vista dividida recién creada.

splitview_size

Establece el tamaño por defecto de la vista.

```
void
splitview_size(SplitView *split,
               const S2Df size);
```

split La vista.

size El tamaño.

Observaciones:

Corresponde al *"Dimensionado naturalDimensionado natural"* (Página 336) del control. Por defecto 128x128.

splitview_view

Añade una vista personalizada al splitview.

```
void
splitview_view(SplitView *split,
               View *view);
```

split El splitview.

view La vista personalizada.

Observaciones:

Ver *"Añadir controlesAñadir controles"* (Página 333).

splitview_text

Añade una vista de texto al splitview.

```
void
splitview_text(SplitView *split,
               TextView *view);
```

split El splitview.

view La vista de texto.

Observaciones:

Ver *"Añadir controlesAñadir controles"* (Página 333).

splitview_split

Añade un splitview (hijo) al splitview.

```
void
splitview_split(SplitView *split,
                SplitView *child);
```

split El splitview.

child El splitview a añadir.

Observaciones:

Ver *"Añadir controlesAñadir controles"* (Página 333).

splitview_panel

Añade un panel al splitview.

```
void
splitview_panel(SplitView *split,
                Panel *panel);
```

split El splitview.

panel El panel.

Observaciones:

Ver *"Añadir controlesAñadir controles"* (Página 333).

splitview_pos

Establece la posición del separador de vistas.

```
void
splitview_pos(SplitView *split,
              const real32_t pos);
```

split El splitview.

pos La nueva posición del separador.

Observaciones:

Ver *"Modos de divisiónModos de división"* (Página 334).

layout_create

Crea un nuevo layout especificando el número de columnas y filas.

```
Layout*
layout_create(const uint32_t ncols,
              const uint32_t nrows);
```

ncols El número de columnas.

nrows El número de filas.

Retorna:

El nuevo layout.

layout_cell

Obtiene una celda del layout.

```
Cell*
layout_cell(Layout *layout,
            const uint32_t col,
            const uint32_t row);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

Retorna:

La celda.

layout_label

Inserta un control `Label` en un layout.

```
void
layout_label(Layout *layout,
             Label *label,
             const uint32_t col,
             const uint32_t row);
```

layout	El layout.
label	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_button

Inserta un control `Button` en un layout.

```
void
layout_button(Layout *layout,
              Button *button,
              const uint32_t col,
              const uint32_t row);
```

layout	El layout.
button	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_popup

Inserta un control `PopUp` en un layout.

```
void
layout_popup(Layout *layout,
             PopUp *popup,
             const uint32_t col,
             const uint32_t row);
```

layout	El layout.
popup	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_edit

Inserta un control `Edit` en un layout.

```
void
layout_edit(Layout *layout,
            Edit *edit,
            const uint32_t col,
            const uint32_t row);
```

layout	El layout.
edit	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_combo

Inserta un control `Combo` en un layout.

```
void
layout_combo(Layout *layout,
             Combo *combo,
             const uint32_t col,
             const uint32_t row);
```

layout	El layout.
combo	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_listbox

Inserta un control `ListBox` en un layout.

```
void
layout_listbox(Layout *layout,
               ListBox *list,
               const uint32_t col,
               const uint32_t row);
```

layout	El layout.
list	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_updown

Inserta un control `UpDown` en un layout.

```
void
layout_updown(Layout *layout,
              UpDown *updown,
              const uint32_t col,
              const uint32_t row);
```

layout	El layout.
updown	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_slider

Inserta un control `Slider` en un layout.

```
void
layout_slider(Layout *layout,
              Slider *slider,
              const uint32_t col,
              const uint32_t row);
```

layout	El layout.
slider	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_progress

Inserta un control `Progress` en un layout.

```
void
layout_progress(Layout *layout,
                Progress *progress,
                const uint32_t col,
                const uint32_t row);
```

layout	El layout.
progress	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_view

Inserta una vista `View` en un layout.

```
void
layout_view(Layout *layout,
            View *view,
            const uint32_t col,
            const uint32_t row);
```

layout	El layout.
view	La vista a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_textview

Inserta un control `TextView` en un layout.

```
void
layout_textview(Layout *layout,
                TextView *view,
                const uint32_t col,
                const uint32_t row);
```

layout	El layout.
view	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_imageview

Inserta un control `ImageView` en un layout.

```
void
layout_imageview(Layout *layout,
                 ImageView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	El layout.
view	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_tableview

Inserta un control `TableView` en un layout.

```
void
layout_tableview(Layout *layout,
                 TableView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	El layout.
view	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_splitview

Inserta un control `SplitView` en un layout.

```
void
layout_splitview(Layout *layout,
                 SplitView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	El layout.
view	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_panel

Inserta un control `Panel` en un layout.

```
void
layout_panel(Layout *layout,
             Panel *panel,
             const uint32_t col,
             const uint32_t row);
```

layout	El layout.
panel	El control a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_layout

Inserta un layout en una celda de otro layout.

```
void
layout_layout(Layout *layout,
              Layout *sublayout,
              const uint32_t col,
              const uint32_t row);
```

layout	El layout principal.
sublayout	El layout a insertar.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

layout_get_label

Obtiene el `Label` de una celda.

```
Label*
layout_get_label(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_button

Obtiene el `Button` de una celda.

```
Button*
layout_get_button(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_popup

Obtiene el `PopUp` de una celda.

```
PopUp*
layout_get_popup(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_edit

Obtiene el `Edit` de una celda.

```
Edit*
layout_get_edit(const Layout *layout,
                const uint32_t col,
                const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_combo

Obtiene el `Combo` de una celda.

```
Combo*
layout_get_combo(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_listbox

Obtiene el `ListBox` de una celda.

```
ListBox*
layout_get_listbox(const Layout *layout,
                   const uint32_t col,
                   const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_updown

Obtiene el `UpDown` de una celda.

```
UpDown*
layout_get_updown(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_slider

Obtiene el `Slider` de una celda.

```
Slider*
layout_get_slider(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_progress

Obtiene el `Progress` de una celda.

```
Progress*
layout_get_progress(const Layout *layout,
                    const uint32_t col,
                    const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_view

Obtiene el `View` de una celda.

