Cross-Platform Development in C

Mastering NAPPGUI

Francisco García Collado

Cross-Platform C language development

How to create high-performance applications for Windows, macOS, and Linux systems.

Version 1.4.0.4577 [1]

frang@nappgui.com
www.nappgui.com

1 November 2023

[1] This book has been edited in LaTeXgenerated automatically by **ndoc**.

Contents

Part 1

Users guide

Chapter 1

Quick start

"...the number of UNIX installations has grown to 10, with more expected..."

Dennis Ritchie and Ken Thompson - June 1972

NAppGUI is an SDK to develop software projects, that work on any desktop platform (Windows, macOS or Linux), using the C programming language (Figure 1.1). C++ is allowed, but not indispensable. We can write a complete program using only ANSI-C.

1.1. Quick start in Windows

Before starting you need to have these tools installed (Figure 1.2):

- Visual Studio[1] to compile under Windows. Microsoft offers the free *Community* version.
- CMake[2]. Cross-platform tool to create compilation projects automatically, from source code. Be careful to select ***Add CMake to the system PATH for all users*** during installation (Figure 1.3).

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

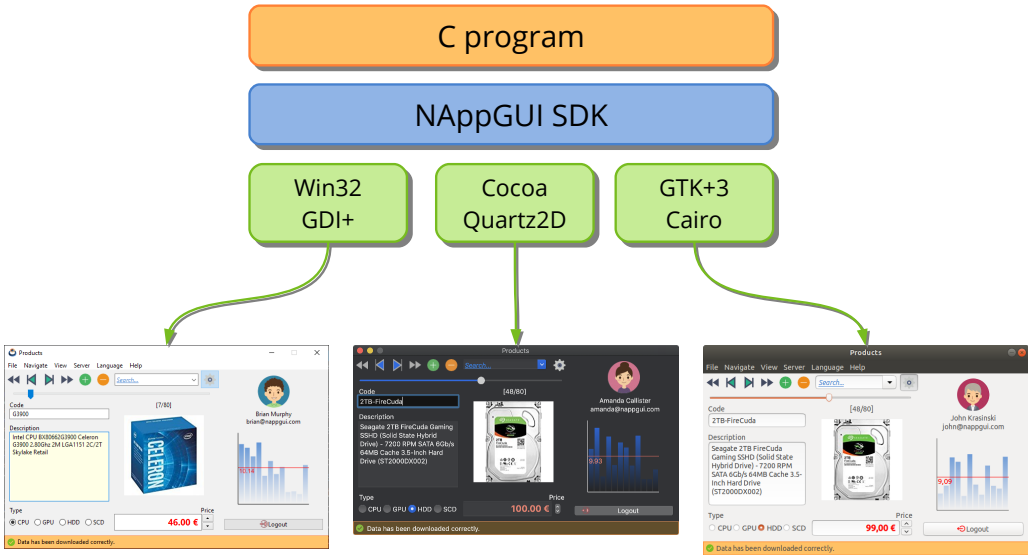


Figure 1.1: NAppGUI allows the easy port of applications written in ANSI C.

- Git[3]. For download the project from GitHub.

Figure 1.2: Basic tools in Windows.

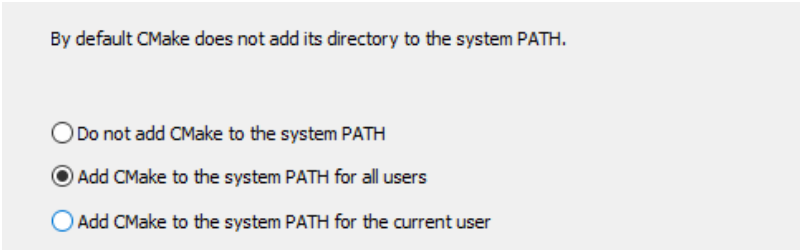


Figure 1.3: Access to CMake from the command line.

From a console on 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/

Once compiled, you will be able to run the existing example applications in the **demo** and **howto** directories (Figure 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
...
```

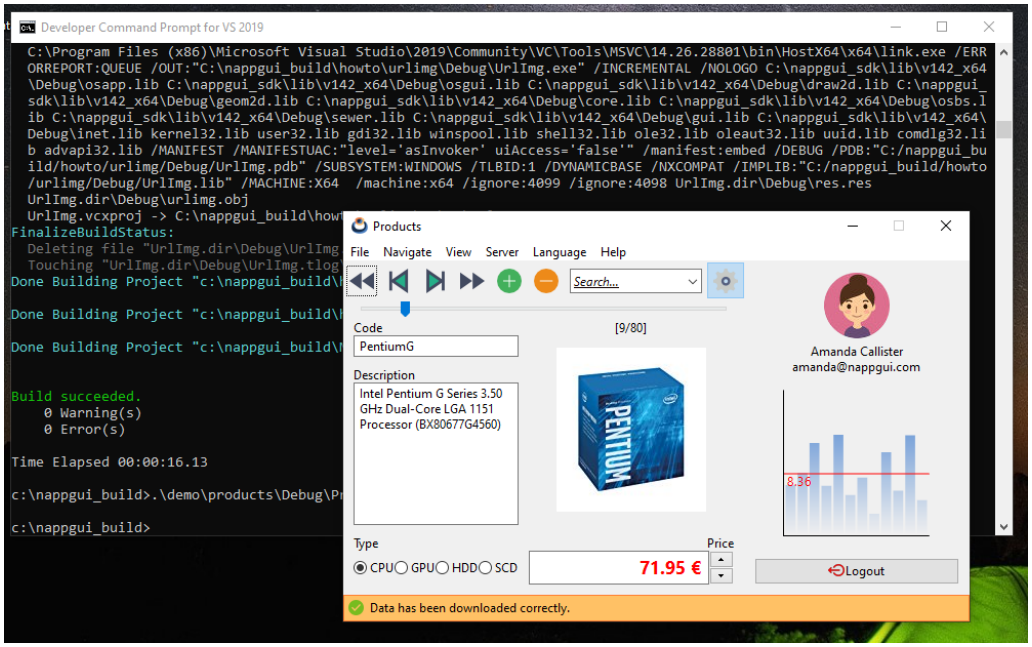


Figure 1.4: Running the **Products** sample program after compilation.

1.2. Quick start on macOS

Before starting, make sure you have installed and configured Xcode[4], an essential environment for development under macOS. You will also need to download and install CMake from www.cmake.org[5] (Figure 1.5).

> *By default, CMake does not configure command line access on macOS. You can create symbolic links with* ***sudo "/Applications/CMake.app/Contents/bin/cmake-gui" –install****.*

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

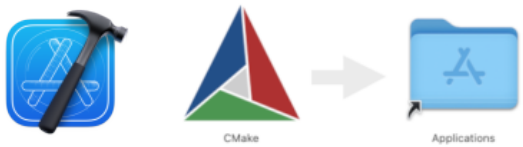


Figure 1.5: Xcode and CMake on macOS.

Open a terminal in 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
```

Once compiled, you can run the existing sample applications in the directories **demo** and **howto** (Figure 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
...
```

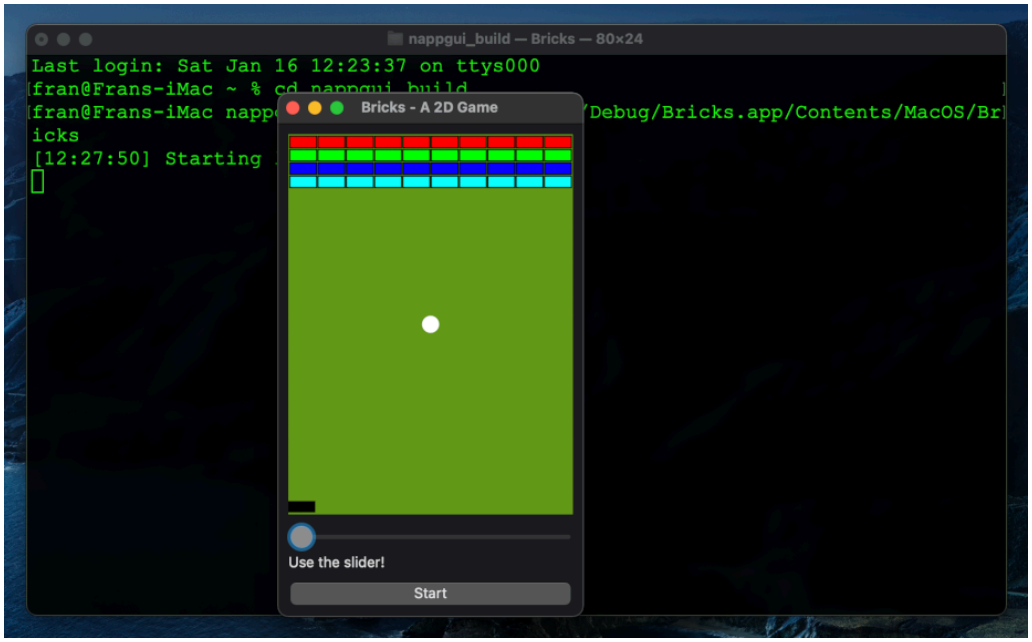


Figure 1.6: Running the **Bricks** sample program after compilation.

1.3. Quick start on Linux

Before starting, make sure you have the necessary compilers, tools and libraries installed:

```
// 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 requires at least **gcc 4.6**, **gtk3** (for desktop applications), **OpenGL** (if your application uses 3D graphics) and **Curl** (for Web protocols). All this is included as of Ubuntu 12.04 LTS or similar distributions.*

Open a 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
```

Once compiled, you will be able to launch the existing example applications in the **demo** and **howto** directories (Figure 1.7).

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

1.4. MIT License

NAppGUI is distributed under the MIT license, which essentially means that you have complete freedom to use this software freely and for free, both in commercial and free projects. The only restriction is that you must include a copy of this License[6] in every substantial part of the software you distribute.

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

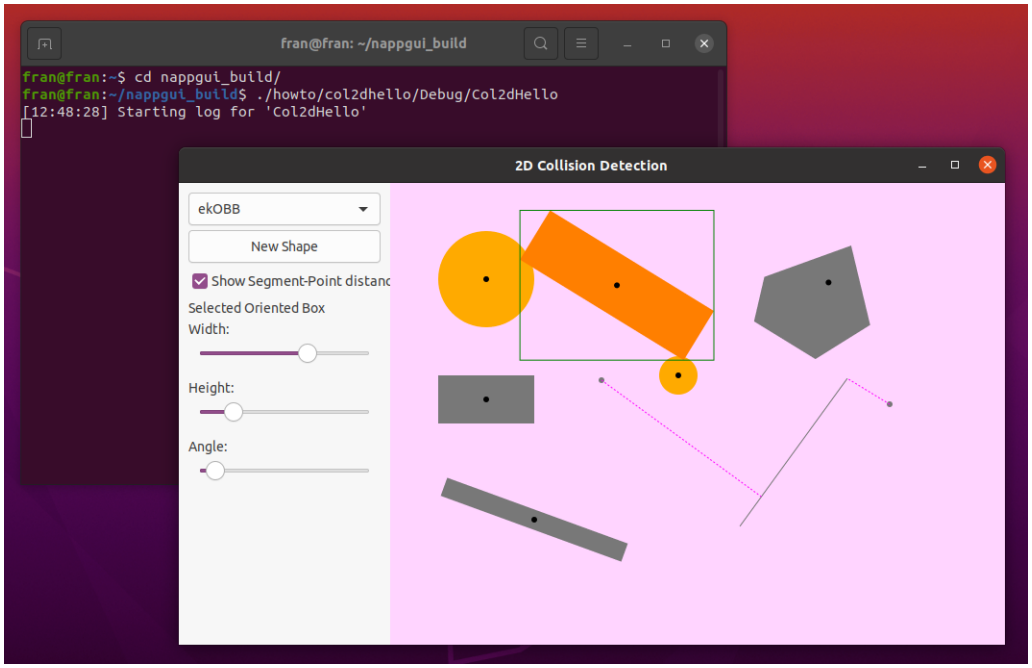


Figure 1.7: Running the **Col2dHello** sample program after compilation.

1.5. Previous knowledge

This book is not intended for beginners. Although the NAppGUI project is aimed at simplifying the construction of cross-platform applications, it requires certain prior knowledge on the part of the user. You will need, at least, to be fluent in C or C++ since at no time we will stop to explain basic programming concepts. If you come from Java or C#, you should review **pointers**. You will also need some skill with Visual Studio and Xcode development environments, and Unix tools such as gcc, make or the command interpreter.

On the other hand, if you are an advanced user, you will find a simple system to create very fast and small C applications that will compile without changes in all desktop environments. You will also have at your disposal a set of precompiled C libraries to create user interfaces or command line applications, without the need to mess up your projects with the cumbersome class templates that **stl** or **boost** provide.

1.6. And now what?

- In *"Welcome to NAppGUI"* (page 13) we continue with the tutorial.
- In *"Hello World!"* (page 23) we see the minimal code of a desktop application.

- In *"Generate NAppGUI binaries"* (page 65) we see how to compile the static or dynamic version.
- In *"Compilers and IDEs"* (page 73) you will have information about porting.
- In *"Create new application"* (page 99) you will start creating your own applications.
- In *"NAppGUI API"* (page 145) you have the documentation of the libraries and functions.
- In *"Products"* (page 433) you have the source code of a medium-sized application.

Chapter 2

Welcome to NAppGUI

While others were content to write programs that just solved problems, early hackers were obsessed with writing programs that solved problems well. A new program that achieved the same result as an existing one but used fewer punch cards was considered better, even if it did the same thing. The fundamental difference was how the program achieved its result. - ***elegance****.*

Jon Erickson - Hacking: The Art of Exploitation

NAppGUI is an SDK for creating cross-platform native applications in C. By **native software** we understand that which is compiled/assembled using the specific instructions of the CPU (it is not interpreted or used bytecode) and by **cross-platform** the ability to build versions for Windows, macOS, and Linux using the same (Figure 2.1) source code base. Since its first functions written in August 2010, the main objective of NAppGUI has been to simplify as much as possible the arduous task of creating applications with a graphical interface in C. Although different solutions already exist, we have opted for simplicity by creating a light abstraction layer that encapsulates native technologies, unifies them under the same API and adds some logic for task management and automation. Being somewhat more specific, the philosophy on which the project is based and some of its characteristics are:

- Rapid prototyping, evolution and maintenance in **real** applications, apart from the simple examples we find in the literature and the Internet.

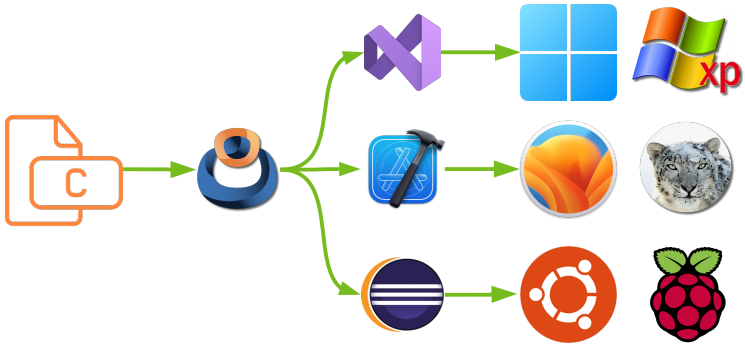


Figure 2.1: Native cross-platform development with NAppGUI.

- The user interface is described using ANSI-C functions, completely eliminating visual design. This fact facilitates the creation of dynamic interfaces, guarantees portability and enables access to the API from any programming language.
- Windows are automatically laid out and sized, without the programmer having to explicitly specify the coordinates and size of the controls.
- It is possible to have a complete application in a single `.c` file, by removing the usual resource files (`*.rc`, `*.xvid` , etc) and their associated controllers. The programmer has complete freedom when defining his own file structure.
- Automatic synchronization of internal data structures with the interface or with I/O channels. *"Data binding"* (page 225).
- Unified management of resources which facilitates internationalization. *"Resources"* (page 129).
- Translations between languages at runtime without the need to restart the application. *"Runtime translationRuntime translation"* (page 134).
- The compiled version of NAppGUI occupies less than 1Mb, and is distributed in several static libraries that generate very small executables. This is a great advantage over other solutions that require the distribution of heavy *.DLLs*, sometimes larger than the application itself.
- Native Appearance: The applications will be integrated into each system respecting their original aesthetic (Figure 2.2).
- *Backends.* The NAppGUI core provides structures and objects for creating highly efficient command-line applications on Windows or Linux servers.

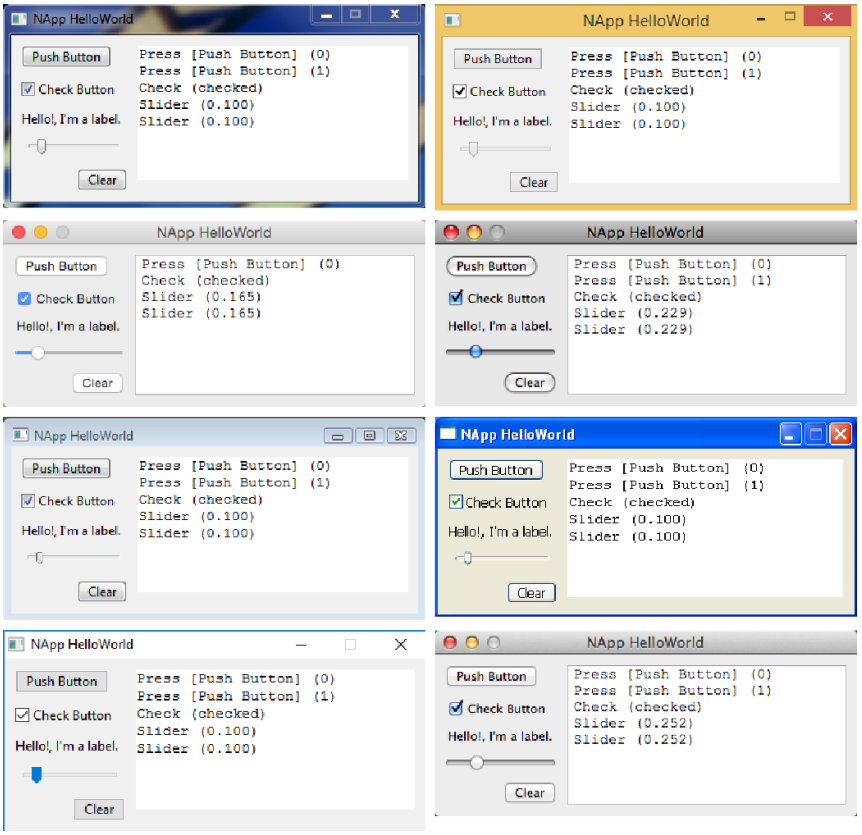


Figure 2.2: Native appearance of the *Hello, World!* demo.

2.1. Original APIs

Microsoft, Apple and GNU/Linux propose different APIs to interact with their systems. This means that the same application must be rewritten to work correctly on each platform. NAppGUI provides a unified set of functions for creating graphical user interfaces and allowing direct access to machine resources (memory, disk, network, etc.) (Figure 2.3). Each implementation takes into account the particular conditions of the target platform and uses the appropriate native commands to perform the task in the most optimal way possible.

2.2. C-based

Despite the fact that today we have a large number of programming languages, the C language is still the most powerful and portable in the world. The core of Windows, macOS, Linux, Android, iOS, and other major programs are largely written in C. In the world of apps, its use has waned a bit in favor of more *glamour*. Perhaps this is one of the

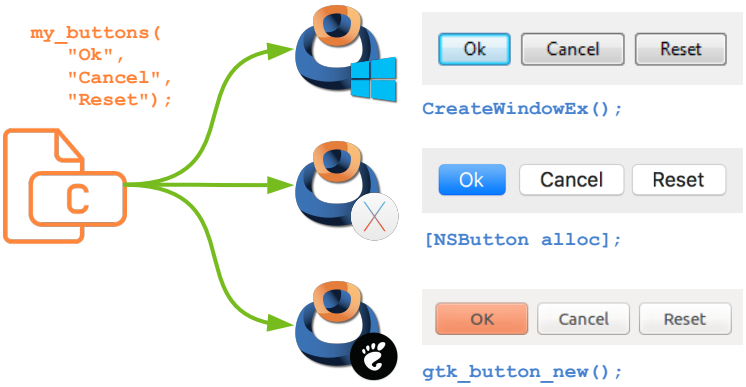


Figure 2.3: Calls to the native APIs, from the source code itself.

reasons why Wirth's law[1] is more and more true every day.

> *"Software slows down faster than hardware speeds up."*

NAppGUI is written, almost entirely, in C language with small parts in C++ and Objective-C. This language is widely supported and cross-platform compatible. In its development we have dispensed with minority, proprietary or brand-linked languages such as: C#, Swift, Java or Objective-C. Also interpreted (such as Python or JavaScript) and those based on virtual machines (Java and C#) due to the performance penalty (Figure 2.4). Finally, we have not used C++, since we do not present NAppGUI as a hierarchy of classes but as a library of functions. Our goals have been to minimize the impact of the SDK, simplify programming, increase readability, and produce high-performance binaries.

Figure 2.4: Interpreter, virtual machine, and binary code. The closer we get to machine language, the more performance we will get from the software.

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

2.3. No visual editors

The creation of graphical interfaces can become a tedious process, since it is difficult to know in advance the final size of elements that contain text or images, such as buttons. On the other hand, windows are dynamic entities subject to changes at runtime (size, translation, changing subpanels, hidden areas, etc.). When using a visual editor, we have to place elements at the exact (Figure 2.5) position and size. This is a mouse-intensive task, which slows down the connection between GUI objects and event handlers. In the development cycle, if the texts or other elements change (and of course they will), we will have to relocate the components by hand again. This problem grows in multilingual solutions. Keeping developers moving pixels and filling property forms is expensive for companies and very boring for them. This is not to mention that all of these visual designs will not be cross-platform compatible (.rc Windows, .xib macOS, .glade GTK/Gnome, etc.).

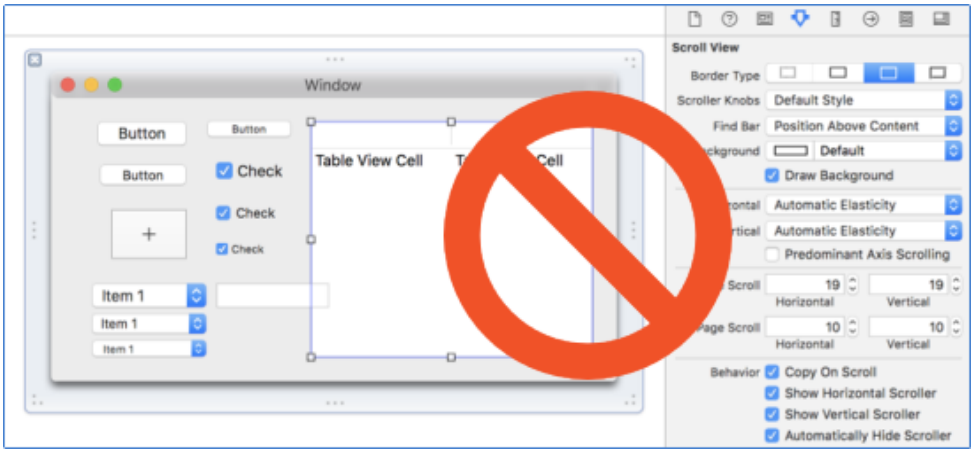


Figure 2.5: Resource editors are not good friends for creating complex dynamic interfaces.

> *Many programmers prefer not to move their hands from the keyboard, since they consider it much more productive.*

NAppGUI uses a declarative strategy, where it is only necessary to indicate the cell where the element will be located within a rectangular grid (`Layout`). The final size and position will be calculated at runtime, performing a recursive composition of the *layouts* and *sublayouts* based on their (Listing 2.1) content .

Listing 2.1: Creating a window.