```
View*
layout_get_view(const Layout *layout,
                const uint32_t col,
                const uint32_t row);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

Retorna:

La vista.

layout_get_textview

Obtiene el `TextView` de una celda.

```
TextView*
layout_get_textview(const Layout *layout,
                    const uint32_t col,
                    const uint32_t row);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_imageview

Obtiene el `ImageView` de una celda.

```
ImageView*
layout_get_imageview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_tableview

Obtiene el `TableView` de una celda.

```
TableView*
layout_get_tableview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_splitview

Obtiene el `SplitView` de una celda.

```
SplitView*
layout_get_splitview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_panel

Obtiene el `Panel` de una celda.

```
Panel*
layout_get_panel(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El control.

layout_get_layout

Obtiene el `Layout` de una celda.

```
Layout*
layout_get_layout(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout El layout.

col Columna, coordenada x de la celda.

row Fila, coordenada y de la celda.

Retorna:

El sublayout.

layout_taborder

Establece como se moverá el foco del teclado al pulsar `[TAB]`.

```
void
layout_taborder(Layout *layout,
                const gui_orient_t order);
```

layout El layout.

order Recorrido por filas o columnas.

Observaciones:

Ver *"TabstopsTabstops"* (Página 342).

layout_tabstop

Establece si una celda del layout recibirá o no el foco del teclado al navegar con `[TAB]-[SHIFT][TAB]`.

```
void
layout_tabstop(Layout *layout,
               const uint32_t col,
               const uint32_t row,
               const bool_t tabstop);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.
tabstop	Habilitar o deshabilitar el punto de parada en la celda.

Observaciones:

Ver *"TabstopsTabstops"* (Página 342).

layout_next_tabstop

Mueve el foco del teclado al siguiente control en la *tab-list*. Produce el mismo efecto que pulsar `[TAB]`.

```
void
layout_next_tabstop(Layout *layout);
```

layout	El layout.

Observaciones:

Solo funcionará si el layout está activo en una ventana. Equivalente a `window_next_tabstop`. Ver *"TabstopsTabstops"* (Página 342).

layout_previous_tabstop

Mueve el foco del teclado al anterior control en la *tab-list*. Produce el mismo efecto que pulsar `[SHIFT]+[TAB]`.

```
void
layout_previous_tabstop(Layout *layout);
```

layout	El layout.

Observaciones:

Solo funcionará si el layout está activo en una ventana. Equivalente a `window_previous_tabstop`. Ver *"TabstopsTabstops"* (Página 342).

layout_hsize

Establece un ancho fijo para la columna de un layout.

```
void
layout_hsize(Layout *layout,
             const uint32_t col,
             const real32_t width);
```

layout	El layout.
col	Índice de la columna.
width	Anchura.

layout_vsize

Fuerza un alto fijo para la fila de un layout.

```
void
layout_vsize(Layout *layout,
             const uint32_t row,
             const real32_t height);
```

layout	El layout.
row	Índice de la fila.
height	Altura.

layout_hmargin

Establece un margen inter-columna dentro del layout. Es la separación entre dos columnas consecutivas.

```
void
layout_hmargin(Layout *layout,
               const uint32_t col,
               const real32_t margin);
```

layout El layout.

col Índice de la columna. El índice 0 se refiere a la separación entre la columna 0 y la columna 1. `ncols-2` es el máximo valor aceptado.

margin Margen.

layout_vmargin

Establece un margen inter-fila dentro del layout. Es la separación entre dos filas consecutivas.

```
void
layout_vmargin(Layout *layout,
               const uint32_t row,
               const real32_t margin);
```

layout El layout.

row Índice de la fila. El índice 0 se refiere a la separación entre la fila 0 y la fila 1. `nrows-2` es el máximo valor aceptado.

margin Margen.

layout_hexpand

Establece la columna que se expandirá horizontalmente.

```
void
layout_hexpand(Layout *layout,
               const uint32_t col);
```

layout El layout.

col Índice de la columna.

Observaciones:

Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_hexpand2

Establece las dos columnas que se expandirán horizontalmente.

```
void
layout_hexpand2(Layout *layout,
                const uint32_t col1,
                const uint32_t col2,
                const real32_t exp);
```

layout	El layout.
col1	Índice de la columna 1.
col2	Índice de la columna 2.
exp	Expansión de `col1` entre 0 y 1.

Observaciones:

La expansión de `col2 = 1 - exp`. Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_hexpand3

Establece las tres columnas que se expandirán horizontalmente.

```
void
layout_hexpand3(Layout *layout,
                const uint32_t col1,
                const uint32_t col2,
                const uint32_t col3,
                const real32_t exp1,
                const real32_t exp2);
```

layout	El layout.
col1	Índice de la columna 1.
col2	Índice de la columna 2.
col3	Índice de la columna 3.
exp1	Expansión de `col1` entre 0 y 1.
exp2	Expansión de `col2` entre 0 y 1.

Observaciones:

`exp1 + exp2 < = 1`. La expansión de `col3 = 1 - exp1 - exp2`. Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_vexpand

Establece la fila que se expandirá verticalmente.

```
void
layout_vexpand(Layout *layout,
               const uint32_t row);
```

layout El layout.

row Índice de la fila.

Observaciones:

Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_vexpand2

Establece las dos filas que se expandirán verticalmente.

```
void
layout_vexpand2(Layout *layout,
                const uint32_t row1,
                const uint32_t row2,
                const real32_t exp);
```

layout El layout.

row1 Índice de la fila 1.

row2 Índice de la fila 2.

exp Expansión de `row1` entre 0 y 1.

Observaciones:

La expansión de `row2 = 1 - exp`. Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_vexpand3

Establece las tres filas que se expandirán verticalmente.

```
void
layout_vexpand3(Layout *layout,
                const uint32_t row1,
                const uint32_t row2,
                const uint32_t row3,
                const real32_t exp1,
                const real32_t exp2);
```

layout	El layout.
row1	Índice de la fila 1.
row2	Índice de la fila 2.
row3	Índice de la fila 3.
exp1	Expansión de `row1` entre 0 y 1.
exp2	Expansión de `row2` entre 0 y 1.

Observaciones:

`exp1 + exp2 < = 1`. La expansión de `row3 = 1 - exp1 - exp2`. Ver *"Expansión de celdasExpansión de celdas"* (Página 341).

layout_halign

Establece la alineación horizontal de una celda. Tendrá efecto cuando la columna sea más ancha que la celda.