```
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!");
```

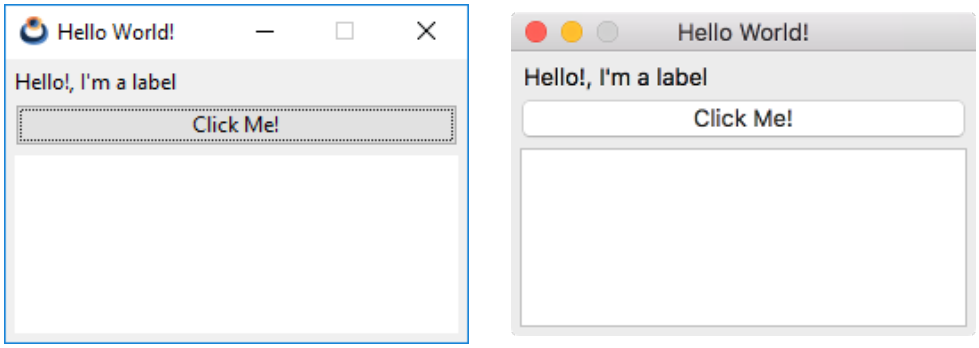


Figure 2.6: Declarative composition is fast, adaptable, and portable.

2.4. Dependencies

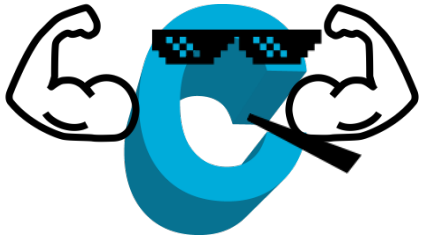
NAppGUI does not use third-party libraries, it only connects with the native APIs of each operating system. This fact, together with the use of C and static linking, makes it possible to:

- Applications don't need additional runtimes like Python, Java, or C# do. They go directly to the CPU via the system *scheduler*.
- The entire application can be contained in a single `.exe` file. As little code as possible is linked and no additional `.dll` need to be distributed.

As of version 1.3, NAppGUI supports the generation of dynamic libraries.

- Applications take up very little disk space, since all their dependencies are naturally present on the systems where they run.

- The performance is maximum, since they are compiled in native machine code, using the highest level of optimization that each CPU supports.
- They can be edited, compiled and run on obsolete platforms today like a Pentium III with Visual Studio 2005 and WindowsXP.
- With NAppGUI we can move them from Windows to macOS or Linux, without touching a single line of source code. See *"Compilers and IDEs"* (page 73).



Three packages within the SDK will act as technology *wrappers* (Figure 2.7), hiding platform-specific details under a common interface, without causing overhead to the program.

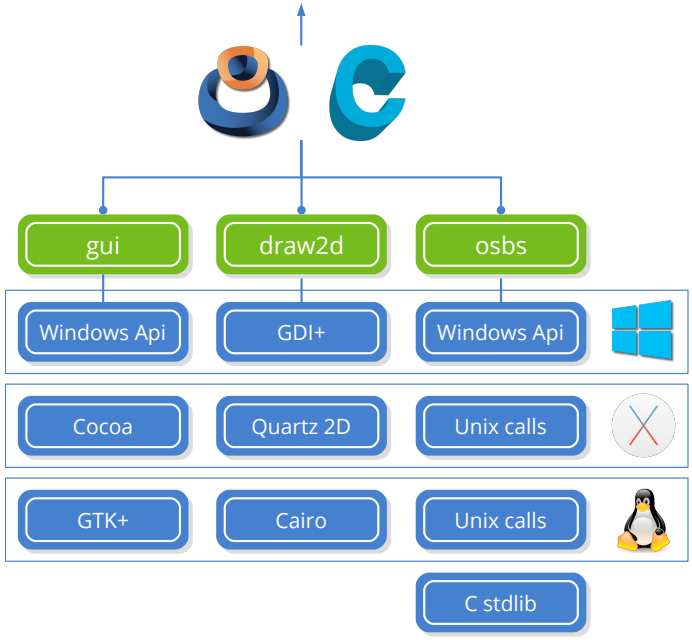


Figure 2.7: Different technologies at the base of NAppGUI. In *"NAppGUI API"* (page 145) you have the complete schematic.

- *"Osbs"* (page 166): *Operating System Basic Services.* API about files and directories, processes, threads, memory, etc.
- *"Draw2D"* (page 256): API for 2d vector drawing, images and fonts.

- *"Gui"* (page 297): API about graphical interfaces: Windows, controls and menus.
- **Unix system calls**: In Unix-like systems (Linux, macOS) it is the way in which a program communicates with the kernel to perform some task related to files, processes, memory, network or hardware usually.
- **Windows API**: It is the lowest level API provided by Microsoft for programming under Windows. It is very broad and integrates different aspects of development:
  - kernel32.dll: The equivalent of Unix calls (files, processes, memory, etc).
  - ws2_32.dll: Provides TCP/IP network functions (Unix calls include TCP/IP support).
  - user32.dll, comctl32.dll, comdlg32.dll, uxtheme.dll: Implements standard controls for graphical user interfaces (labels, edit boxes, combos, progress bars, common dialogs, etc.).
- **Cocoa**: Object-oriented programming API for Mac OSX (now macOS) systems. It is written in Objective-C, therefore it is not directly accessible from "pure" C. Cocoa is based on OpenStep, the API of NeXTSTEP, the operating system created by Steve Jobs when he was fired from Apple. In 1996, Apple buys NeXT and gets Jobs back, using Jobs' technology as the basis for the new Macintosh. Many classes in Cocoa still retain the NS prefix as NeXTSTEP inheritance. Although there is a lower level C-based API called Carbon, it has been discontinued since Mac OSX 10.4 Tiger. It does not have access to all system functionality nor is it compatible with 64-bit applications. Thus, Cocoa is the current lowest level API for Apple systems.
- **Gtk+**: Acronym for **G**IMP **T**ool**K**it. It is a high-level library for creating graphical interfaces with a multitude of predefined objects (called *widgets*). It is one of the most widespread in GNU/Linux systems, but it is actually multiplatform with versions for Windows and macOS. Desktop environments like Gnome, Xfce or applications like GIMP are based on GTK.
- **GDI+**: It is the evolution of GDI *(Graphics Device Interface)*, a 2d vector drawing API developed by Microsoft for the first 16-bit version of Windows. GDI+ was introduced with Windows XP as a set of C++ classes and is encapsulated in the .NET platform via the `System.Drawing` namespace. It is also accessible directly from C via the *GDI+ Flat API*, but Microsoft recommends using it via C++ classes. It incorporates substantial improvements over GDI, such as floating point coordinates, affine transformations, anti-aliasing, gradient shading, and support for image formats such as JPG, PNG, or GIF. Drawing with masks and incompatibility with PDF are the two most notable drawbacks compared to Quartz 2D and Cairo, its direct "competitors" on other platforms.
- **Quartz 2D**: It is the trade name of *Core Graphics*, the powerful drawing API

of macOS. Like Cocoa, Core Graphics is an evolution of the NeXTSTEP graphics libraries and came to Apple after the NeXT acquisition. Quartz 2D is based on Adobe PostScript and PDF formats, incorporating alpha channel and anti-aliasing. Classic Macs (pre-NeXT) used the *QuickDraw* library, originally developed by Bill Atkinson for the Apple Lisa. Modern macs still have QuickDraw built in, but Xcode no longer provides headers, so it can't be used in new projects. Core Graphics is a C-based API and all of its functions begin with the **CG** prefix.

- **Cairo**: Cairo is a C-based 2d vector drawing library. Unlike GDI+ or Quartz 2D, it is cross-platform, can be downloaded independently and incorporated into any project (under LGPL license) . Since version 3, GTK+ uses Cairo for all widget drawing tasks. GTK+2 also used Cairo to generate PDF documents for printing. NAppGUI uses Cairo to implement the **draw2d** API on the GNU/Linux platform, as this library is found naturally in all GTK+ based desktop environments: Gnome, Cinnamon, LXDE, Mate, Pantheon , Sugar or Xfce. Technically, Cairo is quite advanced, matching Quartz 2D in terms of functionality. It supports affine transformations, image masks, bezier curves, text processing, and drawing on PDF and PostScript surfaces.

- **C stdlib**: C is a beautiful little language, but it doesn't provide any additional support functions. During the 1970s, the C language became very popular and users began to share ideas on how to solve common and repetitive tasks. With its standardization in the 1980s, some of these ideas became the C standard library, which provides a basic set of mathematical functions, string manipulation, type conversions, and input/output. NAppGUI integrates in one way or another the functionality of the standard library, so we do not recommend its use in final applications (see *"Sewer"* (page 149)).

2.5. Low and high level

During its design and implementation, NAppGUI has tried to maintain a balanced balance between low-level and high-level programming. Low-level lovers will find a kind of *extended and cross-platform C library* to access the system, interface elements and drawing commands. However, they will still retain the power to create optimized code and direct memory access. Remember, we are in C!

On the other hand, NAppGUI integrates some high-level solutions such as resource management, interface composition, automatic translations or data binding, among others. NAppGUI also incorporates CMake scripts for automated project creation in Visual Studio, Xcode, or Eclipse/Make.

Finally, it is the developers who decide which libraries to link with according to the needs of the project and the degree of automation they wish to adopt. Each application based on

NAppGUI performs a static link of all its dependencies, so neither the executable nor its final distribution will have traces of unnecessary binary code. In this way, we will produce small self-contained executables that will not require an installer or include megabytes of dependencies in the form of .DLLs.

Chapter 3

Hello World!

Once upon a time, there was a company called Taligent. Taligent was created by IBM and Apple to develop a set of tools and libraries like Cocoa. About the time Taligent reached the peak of its mindshare, Aaron met one of its engineers at a trade show and asked him to create a simple application: A window appears with a button. When the button is clicked, the words "Hello, World!" appear in a text field. The engineer created a project and started subclassing madly, subclassing the window and the button and the event handler. Then he started generating code: dozens of lines to get the button and the text field onto the window. After 45 minutes, he was still trying to get the app to work. A couple of years later, Taligent quietly closed its doors forever.

Hillegass, Preble & Chandler - Cocoa Programming for OSX.

There is little we can say about the meaning of the *Hello World!* program every time we are faced with a new technology or programming methodology. So, let's get down to business.

3.1. The complete program

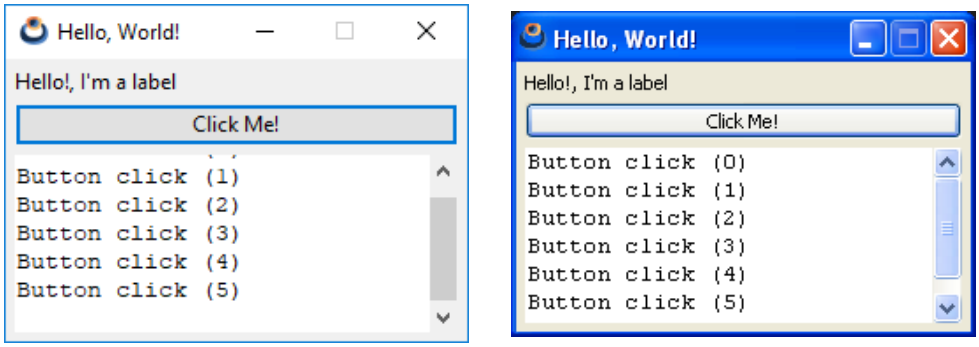


Figure 3.1: Windows 10 y Windows XP.

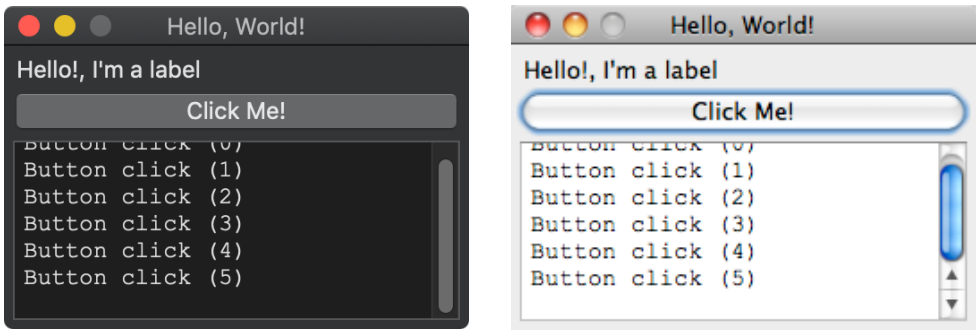


Figure 3.2: macOS 10.14 Mojave and MacOSX 10.6 Snow Leopard.

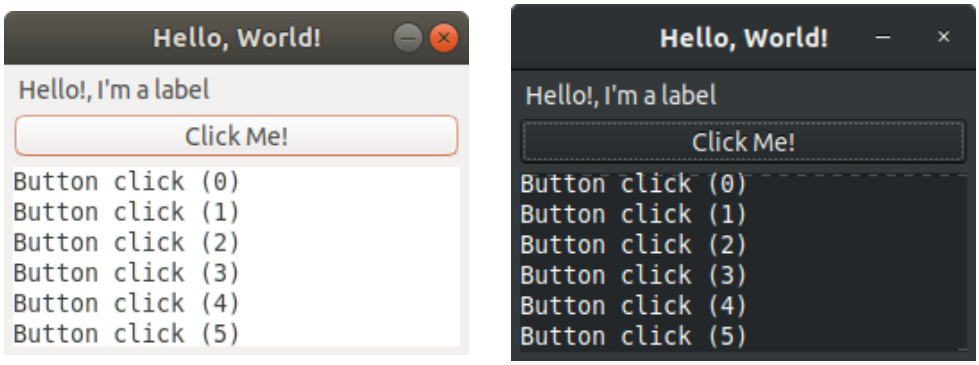


Figure 3.3: GTK+3 Ambiance (Ubuntu) and Adwaita Dark (Raspbian).

Listing 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. The skeleton

A NAppGUI application starts at `osmain`, a cross-platform macro that unifies the startup of a desktop program under different systems. It is defined in `#include "osmain.h"` and will receive four parameters: constructor, destructor, arguments (`char_t`), and the object type. In this way, any basic skeleton looks like this:

```
#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)
```

The `#include "nappgui.h"` directive, includes much of NAppGUI with a single statement. If you prefer, you can choose to include the headers separately as needed. In this case, we should replace a single `#include` with eleven. In the Reference Manual, it is indicated which header to include according to the function module that we are going to use.

```
#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. The constructor

The first parameter of `osmain` is the application constructor. As soon as the program starts, certain internal structures must be initialized, as well as starting the message loop inherent to all desktop applications. When everything is ready, the constructor will be called to create the **application object**. This object can be of any type and does not need to be derived from any `class Application` or similar, we are in C ;-). Because of the simplicity of this example, the application object contains only one window.

```
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. The main panel

To create the main window, we need the **main panel**, a container that integrates all the interface controls that are displayed in the window. The space inside the panel is arranged in an invisible grid called `Layout`. Each panel can have multiple layouts and switch between them, but at least one is required. Within its cells we will locate the different interface controls.

```
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. The destructor

When the application terminates, `osmain` will call the destructor (macro's second parameter) to free the application object and everything that depends on it, in order to perform a clean exit from the program. We'll put **a lot of emphasis on this**, as failure to properly free all memory will be considered a serious programming error.

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

3.6. Launch the window

By default, NAppGUI creates all hidden windows, so you need to show them explicitly. We set a title, an initial position and launch it with `window_show`. We note that in this first version our window does not look very aesthetic (Figure 3.4). In a moment we will format it.

```
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);
    ...
}
```

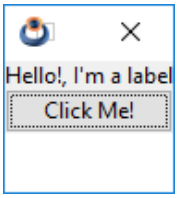


Figure 3.4: First version of *Hello, World!* (without format).

3.7. Layout format

To improve the appearance of our window, let's format the layout a bit. Specifically, we are going to set a column width and a height for the third row (text control). Then we will leave a margin on the edge and a separation between rows. (Figure 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);
```

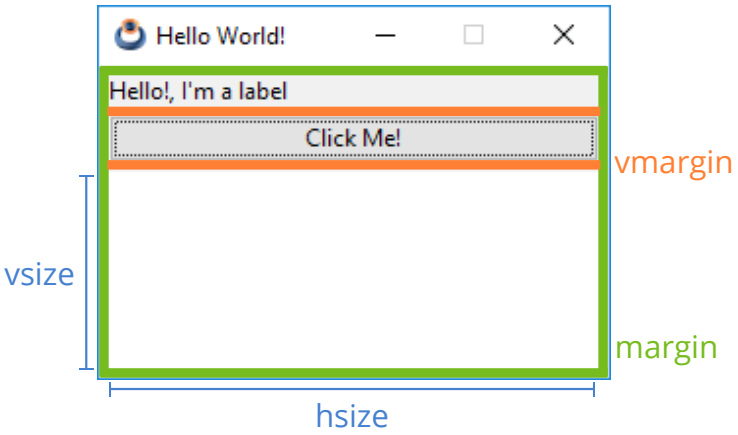


Figure 3.5: *Hello World!* after Layout formatting.

3.8. Exiting the program

When we press the button to close the main window, the program does not end its execution. This is typical of macOS applications, where they still run in the Dock even though no window is open. NAppGUI follows the same criteria of not closing the program, so we must make an explicit call to the `osapp_finish` function. To do this, we will capture the **button event**, through the `listener` macro.

```
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. Button Events

Finally, we'll catch the *click* event of the button and print a message in the text box each time it's clicked. We are going to implement the `i_OnButton` handler, responsible for composing and displaying the message, and connect it to the Button control we created earlier.

```
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));
```

An event is an action that occurs during the execution of the program. The operating system captures it and sends it to us via its controller (defined in `listener()`). More at "Events" (page 230).

Chapter 4

Use of C

Most programming languages contain good parts and bad parts. I discovered that I could be a better programmer by using only the good parts and avoiding the bad parts. After all, how can you build something good out of bad parts?

Douglas Crockford - JavaScript: The Good Parts.

Programming fast, reducing the probability of error, ensuring portability and generating optimized binaries have been the main purposes of NAppGUI since its inception and that includes a revision of the C language itself. A **subset> has been used as a base ANSI-C90** with fixed-size integers `<stdint.h>`, a feature introduced in C99. We recommend that applications based on this SDK follow the same philosophy. Going into more detail, the objectives pursued have been these:

- Maximum portability: Even on already outdated compilers like MSVC 8.0 (Visual Studio 2005) or GCC 4.2 (Xcode 3). The latest language features may not be avail-

able on platforms where you must port your code (think embedded devices). You also ensure that such code will be compatible with future versions of major compilers.

- Focus attention: On the "what" and not on the "how". There are times when we make the simple complicated just to justify the use of that new "cool" feature. It is also possible that you are a "hip" addict, which will force you to "modernize" the code to adapt it to a new version of the standard. Focus on solving the problem at hand and, if you can, spend more time on lowering the asymptotic complexity of your solution. NAppGUI will make sure that your applications work wherever they are needed.
- Avoid irrelevant features: Like C11's multi-threading support (`<threads.h>`). This is solved with system calls. See *"Threads"* (page 170).
- Fast compilation: Certain C constructs are nothing more than a kind of "portable assembler", which the compiler can interpret and translate incredibly efficiently.
- Small and fast binaries: Derived from the previous one, the generated code will require few assembly statements and will be very easy for the compiler to optimize.