```
void
layout_halign(Layout *layout,
              const uint32_t col,
              const uint32_t row,
              const align_t align);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.
align	Alineación horizontal.

layout_valign

Establece la alineación vertical de una celda. Tendrá efecto cuando la fila sea más alta que la celda.

```
void
layout_valign(Layout *layout,
              const uint32_t col,
              const uint32_t row,
              const align_t align);
```

layout	El layout.
col	Columna, coordenada x de la celda.
row	Fila, coordenada y de la celda.
align	Alineación vertical.

layout_show_col

Muestra u oculta una columna del layout.

```
void
layout_show_col(Layout *layout,
                const uint32_t col,
                const bool_t visible);
```

layout	El layout.
col	Índice de columna.
visible	Visible u oculta.

layout_show_row

Muestra u oculta una fila del layout.

```
void
layout_show_row(Layout *layout,
                const uint32_t row,
                const bool_t visible);
```

layout	El layout.
row	Índice de fila.
visible	Visible u oculta.

layout_margin

Establece un margen uniforme para el borde del layout.

```
void
layout_margin(Layout *layout,
              const real32_t mall);
```

layout	El layout.
mall	Margen para los cuatro lados (izquierda, derecha, arriba y abajo).

layout_margin2

Establece un margen horizontal y vertical para el borde del layout.

```
void
layout_margin2(Layout *layout,
               const real32_t mtb,
               const real32_t mlr);
```

layout El layout.

mtb Margen superior e inferior.

mlr Margen izquierdo y derecho.

layout_margin4

Establece márgenes para el borde del layout.

```
void
layout_margin4(Layout *layout,
               const real32_t mt,
               const real32_t mr,
               const real32_t mb,
               const real32_t ml);
```

layout El layout.

mt Margen del borde superior.

mr Margen del borde derecho.

mb Margen del borde inferior.

ml Margen del borde izquierdo.

layout_bgcolor

Asigna un color de fondo al layout.

```
void
layout_bgcolor(Layout *layout,
               const color_t color);
```

layout El layout.

color El color. Pasando `ekCOLOR_TRANSPARENT` se restablece el color por defecto.

layout_skcolor

Asigna un color al borde del layout.

```
void
layout_skcolor(Layout *layout,
               const color_t color);
```

layout El layout.

color El color. Pasando `ekCOLOR_TRANSPARENT` se restablece el color por defecto.

layout_update

Actualiza la ventana asociada con el layout.

```
void
layout_update(Layout *layout);
```

layout El layout.

Observaciones:

Es equivalente a llamar a `window_update`.

layout_dbind

Asocia un tipo `struct` con un layout.

```
void
layout_dbind(Layout *layout,
             Listener *listener,
             type);
```

layout El layout.

listener Notificará a través de este `listener` cada vez que el objeto cambie. Puede ser `NULL`.

type El tipo de **`struct`**.

Observaciones:

Ver *"GUI Data binding"* (Página 355).

layout_dbind_obj

Asocia un objeto con un layout para visualizarlo y editarlo.

```
void
layout_dbind_obj(Layout *layout,
                 type *obj,
                 type);
```

layout El layout.

obj El objeto a editar.

type El tipo del objeto.

Observaciones:

Ver *"GUI Data binding"* (Página 355).

layout_dbind_update

Actualiza la interfaz del objeto asociado al layout.

```
void
layout_dbind_update(Layout *layout,
                    type,
                    mtype,
                    mname);
```

layout El layout.

type El tipo de objeto.

mtype El tipo del campo a actualizar.

mname El nombre del campo a actualizar.

Observaciones:

Ver *"GUI Data binding"* (Página 355).

cell_label

Obtiene el label del interior de la celda.

```
Label*
cell_label(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_button

Obtiene el botón del interior de la celda.

```
Button*
cell_button(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_popup

Obtiene el popup del interior de la celda.

```
PopUp*
cell_popup(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_edit

Obtiene el edit del interior de la celda.

```
Edit*
cell_edit(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_combo

Obtiene el combo del interior de la celda.

```
Combo*
cell_combo(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_listbox

Obtiene el listbox del interior de la celda.

```
ListBox*
cell_listbox(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_updown

Obtiene el updown del interior de la celda.

```
UpDown*
cell_updown(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_slider

Obtiene el slider del interior de la celda.

```
Slider*
cell_slider(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_progress

Obtiene el progress del interior de la celda.

```
Progress*
cell_progress(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_view

Obtiene la vista del interior de la celda.

```
View*
cell_view(Cell *cell);
```

cell La celda.

Retorna:

La vista.

cell_textview

Obtiene el textview del interior de la celda.

```
TextView*
cell_textview(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_imageview

Obtiene el imageview del interior de la celda.

```
ImageView*
cell_imageview(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_tableview

Obtiene el tableview del interior de la celda.

```
TableView*
cell_tableview(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_splitview

Obtiene el splitview del interior de la celda.

```
SplitView*
cell_splitview(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_panel

Obtiene el panel del interior de la celda.

```
Panel*
cell_panel(Cell *cell);
```

cell La celda.

Retorna:

El control.

cell_layout

Obtiene el layout del interior de la celda.

```
Layout*
cell_layout(Cell *cell);
```

cell La celda.

Retorna:

El layout.

cell_enabled

Activa o desactiva una celda.

```
void
cell_enabled(Cell *cell,
             const bool_t enabled);
```

cell	La celda.
enabled	Habilitada o no.

cell_visible

Muestra u oculta una celda.

```
void
cell_visible(Cell *cell,
             const bool_t visible);
```

cell	La celda.
visible	Visible o no.

cell_focus

Establece el foco del teclado en la celda.

```
void
cell_focus(Cell *cell);
```

cell	La celda.

cell_padding

Establece un margen interior.

```
void
cell_padding(Cell *cell,
             const real32_t pall);
```

cell	La celda.
pall	Margen interior.

cell_padding2

Establece un margen interior.

```
void
cell_padding2(Cell *cell,
              const real32_t ptb,
              const real32_t plr);
```

cell La celda.

ptb Margen superior e inferior.

plr Margen izquierdo y derecho.

cell_padding4

Establece un margen interior.

```
void
cell_padding4(Cell *cell,
              const real32_t pt,
              const real32_t pr,
              const real32_t pb,
              const real32_t pl);
```

cell La celda.

pt Margen superior.

pr Margen derecho.

pb Margen inferior.

pl Margen izquierdo.

cell_dbind

Asocia una celda con el campo de un `struct`.

```
void
cell_dbind(Cell *cell,
           type,
           mtype,
           mname);
```

```
cell_dbind(cell, Product, String*, description);
```

cell	La celda.
type	El tipo de `struct`.
mtype	El tipo del campo del `struct`.
mname	El nombre del campo.

Observaciones:

Ver *"GUI Data binding"* (Página 355).

panel_create

Crea un panel.

```
Panel*
panel_create(void);
```

Retorna:

El nuevo panel.

panel_scroll

Crea un panel con barras de scroll.

```
Panel*
panel_scroll(const bool_t hscroll,
             const bool_t vscroll);
```

hscroll	`TRUE` si queremos barra de scroll horizontal.
vscroll	`TRUE` si queremos barra de scroll vertical.

Retorna:

El nuevo panel.

Observaciones:

Ver *"Entendiendo el dimensionado de panelesEntendiendo el dimensionado de paneles"* (Página 345).

panel_data

Asocia datos de usuario con el panel.

```
void
panel_data(Panel *panel,
           type **data,
           FPtr_destroy func_destroy_data,
           type);
```

panel	El panel.
data	Datos de usuario.
func_destroy_data	Destructor de los datos de usuario. Será llamado al destruir el panel.
type	Tipo de datos de usuario.

panel_get_data

Obtiene los datos de usuario asociados con el panel.

```
type*
panel_get_data(const Panel *panel,
               type);
```

panel	El panel.
type	Tipo de datos de usuario.

Retorna:

Los datos de usuario.

panel_size

Establece el tamaño por defecto del área visible de un panel.

```
void
panel_size(Panel *panel,
           const S2Df size);
```

panel	El panel.
size	El tamaño por defecto.

Observaciones:

Ver *"Entendiendo el dimensionado de panelesEntendiendo el dimensionado de paneles"* (Página 345).

panel_layout

Añade un layout a un panel.

```
uint32_t
panel_layout(Panel *panel,
             Layout *layout);
```

panel	El panel.
layout	El layout.

Retorna:

El índice del layout recién añadido.

panel_get_layout

Obtiene un layout de un panel.

```
Layout*
panel_get_layout(Panel *panel,
                 const uint32_t index);
```

panel	El panel.
index	El índice del layout.

Retorna:

El layout.

panel_visible_layout

Establece el layout activo dentro del panel.

```
void
panel_visible_layout(Panel *panel,
                     const uint32_t index);
```

panel	El panel.
index	El índice del layout.

Observaciones:

Para el cambio se haga efectivo, hay que llamar a `panel_update`.

panel_update

Refresca el contenido de la ventana que contiene el panel.

```
void
panel_update(Panel *panel);
```

panel El panel.

Observaciones:

Es equivalente a llamar a `window_update`.

panel_scroll_width

Obtiene el ancho de la barra de scroll del panel asociado.

```
real32_t
panel_scroll_width(const Panel *panel);
```

panel El panel.

Retorna:

El ancho de la barra.

Observaciones:

Solo válida si el panel ha sido creado con `panel_scroll`. Útil si queremos tener en cuenta el tamaño de las barras de scroll al establecer los márgenes del Layout.

panel_scroll_height

Obtiene el alto de la barra de scroll.

```
real32_t
panel_scroll_height(const Panel *panel);
```

panel El panel.

Retorna:

El alto de la barra.

Observaciones:

Ver `panel_scroll_width`.

window_create

Crea una nueva ventana.

```
Window*
window_create(const uint32_t flags);
```

flags Combinación de valores `window_flag_t`.

Retorna:

La ventana recién creada.

window_destroy

Destruye la ventana y todo su contenido.

```
void
window_destroy(Window **window);
```

window La ventana. Será puesto a `NULL` tras la destrucción.

Observaciones:

Se destruirán recursivamente paneles, layouts y componentes.

window_panel

Asocia el panel principal a una ventana.

```
void
window_panel(Window *window,
             Panel *panel);
```

window La ventana.

panel Panel principal, que integra todo el contenido de la ventana (vistas, controles, etc).

Observaciones:

El tamaño de la ventana se ajustará en función del *"Dimensionado naturalDimensionado natural"* (Página 336) del panel principal.

window_OnClose

Establece un manejador para el evento de cierre de la ventana.

```
void
window_OnClose(Window *window,
               Listener *listener);
```

window La ventana.

listener Función *callback* que se llamará antes de cerrar una ventana.

Observaciones:

Ver *"Cierre de la ventanaCierre de la ventana"* (Página 352).

window_OnMoved

Establece un manejador para el desplazamiento de la ventana por el escritorio.

```
void
window_OnMoved(Window *window,
               Listener *listener);
```

window La ventana.

listener Función *callback* que se llamará a medida que se arrastre la barra de título y se mueva la ventana por el escritorio.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

window_OnResize

Establece un manejador para el redimensionado de la ventana.

```
void
window_OnResize(Window *window,
                Listener *listener);
```

window La ventana.

listener Función *callback* que se llamará a medida que se arrastren los bordes externos de la ventana para cambiar su tamaño.

Observaciones:

El redimensionado y rehubicación de elementos se realiza de forma automática en función del `Layout` principal, por lo que no suele ser necesario que la aplicación responda a este evento. Ver *"Eventos GUIEventos GUI"* (Página 306).

window_title

Establece el texto que mostrará la ventana en la barra de título.

```
void
window_title(Window *window,
             const char_t *text);
```

window La ventana.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

window_show

Muestra la ventana. Por defecto las ventanas se crean ocultas. Hay que mostrarlas explícitamente.

```
void
window_show(Window *window);
```

window La ventana.

window_hide

Oculta la ventana.

```
void
window_hide(Window *window);
```

window La ventana.

window_modal

Lanza una ventana en modo **modal**.