Evidently, this is not the place to learn C nor is it our intention. The core of the language is small and easy to remember, but programming well requires years of practice. What we will do here is show the minimum expression of the language that we use daily. In short, these are our standards.

4.1. Basic types

- **Void:** `void`.
- **Boolean:** `bool_t`. 8-bit type with only two possible values `TRUE` (1) and `FALSE` (0).
- **Integers:** `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t`, `int8_t`, `int16_t`, `int32_t`, `int64_t`. Fixed-size integers were introduced in C99 by `<stdint.h>`. We consider it an advantage to know that our variables will have the same size in all systems. The use of `int`, `long`, `short` or `unsigned` is prohibited, with the sole exception of the comparison functions .

```
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);
```

- **Floating point:** `real32_t`, `real64_t`. `float` and `double` are not used for consistency with integer types.

- **Character:** `char_t` (8 bits). The UTF8 representation is used "de facto" throughout the SDK, so random access to elements of a string is prohibited, since it is a variable-length encoding. Functions included in *"Unicode"* (page 155) or *"Strings"* (page 192) must be used to manipulate arrays of characters. The types `wchar_t`, `char16_t`, `char32_t` are not used (or recommended). However, if you have *wide-char* strings you will need to convert them to UTF8 before using them in any NAppGUI functions.

Using UTF8 strings

```
/* 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);
```

- **Enumerated:** Their main task is to manage the specialization and they will be evaluated exclusively within a `switch`. It is forbidden to assign random values to the elements of an `enum`, except `1` to the first one. Consider `0` as **not initialized** and `ENUM_MAX(align_t)` as **invalid**.

Defining enumerated types

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

4.2. Structures and unions

Definition of structures and unions

```
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;
};
```

In general, structure definitions will not be public and will remain hidden in the `*.c`. This means that automatic variables cannot be declared in the *"Stack SegmentStack Segment"* (page 162) and will only be accessible by functions that accept **opaque dynamic objects**.

Use of opaque pointers

```
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! */
```

Normally, all dynamic objects will have a destroy function. If it does not exist, it is because said object **only makes sense as part of another object**. For example, there is no `layout_destroy()` or `panel_destroy()`, but there is `window_destroy` which will destroy the entire hierarchy of panels and associated layouts to the window.

4.3. Control

- **if/else**. They always open a `{...}` block, unless ALL paths consist of a single statement. Using functions as arguments to `if/else` is generally avoided with the exception of **pure functions**.

  Use of 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**. Nothing to comment.
- **do/while**. Not allowed. Use `for` or `while`.
- **for**. For infinite loops, use `for(;;)` instead of `while(TRUE)`, as it avoids warnings in some compilers. Since there are ANSI-C based compilers, such as MSVC++ 8.0, we **do not use variable declarations** inside the `for()`, a feature that was introduced in C99.

Use of 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**. It is only used to discriminate between the values of an **enum**. Any other data type will NEVER be evaluated in a `switch` nor will an `enum` be discriminated within an `if/else` construct. The compiler can drastically optimize the performance of a build with these features.

Use of switch

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

case ekBOTTOM:
    ...
    break;

case ekLEFT:
    ...
    break;

case ekRIGHT:
    ...
    break;

cassert_default();
}
```

4.4. Functions

- A function can return nothing (`void`), a basic type, or a pointer.
- Input parameters are always **`const`** even if they are simple types passed by value.
- Any input parameter that is not of basic type will be passed by pointer. Never a structure by value.
- For the output parameters, pointers will always be used. In C there are no references.

Parameters in functions.

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

- The number of public functions should be kept to a minimum, which will be declared in the `*.h` and defined in the `*.c`.
- Supporting (or private) functions will be defined `static`, inside the `*.c` module and will have no declaration.

Public function.

```
/* 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;
}
```

Private function. It can only be called inside `layout.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. Scopes

Variables are declared at the beginning of a block and cannot be mixed with statements, unless we open a new scope. Declarations mixed with statements is a C++ feature added to the C99 standard, but not all C compilers support it. Yes, it is allowed to initialize a variable by calling a function.

Variable scopes in 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. Pointers

Apart from the advantages of using pointer arithmetic when implementing certain algorithms, in NAppGUI pointers are used essentially in two situations:

- Passing parameters to a function, when said parameter is not a basic type.

Passing of parameters through pointers.

```
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);
```

- Handling opaque objects. Where the definition of the `struct` is not available and therefore the only way to communicate with the object is through functions that accept a pointer to it.

Use of opaque objects.

```
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);
```

Special mention should be made of the **function pointers** that are widely used in C, but less so in C++ as the language hides them inside **vtables**. However, a strategically placed function pointer can make it easier for us to add specialized functionality to existing objects, without having to adopt a more purist object-oriented design.

Listing 4.1: Use of function pointers.

```
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. Preprocessor

Our standards make heavy use of the preprocessor, especially for type checking at compile time. This helps to detect errors in the code before running the program (static analysis), as opposed to the C++ RTTI that does it once it is running (dynamic analysis).

Using the preprocessor to check types.

```
#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);
```

Dynamic typing is not necessarily good. You get static errors at runtime, which really should be catchable at compile time. ***Rob Pike****.*

4.8. Comments

In general, the use of comments will be reduced as much as possible. A comment will be placed at the beginning of each file as a general description. We also use a comment line as a separator when implementing functions.

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 */
}
```

C++ comments `//Comment...` *are NOT allowed, as they generate warnings in certain* `gcc -std=gnu90` *compilers.*

Another aspect that is **totally prohibited** is the inclusion of documentation blocks within the source code, not even in the headers themselves. NAppGUI uses `ndoc` for documentation tasks, a utility that allows you to create html/pdf documents enriched with images, cross-references, examples, etc. and that uses its own files totally separated from the code. Another added advantage is the cleanliness of the `*.h` headers of all the modules, where it is very easy to locate what we are looking for.

Documentation blocks are NOT allowed.

```
/* Forbidden, non used */
```

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

Header example in 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
```

All comments in NAppGUI are made in English language.

4.9. Input/Output

Input/output is not part of the C language as such. As the language spread in the mid-1970s, a number of useful routines were grouped together into what became the **Standard C Library**. NAppGUI encapsulates all its functionality in *"Sewer"* (page 149), *"Osbs"* (page 166) or *"Core"* (page 187) generally implementing it as much more direct and efficient calls to the operating system.

Use of safe I/O functions.

```
/* 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);
```

*Use of the **Standard C Library** is not recommended. Look for the equivalent function in **Sewer**, **Osbs**, or **Core**.*

4.10. Mathematical algorithms

NAppGUI uses *C++ templates* to implement any function or mathematical algorithm. With this it is possible to offer `float` and `double` versions in an elegant way and with easy maintenance. The templates are hidden and not exposed in the API, so that their use remains ANSI-C90 compliant. For more information *"Math templatesMath templates"* (page 53).

NAppGUI makes internal use of C++98 `template<>` to implement everything related to mathematical calculation.

Chapter 5

Use of C++

Web servers are written in C, and if they're not, they're written in Java or C++, which are C derivatives, or Python or Ruby, which are implemented in C.

Rob Pike

Object-oriented programming (encapsulation, inheritance and polymorphism) is a very powerful tool for modeling certain kinds of problems. However, at NAppGUI we believe that it is wrong to impose a class hierarchy at the SDK level, as this is too low a level. The SDK is closer to the operating system and the machine than to the real-world problems solved by applications, where an object-oriented approach may (or may not) be more successful.

Although NAppGUI has been designed to create applications in "pure" C, it is possible to use C++ or mix both languages. We'll give some advice, porting our *"Hello World!"* (page 23) application to C++ .

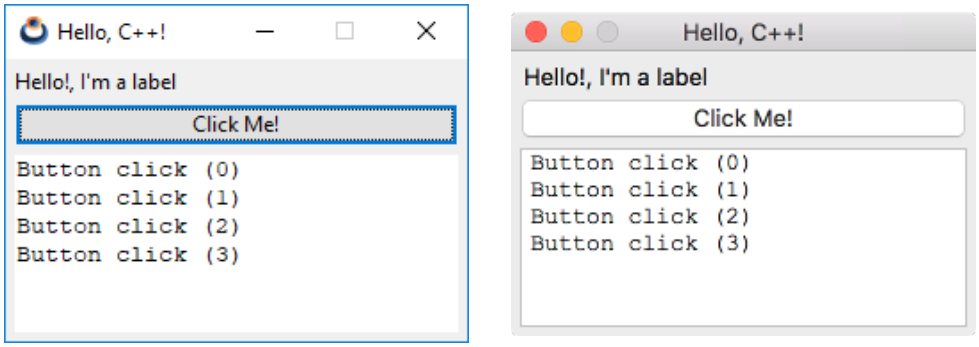


Figure 5.1: Migration from *Hello, world!* to C++.

5.1. Encapsulation

NAppGUI does not enforce any class hierarchy, leaving the programmer the freedom to encapsulate using their own classes. Of course, since C++ includes C, we can call any SDK C function inside a member function. For example, we can encapsulate the main window like this.

```
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;
};
```

As you can see, relative to the C version, `i_panel` no longer needs parameters, as it uses the implicit `this` pointer to access class members.

5.2. Class callbacks

Event handlers are C functions whose first parameter is a pointer to the object that receives the message. This works the same way using static functions within a C++ class:

```
...
static void i_OnClose(MainWindow *window, Event *e);
```

```
...
window_OnClose(this->window, listener(this, i_OnClose, MainWindow));
...
```

However, we may want to use member functions as event handlers, using the **this** pointer as the receiver. To do this, we derive our `MainWindow` from the `IListener` interface and use the `listen` macro instead of `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` is a C++ interface that allows you to use class member methods as event handlers.

It is also possible to direct the event to a different object (and of a different class) than the control owner. To do this, we indicate the receiver as the first parameter of `listen`, as we see below. The click of the close button will be processed in the `App` class and not in `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();
}
```

We can establish as event receiver, any object that implements the `IListener` interface.

5.3. Combine C and C++ modules

A C/C++ project selects the compiler based on the file extension. For ***.c** the C compiler will be used and for ***.cpp** the C++ compiler. The same project can combine modules in both languages if we consider the following.

5.3.1. Using C from C++

There is no problem if the C header function declarations are between the macros: `__EXTERN_C` and `__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` and `__END_C` are aliases for `extern "C"{}`. This tells the C++ compiler not to use name mangling[a] with C functions.

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

5.3.2. Using C++ from C

C does not understand the `class` keyword and will give a compile error when including C++ headers. It is necessary to define an interface in C over C++ code.

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. new and delete overload

C++ uses the `new` and `delete` operators to create dynamic instances of objects. We can make reservations through `Heap`, the *"Heap - Memory manager"* (page 188) manager that NAppGUI incorporates, in order to optimize C++ and control *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. Hello C++ complete

Listing 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. Math templates

In NAppGUI there are two versions for all (Listing 5.2) functions and math types: `float (real32_t)` and `double (real64_t)`. We can use one or the other as appropriate in each case.

Listing 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
```

*All single-precision functions and types end with the suffix **"f"** and double-precision types end with **"d"**.*

When we implement more complex geometric or algebraic functions, it is not easy to be clear in advance what the correct precision is. When in doubt, we can always choose to use `double`, but this will have an impact on performance, especially due to the use of memory bandwidth. Consider the case of 3D meshes with thousands of vertices. It would be great to have both versions and be able to use one or the other according to each specific case.