```
uint32_t
window_modal(Window *window,
             Window *parent);
```

window La ventana.

parent La ventana bloqueada.

Retorna:

Valor retornado por `window_stop_modal`.

Observaciones:

`parent` dejará de recibir eventos hasta que no se llame a `window_stop_modal`.

window_stop_modal

Termina el ciclo **modal** de una ventana.

```
void
window_stop_modal(Window *window,
                  const uint32_t return_value);
```

window	La ventana lanzada previamente con `window_modal`.
return_value	Valor que deberá devolver `window_modal`.

window_hotkey

Establece una acción asociada a la pulsación de una tecla.

```
void
window_hotkey(Window *window,
              const vkey_t key,
              const uint32_t modifiers);
```

window	La ventana.
key	La tecla.
modifiers	Modificadores. `0` o combinación de `mkey_t`.

Observaciones:

Ver *"Teclas de acceso directoTeclas de acceso directo"* (Página 355).

window_next_tabstop

Mueve el foco del teclado al siguiente control en la *tab-list*. Produce el mismo efecto que pulsar `[TAB]`.

```
void
window_next_tabstop(Window *window);
```

window	La ventana.

Observaciones:

Equivalente a `layout_next_tabstop`. Ver *"TabstopsTabstops"* (Página 342).

window_previous_tabstop

Mueve el foco del teclado al anterior control en la *tab-list*. Produce el mismo efecto que pulsar `[SHIFT]+[TAB]`.

```
void
window_previous_tabstop(Window *window);
```

window La ventana.

Observaciones:

Equivalente a `layout_previous_tabstop`. Ver *"TabstopsTabstops"* (Página 342).

window_update

Recalcula la posición y tamaño de los controles tras modificar cualquier `Layout`.

```
void
window_update(Window *window);
```

window La ventana.

window_origin

Mueve la ventana a unas coordenadas concretas del escritorio.

```
void
window_origin(Window *window,
              const V2Df origin);
```

window La ventana.

origin Posición `(x,y)` de la esquina superior-izquierda de la ventana.

window_size

Establece el tamaño del área cliente de la ventana.

```
void
window_size(Window *window,
            const S2Df size);
```

window La ventana.

size Tamaño del panel principal.

Observaciones:

El tamaño final dependerá de la configuración del marco de la ventana y del tema de escritorio. Esta medida solo hace referencia al área interior.

window_get_origin

Obtiene la posición de la ventana.

```
V2Df
window_get_origin(const Window *window);
```

window La ventana.

Retorna:

Posición `(x,y)` de la esquina superior-izquierda de la ventana.

window_get_size

Obtiene las dimensiones totales de la ventana.

```
S2Df
window_get_size(const Window *window);
```

window La ventana.

Retorna:

Tamaño de la ventana.

Observaciones:

Se tiene en cuenta el marco y barra de título.

window_get_client_size

Obtiene las dimensiones del área cliente de la ventana.

```
S2Df
window_get_client_size(const Window *window);
```

window La ventana.

Retorna:

Tamaño del panel principal.

window_defbutton

Establece el botón por defecto de la ventana. Será activado cuando se pulse `[Intro]`.

```
void
window_defbutton(Window *window,
                 Button *button);
```

window La ventana.

button El botón.

window_cursor

Cambia el cursor del ratón.

```
void
window_cursor(Window *window,
              const gui_cursor_t cursor,
              const Image *image,
              const real32_t hot_x,
              const real32_t hot_y);
```

window La ventana.

cursor Identificador del nuevo cursor.

image Imagen personalizada. Solo válido en `ekGUI_CURSOR_USER`.

hot_x Coordenada x del punto de clic. Solo válido en `ekGUI_CURSOR_USER`.

hot_y Coordenada y del punto de clic. Solo válido en `ekGUI_CURSOR_USER`.

Observaciones:

`hot_x, hot_y` indican el punto "sensible" dentro de la imagen, que indicará la posición exacta del ratón.

menu_create

Crea un nuevo menú.

```
Menu*
menu_create(void);
```

Retorna:

El menú recién creado.

menu_destroy

Destruye un menú y toda su jerarquía.

```
void
menu_destroy(Menu **menu);
```

menu El menú. Será puesto a `NULL` tras la destrucción.

menu_launch

Lanza un menú como secundario o *PopUp*.

```
void
menu_launch(Menu *menu,
            const V2Df position);
```

menu El menú.

position Coordenadas de la esquina superior izquierda.

menu_hide

Oculta un menú como secundario o *PopUp*.

```
void
menu_hide(Menu *menu);
```

menu El menú.

menu_item

Añade un item al menú.

```
void
menu_item(Menu *menu,
          MenuItem *item);
```

menu El menú.

item El item a añadir.

menu_off_items

Establece el estado `ekGUI_OFF` para todos los items del menú.

```
void
menu_off_items(Menu *menu);
```

menu El menú.

menu_get_item

Obtiene un item del menú.

```
MenuItem*
menu_get_item(Menu *menu,
              const uint32_t index);
```

menu El menú.

index El índice del item.

Retorna:

El item.

menu_size

Obtiene el número de items.

```
uint32_t
menu_size(const Menu *menu);
```

menu El menú.

Retorna:

Número de items.

menuitem_create

Crea un nuevo item para un menú.

```
MenuItem*
menuitem_create(void);
```

Retorna:

El item recién creado.

menuitem_separator

Crea un nuevo separador para un menú.

```
MenuItem*
menuitem_separator(void);
```

Retorna:

El item recién creado.

menuitem_OnClick

Establece un manejador para la selección del item.

```
void
menuitem_OnClick(MenuItem *item,
                 Listener *listener);
```

item El item.

listener Función *callback* que se llamará tras hacer clic.

Observaciones:

Ver *"Eventos GUIEventos GUI"* (Página 306).

menuitem_enabled

Activa o desactiva un item del menú.

```
void
menuitem_enabled(MenuItem *item,
                 const bool_t enabled);
```

item El item.

enabled Habilitado o no.

menuitem_visible

Muestra u oculta un item del menú.

```
void
menuitem_visible(MenuItem *item,
                 const bool_t enabled);
```

item El item.

enabled Habilitado o no.

menuitem_text

Establece el texto del item.

```
void
menuitem_text(MenuItem *item,
              const char_t *text);
```

item El item.

text Cadena C UTF8 terminada en carácter nulo `'\0'`.

menuitem_image

Establece el icono que mostrará el item.

```
void
menuitem_image(MenuItem *item,
               const Image *image);
```

item El item.

image Imagen.

menuitem_key

Establece un atajo de teclado para seleccionar el item del menú.

```
void
menuitem_key(MenuItem *item,
             const vkey_t key,
             const uint32_t modifiers);
```

item El item.

key El código de teclado.

modifiers Modificadores.

menuitem_submenu

Asigna un submenú desplegable al seleccionar el item.

```
void
menuitem_submenu(MenuItem *item,
                 Menu **submenu);
```

item El item.

submenu El submenú.

menuitem_state

Establece el estado del item, que se reflejará con una marca junto al texto.

```
void
menuitem_state(MenuItem *item,
               const gui_state_t state);
```

item El item.

state Estado.

comwin_open_file

Lanza el diálogo de abrir archivo.

```
const char_t*
comwin_open_file(Window *parent,
                 const char_t **ftypes,
                 const uint32_t size,
                 const char_t *start_dir);
```

parent	Ventana padre.
ftypes	Tipos de archivo para el filtro.
size	Número de tipos de archivo.
start_dir	Directorio de inicio del diálogo. Pueder ser `NULL`.

Retorna:

El nombre del archivo seleccionado o `NULL` si el usuario ha abortado el diálogo.

Observaciones:

Se lanzará de forma **modal**. `parent` quedará bloqueada hasta que se acepte el diálogo.

comwin_save_file

Lanza el diálogo de guardar archivo.

```
const char_t*
comwin_save_file(Window *parent,
                 const char_t **ftypes,
                 const uint32_t size,
                 const char_t *start_dir);
```

parent	Ventana padre.
ftypes	Tipos de archivo para el filtro.
size	Número de tipos de archivo.
start_dir	Directorio de inicio del diálogo. Pueder ser `NULL`.

Retorna:

El nombre del archivo seleccionado o `NULL` si el usuario ha abortado el diálogo.

Observaciones:

Se lanzará de forma **modal**. `parent` quedará bloqueada hasta que se acepte el diálogo.

comwin_color

Lanza el diálogo de selección de colores.

```
void
comwin_color(Window *parent,
             const real32_t x,
             const real32_t y,
             const align_t halign,
             const align_t valign,
             const color_t current,
             color_t *colors,
             const uint32_t n,
             Listener *OnChange);
```

```
static void i_OnColorChange(App *app, Event *e)
{
    color_t *color = event_params(e, color_t);
    // Do something
    ...
}

comwin_color(window, "Select color", 100, 50, ekRIGHT, ekTOP, kCOLOR_BLUE, NULL
    ↪ , 0, listener(app, i_OnColorChange, App));
```

parent	Ventana padre.
x	Posición x inicial.
y	Posición y inicial.
halign	Alineación horizontal respecto a `x`.
valign	Alineación vertical respecto a `y`.
current	Color actual que mostrará el panel.
colors	Colores personalizados que mostrará el panel y que pueden también ser editados. Puede ser `NULL` sólo si `n=0`.
n	Número de colores personalizados.
OnChange	Función *callback* que se llamará tras cada cambio de color.

Observaciones:

En sistemas Windows y Linux el diálogo se lanzará de forma modal y deberá ser aceptado para que se produzca una notificación del cambio de color a través de `OnChange`. En macOS se lanzarán notificaciones continuamente a medida que se manipula el diálogo.

Capítulo 41

Librería OSApp

41.1. Funciones

FPtr_app_create

Prototipo de constructor de una aplicación.

```
type*
(*FPtr_app_create)(void);
```

Retorna:

Objeto aplicación.

FPtr_app_update

Prototipo de función de actualización de una aplicación síncrona.

```
void
(*FPtr_app_update)(type *app,
                   const real64_t prtime,
                   const real64_t ctime);
```

app	Objeto aplicación.
prtime	Tiempo de la actualización anterior.
ctime	Tiempo actual.

FPtr_task_main

Prototipo de función de inicio de una tarea.

```
uint32_t
(*FPtr_task_main)(type *data);
```

data Datos iniciales de la tarea.

Retorna:

Valor de retorno de la tarea.

FPtr_task_update

Prototipo de función de actualización de una tarea.

```
void
(*FPtr_task_update)(type *data);
```

data Datos de la tarea.

FPtr_task_end

Prototipo de función de finalización de una tarea.

```
void
(*FPtr_task_end)(type *data,
                 const uint32_t rvalue);
```

data Datos de la tarea.

rvalue Valor de retorno de la tarea.

osmain

Inicia una aplicación de escritorio.

```
void
osmain(FPtr_app_create func_create,
       FPtr_destroy func_destroy,
       const char_t *options,
       type);
```

func_create Constructor del objeto aplicación.

func_destroy Destructor del objeto aplicación.

options Cadena de opciones.

type Tipo de objeto aplicación.

Observaciones:

En *"¡Hola Mundo!"* (Página 23) tienes un ejemplo sencillo de aplicación de escritorio.

osmain_sync

Inicia una aplicación de escritorio síncrona.

```
void
osmain_sync(const real64_t lframe,
            FPtr_app_create func_create,
            FPtr_destroy func_destroy,
            FPtr_app_update func_update,
            const char_t *options,
            type);
```

lframe	Tiempo en segundos del intervalo de actualización (0.04 = 25 fps).
func_create	Constructor del objeto aplicación.
func_destroy	Destructor del objeto aplicación.
func_update	Función que se llamará en cada intervalo de actualización.
options	Cadena de opciones.
type	Tipo de objeto aplicación.

Observaciones:

Ver *"Aplicaciones síncronasAplicaciones síncronas"* (Página 377).

osapp_finish

Finaliza una aplicación de escritorio, destruyendo el ciclo de mensajes y el objeto aplicación.

```
void
osapp_finish(void);
```

osapp_task

Lanza una tarea en paralelo, evitando el bloqueo del hilo que controla la interfaz de usuario.