Unfortunately the "pure" C language does not allow programming with generic types, apart from using horrible and endless macros. We will have to implement both versions (`float` and `double`), with the associated maintenance cost. C++ solves the problem thanks to templates (`template<>`). The downside is that, normally, we must "open" the implementation and include it in the `.h` header, since the compiler does not know how to generate the machine code until the template is instantiated with a specific data type. . This is in direct conflict with our *"StandardsStandards"* (page 58), especially in the part related to information encapsulation. Next we will see how to use C++ `templates` to get

the best of both cases: Generic programming, hiding implementations and keeping headers "clean".

Just as there is a `*.h` header for every math module, there is a counterpart `*.hpp` usable only from C++ (Listing 5.3) modules.

Listing 5.3: Header `bmath.hpp` (partial).

```
/* 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;
};
```

These templates contain pointers to functions, whose implementations are hidden in `bmath.cpp`. In (Listing 5.4) we have an example of use.

Listing 5.4: Implementation of a generic algorithm.

```
#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);
    }
}
```

This algorithm is implemented within a C++ module (Listing 5.5), but we want to be able to call it from other modules, both C and C++. To do this we will define the two types of headers: `*.h` (Listing 5.6) and `*.hpp` (Listing 5.7).

Listing 5.5: `mymath.cpp`. Implementation.

```
#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>;
```

Listing 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
```

Listing 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);
};
```

Now we can use our math library in C and C++, both in `float` and `double` precision (Listing 5.8).

Listing 5.8: Using `mymaths` in generic C++ algorithms.

```
#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]);
}
```

Chapter 6

Error management

There is always one more bug to fix.

Ellen Ullman

Developing software of a certain size and complexity can become a hellish task, if we do not adopt concrete measures to prevent and quickly locate programming *bugs*. Next we will talk about some strategies that we have used in the development of NAppGUI and that you can apply in your own projects.

6.1. Exhaustive tests

Ensuring that our software is bug free is as "easy" as running a test for each and every case the (Figure 6.1) program will face.

Already from trivial theoretical examples, we see that we are dealing with an exponential problem (Figure 6.2), which will overwhelm the resources of any system with relatively few input variables. Therefore, we can intuit that it will be **impossible** to guarantee that

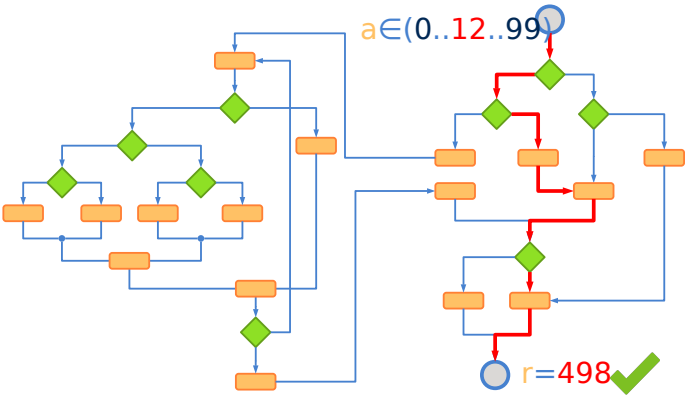


Figure 6.1: *Exhaustive tests* use all possible combinations of the input data.

our software is free of errors since it will not be feasible to reproduce all its use cases. However, we can define a strategy that helps us minimize the impact that these will have on the final product, detecting and correcting them as soon as possible.

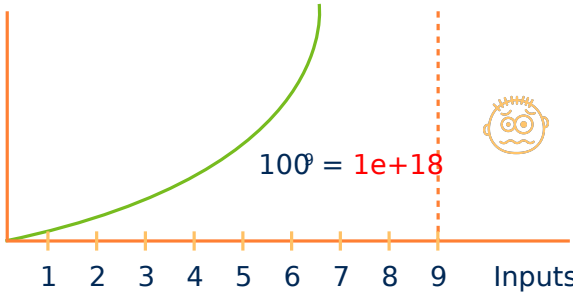


Figure 6.2: With only 9 input variables (in range 0..99) the computational resources will overflow.

6.2. Static analysis

Static analysis is the one that is carried out **before executing the program** and consists of two parts: The use of **standards** where rules and quality controls are applied during the writing of the code itself. And the **compiler warnings** that will help us locate potential compile-time errors.

6.2.1. Standards

The use of standards, understood as rules that we follow when programming, is essential when it comes to maintaining minimum levels of quality in our (Figure 6.3) projects. If they are not applied, a program of a certain size will become anarchic, unreadable, difficult to maintain and difficult to understand. In this scenario it will be easy to add new bugs as we manipulate the source code.

In reality, it is difficult to differentiate between good and bad standards, since they will depend on the type of project, programming languages, company philosophy and objectives

Figure 6.3: Using standards will reduce the probability of *bugs*.

to prioritize. We can see them as a *Style Guide* that evolves over time hand in hand with experience. What is truly important is to become aware of their usefulness, define and apply them. For example, if we decide to name variables with descriptive identifiers in English and an underscore (`product_code`), all our code should follow this rule without exception. Let's take a look at some of the standards we apply within NAppGUI. They are not the best nor do they have to adapt to all cases. They are only ours:

- Use a small subset of the language, as we've seen in *"Use of C"* (page 31). For example, expressions of the type `*((int*)block + i++)= i+1`, are totally prohibited. They are perfectly valid in C but poorly readable and confusing. Some programmers think that cryptic and compact code is much more maintainable, but we think they are wrong.

- Comments are prohibited, except on rare occasions and very justified. If something needs a comment, rewrite it. A comment that even slightly contradicts the code it is intended to clarify causes more confusion than help. And it is very easy for them to become obsolete.

- Reduced and clean public interfaces. Header files (`*.h`) represent a high level of abstraction as they reduce the connections between software components (Figure 6.4). They allow condensing, as an index, hundreds or thousands of lines of code in just fifteen or twenty public functions. It is completely forbidden to include type definitions (they will go in the `*.hxx`), comments (of course) and documentation blocks in `.h` files.

- Opaque objects. Object definitions (`struct _object_t`) will be made inside the implementation files (`*.c`) and never in the `*.h`. The objects will be manipulated with public functions that accept pointers to them, always hiding the fields that compose them. This point, together with the previous one on interfaces, perfectly defines the barriers between modules, clearly marking when one problem ends and

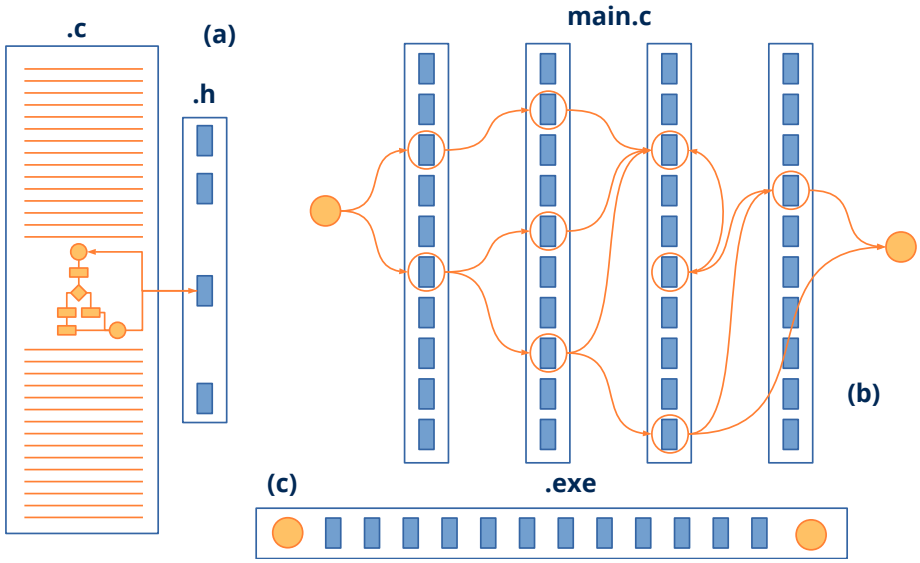


Figure 6.4: The `*.h` headers provide a high level of abstraction hiding the complexity of the **(a)** solution. They facilitate horizontal, problem-based development, as opposed to vertical learning based on **(b)** APIs. They help the linker reduce the size of the **(c)** executable.

another begins.

The first two rules help reduce the internal complexity of a module by making it as readable and less cryptic as possible. We could enrich them with others about indentation, style, variable naming, etc. We more or less strictly follow the advice of the great book *The Practice of Programming* (Figure 6.5).

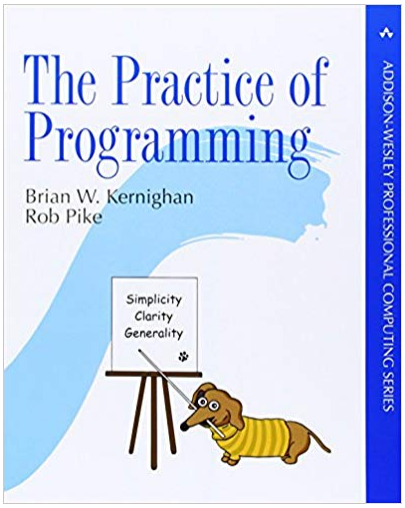


Figure 6.5: *The Practice of Programming* by Brian W. Kernighan and Rob Pike is a good source of inspiration for defining your own programming style.

6.2.2. Compiler warnings

The compiler is our great ally when it comes to examining the code for possible (Figure 6.6) errors. Enabling the highest possible level of *warnings* is essential to reduce errors caused by type conversions, uninitialized variables, unreachable code, etc. All projects built with NAppGUI will trigger the highest level of warnings possible, equivalent to `-Wall -Wpedantic` on all (Figure 6.7) platforms.

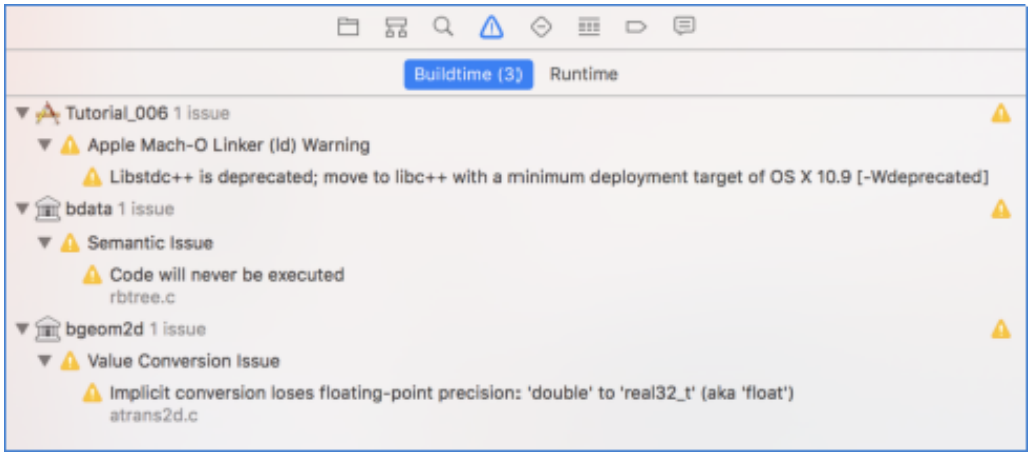


Figure 6.6: Fixing all compiler *warnings* should be a priority.

6.3. Dynamic analysis

Dynamic analysis is performed once the program is running. Here our main weapon is self-validations, implemented as *"Asserts"* (page 153) statements. Asserts are checks distributed throughout the source code, which are evaluated at runtime each time the program goes through them. If a statement resolves to `FALSE`, processing will stop and an (Figure 6.8) informational window will be displayed.

```
void layout_set_row_margin(Layout *layout, const uint32_t row, const real32_t
    ↪ margin)
{
    cassert_no_null(layout);
    cassert_msg(row < layout->num_rows, "'row' out of range");
    ...
}
```

*It is also possible to redirect **assert** statements to standard output or to the **Log** file.*

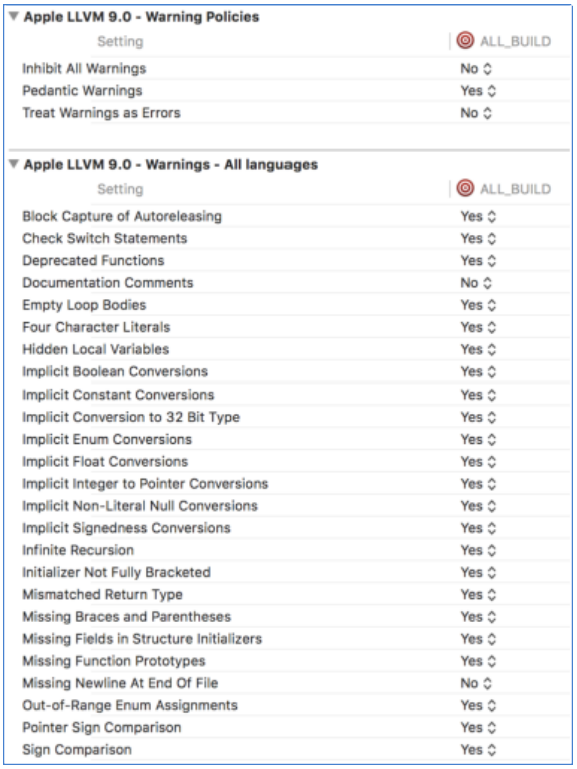


Figure 6.7: NAppGUI enables the highest level of *warnings* possible.

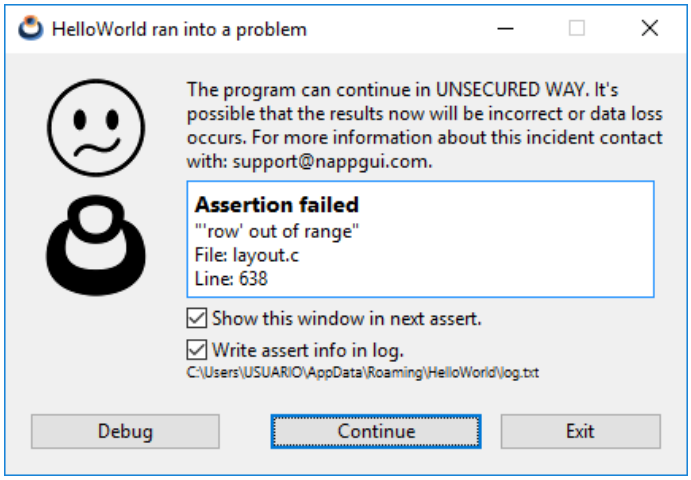


Figure 6.8: Window displayed after activating an assert.

6.3.1. Disabling Asserts

Within the NAppGUI SDK code, more than 5000 assertions have been distributed, located at strategic points, which constantly evaluate the coherence and integrity of the software. Obviously, this number will grow after each revision, as more functionality is integrated. This turns the SDK into a real minefield, where any error in the use of

the API functions will be automatically notified to the programmer. Depending on the configuration we are using, the assertions will be activated or deactivated:

- `Debug`: *Assert* statements are enabled.
- `Release`: The sentences *assert* are disabled.
- `ReleaseWithAssert`: As the name suggests, turns on all Release optimizations, but leaves *assert* statements on.

6.3.2. Debugging the program

When an assert is activated, the program stops right at the check point, showing the assert confirmation window. If we press the `[Debug]` button, we will access the *call stack* (Figure 6.9), which is the current function call stack, from the `main()` itself to the current breakpoint *"Stack SegmentStack Segment"* (page 162). By browsing the stack we can check the values of variables and objects at any call level. This will help us identify the source of the error, as the cause may be a few levels below detection.

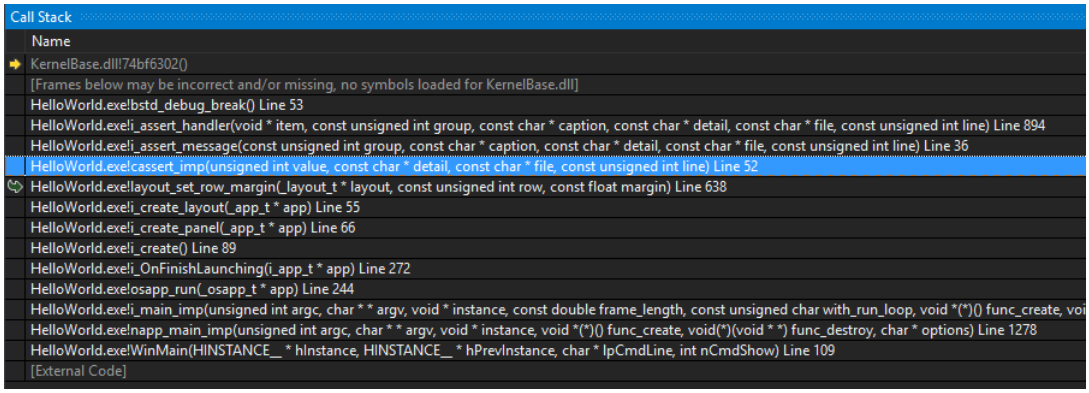


Figure 6.9: *Call stack* while debugging the assertion from the previous example.

6.3.3. Error log

An execution *"Log"* (page 184) is a file where the program dumps information about its status or anomalies detected. It can be very useful to know the cause of a failure when the software has already been distributed and it is not possible to debug it. NAppGUI automatically creates a log file for each application located in the application data directory `APP_DATA\APP_NAME\log.txt`, for example `C:\Users\USER\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'
```

As you can see, the assertions are automatically redirected to the *log* file. It is possible to disable this writing by unchecking the `'Write assert info in log'` check in the info window. You can also add your own messages using the `log_printf` method.

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

6.3.4. Memory auditor

NAppGUI's memory manager *"Heap - Memory manager"* (page 188) has an associated auditor that checks for *leaks* memory after each execution of each application that uses the SDK. This is a great advantage over using external utilities, as dynamic memory checks are being performed **always** and not in isolated phases of development.

```
[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.
```

Chapter 7

Generate NAppGUI binaries

In *"Quick start"* (page 5) we already saw how to download, compile and run the examples from the source code. In this chapter and the next, we'll dive deeper into the build process and cross-platform portability. The entire *build system* of NAppGUI centers around a single `src/CMakeLists.txt` (Listing 7.1) script. It will define a solution with several related projects, as well as their dependencies.

Listing 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. Generate static libraries

By default, the `CMakeLists.txt` will create the static version of the NAppGUI libraries. If you want to use NAppGUI externally in your projects, you just have to follow these steps:

```
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
```

In the `install` folder you will have the binaries and headers:

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

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

- In `/install/inc` you will find the header files of each library.
- In `/install/lib` you will find the static libraries (.lib, .a).
- In `/install/bin` you will find the sample executables.
- `v143_x64` identifies the compiler and architecture. See *"Compilers and IDEs"* (page 73) for more information about supported compilers, platforms, and architectures.
- `Debug` is one of three possible configurations: `Debug, Release, ReleaseWithAssert` . *"ConfigurationsConfigurations"* (page 95)

If you don't want to compile the sample applications, remove the ***desktopApp*** *lines from the script.*

7.2. Generate dynamic libraries

To generate the dynamically linked versions (.dll, .so, .dylib) of NAppGUI, edit `CMakeLists.txt`, replacing the `staticLib` commands with `dynamicLib` . Once this is done, compile and install using `cmake` in the same way as in the static case.

```
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()
```

After installation:

- In `/install/inc` you will find the header files of each library.
- In `/install/lib` the symbol import libraries of the `.dll` (`.lib`) will be stored, only on Windows.
- In `/install/bin` the dynamic libraries `.dll`, `.so`, `.dylib` will be stored together with the example executables.

```
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
```

If you are going to use these libraries in third-party projects, not generated using `CMakeLists.txt`, you must previously define these macros in order for the symbols to be imported correctly.

```
#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. More about CMakeLists.txt

NAppGUI simplifies the use of CMake by providing high-level functions, located in the `/prj` folder of the distribution. The `CMakeLists.txt` defines a solution where different libraries and related executables coexist through dependencies. After being processed by CMake this script will create, in the `/build` folder, the build projects for each platform (VisualStudio, Xcode, Make). Within the script we will work with essentially four commands:

- `staticLib`: To create *"Static librariesStatic libraries"* (page 107).
- `dynamicLib`: To create *"Dynamic librariesDynamic libraries"* (page 114).
- `desktopApp`: To create *"Desktop applicationsDesktop applications"* (page 99).
- `commandApp`: To create *"Command line applicationsCommand line applications"* (page 103).

In the following example we define a solution that contains a dynamic library and two applications, one desktop and one command line. Both make use of (depend on) said dynamic library and NAppGUI (static libraries) for the graphical interface and cross-platform support.