```
void
osapp_task(type *data,
           const real32_t updtime,
           FPtr_task_main func_main,
           FPtr_task_update func_update,
           FPtr_task_end func_end,
           type);
```

data	Datos iniciales de la tarea.
updtime	Tiempo del intervalo de actualización, en caso que lo requiera.
func_main	Función de inicio de la tarea.
func_update	Función de actualización de la tarea.
func_end	Función que se llamará al acabar la tarea.
type	Tipo de los datos iniciales de la tarea.

Observaciones:

Ver *"Tareas multi-hiloTareas multi-hilo"* (Página 378).

osapp_menubar

Establece la barra de menú general de la aplicación.

```
void
osapp_menubar(Menu *menu,
              Window *window);
```

menu	El menú.
window	La ventana que albergará el menú.

Observaciones:

En macOS el menú de aplicación no se vincula con ninguna ventana.

osapp_open_url

Abre una dirección de Internet utilizando el navegador por defecto del sistema operativo.

```
void
osapp_open_url(const char_t *url);
```

url	La dirección URL.

Capítulo 42

Librería INet

42.1. Tipos y Constantes

enum ierror_t

Códigos de error de las conexiones de red.

ekINONET	No hay conexión a Internet en el dispositivo.
ekINOHOST	No se puede conectar con el servidor remoto.
ekITIMEOUT	Se ha excedido el tiempo máximo de espera por la conexión.
ekISTREAM	Error en el canal E/S al leer o escribir.
ekISERVER	Error en el formato de respuesta del servidor.
ekINOIMPL	Funcionalidad no implementada.
ekIUNDEF	Error indeterminado.
ekIOK	No hay error.

struct Http

Gestiona una conexión HTTP iniciada desde el proceso cliente.

```
struct Http;
```

struct Url

Permite el acceso a los campos individuales de una URL (dirección Web) *"URL"* (Página 391).

```
struct Url;
```

struct JsonOpts

Opciones al procesar un script JSON.

```
struct JsonOpts;
```

42.2. Funciones

http_create

Crea una sesión HTTP.

```
Http*
http_create(const char_t *host,
            const uint16_t port);
```

host Nombre del servidor.

port Puerto de conexión. Si pasamos `UINT16_MAX` se utilizará 80 (por defecto para HTTP).

Retorna:

Sesión HTTP.

http_secure

Crea una sesión HTTPS.

```
Http*
http_secure(const char_t *host,
            const uint16_t port);
```

host Nombre del servidor.

port Puerto de conexión. Si pasamos `UINT16_MAX` se utilizará 413 (por defecto para HTTP).

Retorna:

Sesión HTTPS.

http_destroy

Destruye una sesión HTTP.

```
void
http_destroy(Http **http);
```

http La sesión HTTP. Será puesto a NULL tras la destrucción.

http_clear_headers

Elimina las cabeceras HTTP previamente asignadas.

```
void
http_clear_headers(Http *http);
```

http La sesión HTTP.

http_add_header

Añade una cabecera a la petición HTTP.

```
void
http_add_header(Http *http,
                const char_t *name,
                const char_t *value);
```

http La sesión HTTP.

name El nombre de la cabecera.

value El valor de la cabecera.

http_get

Realiza una petición tipo GET.

```
bool_t
http_get(Http *http,
         const char_t *path,
         const byte_t *data,
         const uint32_t size,
         ierror_t *error);
```

http La sesión HTTP.

path Dirección del recurso.

data Datos para añadir en el cuerpo de la petición. Puede ser NULL.

size Tamaño en bytes del bloque de datos.

error Código de error si la función falla. Puede ser NULL.

Retorna:

`TRUE` si la petición se ha llevado a cabo correctamente. Si `FALSE`, en `error` tendremos la causa.

Observaciones:

La petición es síncrona, es decir, el programa quedará detenido hasta que el servidor responda. Si queremos un modelo asíncrono deberemos crear un hilo paralelo que gestione la petición. Las redirecciones HTTP se resuelven automáticamente.

http_post

Realiza una petición tipo POST.

```
bool_t
http_post(Http *http,
          const char_t *path,
          const byte_t *data,
          const uint32_t size,
          ierror_t *error);
```

http	La sesión HTTP.
path	Dirección del recurso.
data	Datos para añadir en el cuerpo de la petición. Puede ser `NULL`.
size	Tamaño en bytes del bloque de datos.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si la petición se ha llevado a cabo correctamente. Si `FALSE`, en `error` tendremos la causa.

Observaciones:

Ver `http_get`.

http_response_status

Devuelve el código de respuesta de una petición HTTP.

```
uint32_t
http_response_status(const Http *http);
```

http	La sesión HTTP.

Retorna:

El código de respuesta del servidor.

http_response_protocol

Devuelve el protocolo utilizado por el servidor HTTP.

```
const char_t*
http_response_protocol(const Http *http);
```

http La sesión HTTP.

Retorna:

El protocolo del servidor.

http_response_message

Devuelve el mensaje de respuesta del servidor HTTP.

```
const char_t*
http_response_message(const Http *http);
```

http La sesión HTTP.

Retorna:

El mensaje de respuesta del servidor.

http_response_size

Devuelve el número de cabeceras de respuesta de una petición HTTP.

```
uint32_t
http_response_size(const Http *http);
```

http La sesión HTTP.

Retorna:

El número de cabeceras.

http_response_name

Devuelve el nombre de la cabecera de respuesta de una petición HTTP.

```
const char_t*
http_response_name(const Http *http,
                   const uint32_t index);
```

http La sesión HTTP.

index El índice de la cabecera (0, size-1).

Retorna:

El nombre de la cabecera.

http_response_value

Devuelve el valor de la cabecera de respuesta de una petición HTTP.

```
const char_t*
http_response_value(const Http *http,
                    const uint32_t index);
```

http La sesión HTTP.

index El índice de la cabecera (0, size-1).

Retorna:

El valor de la cabecera.

http_response_header

Devuelve el valor de una cabecera de respuesta de una petición HTTP.

```
const char_t*
http_response_header(const Http *http,
                     const char_t *name);
```

http La sesión HTTP.

name El nombre de la cabecera deseada.

Retorna:

El valor de la cabecera. Si la cabecera no existe, devolverá una cadena vacía "".

http_response_body

Devuelve el cuerpo de la respuesta de una petición HTTP.

```
bool_t
http_response_body(const Http *http,
                   Stream *body,
                   ierror_t *error);
```

http	La sesión HTTP.
body	Stream de escritura donde se almacenará el contenido de la respuesta.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

`TRUE` si se ha podido leer correctamente. Si `FALSE`, en `error` tendremos la causa.

http_dget

Realiza una petición directa de un recurso Web.

```
Stream*
http_dget(const char_t *url,
          uint32_t *result,
          ierror_t *error);
```

```
Stream *json = http_dget("http://serv.nappgui.com:80/dproducts.php", NULL, NULL
    ↪ );
if (json)
{
    ...
    stm_close(&json);
}
```

url	URL del recurso.
result	Código de respuesta del servido. Puede ser `NULL`.
error	Código de error si la función falla. Puede ser `NULL`.

Retorna:

Stream con el resultado de la petición.

Observaciones:

Utiliza esta función para el acceso directo a un recurso aislado. Si necesitas realizar varias peticiones o configurar las cabeceras, utiliza `http_create` o `http_secure`.

http_exists

Chequea si un recurso Web está disponible/accesible.

```
bool_t
http_exists(const char_t *url);
```

url URL del recurso.

Retorna:

`TRUE` si el recurso (página Web, archivo, etc) está accesible.

Observaciones:

No se resuelven las redirecciones HTTP. Retornará `FALSE` si la URL tal cual no es válida.

json_read

Procesa un script JSON. Transformará texto JSON en un tipo u objeto en C.

```
type*
json_read(Stream *stm,
          const JsonOpts *opts,
          type);
```

stm Entrada de datos en formato JSON.

opts Opciones.

type Tipo de datos.

Retorna:

Objeto resultado.

Observaciones:

Ver *"Análisis de JSON y conversión a datos en CAnálisis de JSON y conversión a datos en C"* (Página 385).

json_write

Escribe datos en C a formato JSON.

```
void
json_write(Stream *stm,
           type *data,
           const JsonOpts *opts,
```

```
            type);
```

stm Salida de datos en formato JSON.

data Objeto.

opts Opciones.

type Tipo de datos.

Observaciones:

Ver *"Convertir desde C a JSONConvertir desde C a JSON"* (Página 389).

json_destroy

Destruye un objeto JSON, previamente creado con `json_read`.

```
void
json_destroy(type **data,
             type);
```

data Objeto.

type Tipo de datos.

json_destopt

Destruye un objeto JSON, previamente creado con `json_read`, si este no es `NULL`.

```
void
json_destopt(type **data,
             type);
```

data Objeto.

type Tipo de datos.

url_parse

Crea un objeto URL a partir de una cadena de texto.

```
Url*
url_parse(const char_t *url);
```

url Cadena de texto C UTF8 terminada en carácter nulo `'\0'`.

Retorna:

Objeto URL resultado tras analizar la cadena.

url_destroy

Destruye el objeto URL.

```
void
url_destroy(Url **url);
```

url Objeto URL. Será puesto a `NULL` tras la destrucción.

url_scheme

Obtiene el esquema (protocolo) de la URL.

```
const char_t*
url_scheme(const Url *url);
```

url Objeto URL.

Retorna:

Protocolo (http, https, ftp, etc).

url_user

Obtiene el usuario.

```
const char_t*
url_user(const Url *url);
```

url Objeto URL.

Retorna:

Usuario ó `""` si no se especificó.

url_pass

Obtiene la contraseña.

```
const char_t*
url_pass(const Url *url);
```

url Objeto URL.

Retorna:

Contraseña ó `""` si no se especificó.

url_host

Obtiene el nombre del servidor.

```
const char_t*
url_host(const Url *url);
```

url Objeto URL.

Retorna:

Host (Pe. `www.google.com`).

url_path

Obtiene el camino (directorios + nombre) del recurso o archivo solicitado.

```
const char_t*
url_path(const Url *url);
```

url Objeto URL.

Retorna:

Pathname (Pe. `/dir1/dir2/file.html`).

url_params

Obtiene los parámetros (a partir de `';'`) de la URL.

```
const char_t*
url_params(const Url *url);
```

url Objeto URL.

Retorna:

Parámetros ó `""` si no se especificaron.

url_query

Obtiene los parámetros (a partir de `'?'`) de la URL.

```
const char_t*
url_query(const Url *url);
```

url Objeto URL.

Retorna:

Parámetros ó "" si no se especificaron.

url_fragment

Obtiene el fragmento (posición o ancla del documento) de la URL.

```
const char_t*
url_fragment(const Url *url);
```

url Objeto URL.

Retorna:

Fragmento ó "" si no se especificó.

url_resource

Obtiene la dirección completa de un recurso dentro del servidor.

```
String*
url_resource(const Url *url);
```

url Objeto URL.

Retorna:

Recurso. `path + ";"+ params + "?"+ query + "#"+ fragment`.

url_port

Obtiene el puerto de acceso al servidor.

```
uint16_t
url_port(const Url *url);
```

url Objeto URL.

Retorna:

Puerto. `UINT16_MAX` si no se especificó.

b64_encoded_size

Obtiene el número de bytes necesarios para codificar un bloque de memoria en formato **base64**.

```
uint32_t
b64_encoded_size(const uint32_t data_size);
```

data_size El tamaño del bloque original.

Retorna:

El tamaño base64.

b64_decoded_size

Obtiene el número de bytes necesarios para decodificar un bloque de memoria en formato **base64**.

```
uint32_t
b64_decoded_size(const uint32_t data_size);
```

data_size El tamaño del bloque codificado en base64.

Retorna:

El tamaño en bytes.

b64_encode

Codifica un bloque de memoria en **base64**.

```
uint32_t
b64_encode(const byte_t *data,
           const uint32_t size,
           char_t *base64);
```

data El bloque de datos.

size El tamaño del bloque.

base64 El búfer donde almacenar el resultado.

Retorna:

El tamaño en bytes.

Observaciones:

El búfer `base64` debe tener, al menos, el tamaño devuelto por `b64_encoded_size`.

b64_decode

De-codifica un bloque **base64**.

```
uint32_t
b64_decode(const char_t *base64,
           const uint32_t size,
           byte_t *data);
```

base64	El bloque base64.
size	El tamaño del bloque.
data	El búfer donde almacenar el resultado.

Retorna:

El tamaño en bytes.

Observaciones:

El búfer `data` debe tener, al menos, el tamaño devuelto por `b64_decoded_size`.

Índice alfabético