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()
```

- Line 6: Minimum required version of CMake.
- Line 7: Name of the project or solution.
- Lines 10-11: Includes the NAppGUI CMake scripts, located in `/prj`.
- Lines 14-22: Generate the static libraries that make up NAppGUI.
- Line 25: Generates a dynamic library with the user's own functions.
- Line 28: Generates a desktop application.
- Line 29: Generates an application by command line.
- Line 31: Processes the solution. This command should be the last one in the script.

7.4. Why nine independent libraries?

NAppGUI provides full cross-platform support at various levels. It is not necessary to create an application with a graphical interface to take advantage of the advantages it offers us in terms of code portability. We can develop powerful server-oriented command line *back-end* applications. Depending on the level of assistance that each project requires, we can choose to link these libraries. More information in *"NAppGUI API"* (page 145).

- *"Sewer"* (page 149): Basic types, assertions, Unicode, math functions, wrapper on top of the C standard library.
- *"Osbs"* (page 166): Operating system services. Portable API on files, directories, processes, threads, memory, etc.

- *"Core"* (page 187): Commonly used non-graphical utilities. Memory auditor, data structures, strings, streams, regular expressions, resources, etc.
- *"Geom2D"* (page 235): 2D geometry. Transformations, vectors, polygons, collisions, etc.
- *"Draw2D"* (page 256): Portable vector drawing API, images and fonts. It can be used in **command line applications**, since it is possible to draw in memory and export to a file or transmit over the network.
- **osgui**: Low-level access to the user interface elements (widgets or controls) of each operating system. It is not documented and it is not recommended to use it directly.
- *"Gui"* (page 297): Composer of user interfaces. Use `osgui` to render.
- *"OSApp"* (page 369): Implements the message loop of a desktop application. Only use it to **build applications from scratch**. If you only need to create windows in an existing application, `gui` will be the top-level dependency.
- *"INet"* (page 377): Use it if your application is going to use Internet protocols like HTTP. Valid for command line or desktop applications.

Chapter 8

Compilers and IDEs

It's hard to write software that runs correctly and efficiently. So once a program works in one environment, you don't want to repeat much of the effort if you move it to a different compiler or processor or operating system. ***Ideally, it should need no changes whatsoever.***

Kernighan & Pike - The Practice of Programming.

We understand by **portability** the ability to compile and debug our programs on platforms other than those on which they were written, without having to touch a single line of code. By **platform** we understand the combination of a compiler and a CPU architecture. For example, `v143_x64` refers to Visual Studio 2022 and Intel 64bit. In (Figure 8.1) we see the different steps in the code migration process.

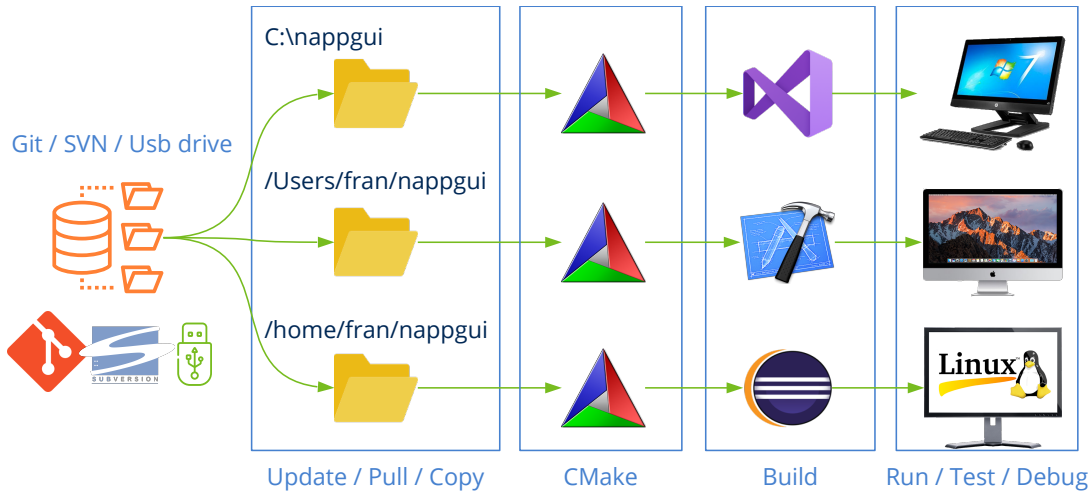


Figure 8.1: Stages in porting code between platforms.

- **Working copy:** A copy of the project's source code must exist on each machine. Normally this will be done through a version control system (SVN, Git, etc).
- **CMake:** will create or update a build project from source code using `/src/CMakeLists.txt` and the scripts in the `/prj` directory. This will be done fully automatically.
- **Compile:** Using Visual Studio, Xcode or GCC, the solution will be compiled and the libraries and executables included in it will be generated.
- **Run/Debug:** The binaries can now be run and debugged on the target platform.

8.1. Windows compilers

We can use any version of Visual Studio from 2005 to compile under Windows (Table 8.1). As we already saw in *"Quick start"* (page 5), the first thing we have to do is launch CMake on the source code:

	Compiler	Platform	Minimum O.S.
	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

Table 8.1: Versions of Visual Studio supported by NAppGUI.

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

- `-G` is the version of the compiler (or generator in CMake jargon).

```
-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` is Intel 32 or 64 bit architecture:

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

- `-T` is the *Platform Toolset*. If you omit this parameter, the last one supported by the compiler will be taken.

```
-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`: Path where the `CMakeLists.txt` is located. Usually in the `/src` directory of the SDK.
- `-B`: Path where the build projects, binaries and temporary files will be generated.

Support for Visual Studio 8 2005 was removed in CMake 3.12. You must use an older version of CMake if you are still using VS2005. NAppGUI does NOT work with versions prior to VS2005.

NAppGUI does not offer support for non-x86, x64 architectures on Windows: ARM, Itanium, etc.

After running CMake, a VisualStudio solution will appear in the `/build` folder, `NAppGUI.sln` or whatever name is configured in `project(NAppGUI)` of the `CMakeLists.txt`. Open that solution and from Visual Studio, `Build->Build Solution` to compile `Debug->Start Debugging` to debug (Figure 8.2).

To change the version of Visual Studio, select another builder in CMake ***-G "Visual Studio 15 2017"****, close and reopen the solution.*

8.1.1. Platform toolset

Starting with Visual Studio 2010, there is a decoupling between the editor and the compiler. The term *Plaform Toolset* identifies the compiler itself, which can continue to be used with more modern IDEs. If we do not specify anything, CMake will use the default toolset included in each version of VS, but it can be changed using the `-T` parameter of CMake (Table 8.2). For example, we can combine `Visual Studio 15 2017` with the VS2013 toolset for Windows XP `v120_xp`:

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

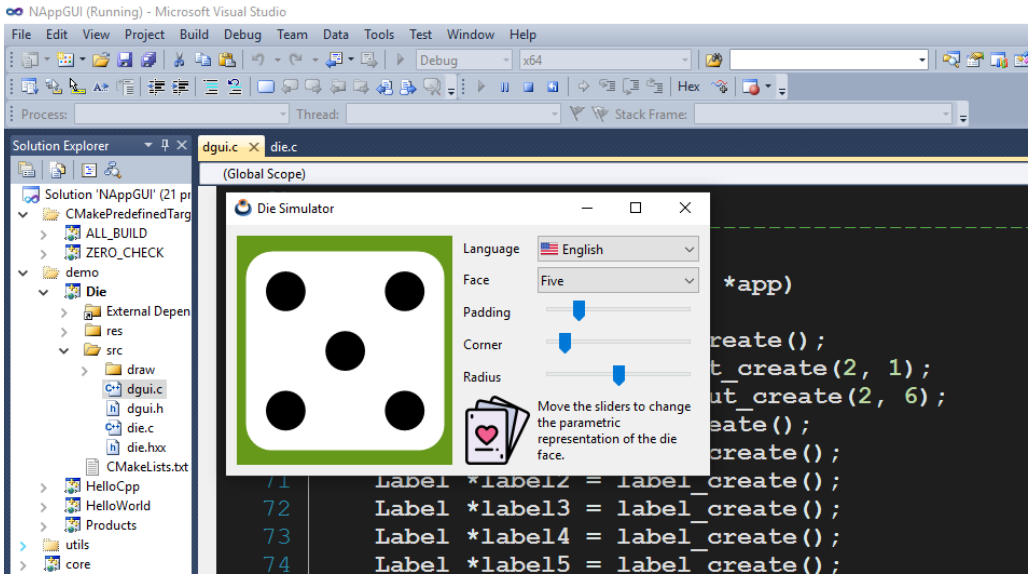


Figure 8.2: Debugging application *Die* in Visual Studio 2010.

Toolset (-T)	VS version
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

Toolset (-T)	VS version

Table 8.2: Toolset included in every version of VS.

*You need to have each version of Visual Studio installed to use its toolset. There are "light" versions that install the **build tools** without the development environment.*

8.1.2. Visual C++ Redistributable

By default, Visual Studio dynamically links the functions of the C standard library, which means that the `.exe` may not work on machines that do not have the `VC++ DLLs` (Figure 8.3). This forces applications to include a copy of `MSVCRT.dll`, `VCRUNTIME.dll` , ... or to install the famous *Visual C++ Redistributable* packages. to ensure that the application can run smoothly.

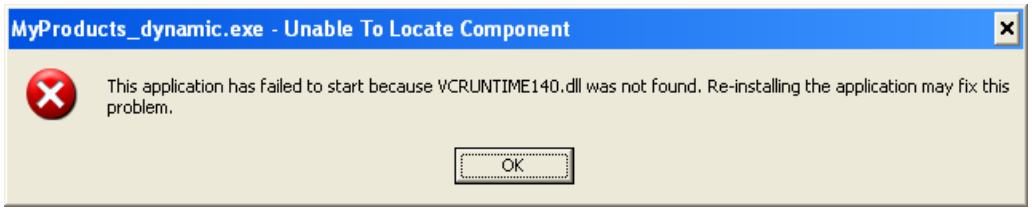


Figure 8.3: Error due to missing VC++ .dll.

NAppGUI uses a reduced set of the C library, since it directly accesses the Windows API whenever possible. For this reason, all applications created with NAppGUI perform a static link (option `/MT`) of the necessary functions of the stdlib, avoiding dependencies at the cost of slightly increasing (a few Kb) the size of the executable. final. This ensures that applications will run smoothly on all Windows machines without the need for additional DLLs and without having to install the *VC++ Redistributable*.

NAppGUI applications do not require the Visual C++ Redistributable. They also do not use the MFC "Microsoft Foundation Classes" or the .NET platform.

8.1.3. WindowsXP support

Starting with VS2012, the *Platform Toolset* generates executables that are not compatible with WindowsXP. If we want our applications to run on this system, we must select the alternative toolset ending in `_xp`: `v141_xp`, `v140_xp`, `v120_xp`, `v110_xp`. Or `v100`, `v90` or `v80` (VS2010, 2008, 2005), which do directly support XP (Figure 8.4) .

*WindowsXP support has been permanently removed in Visual Studio 2019. **There is no Platform Toolset v142_xp**.*

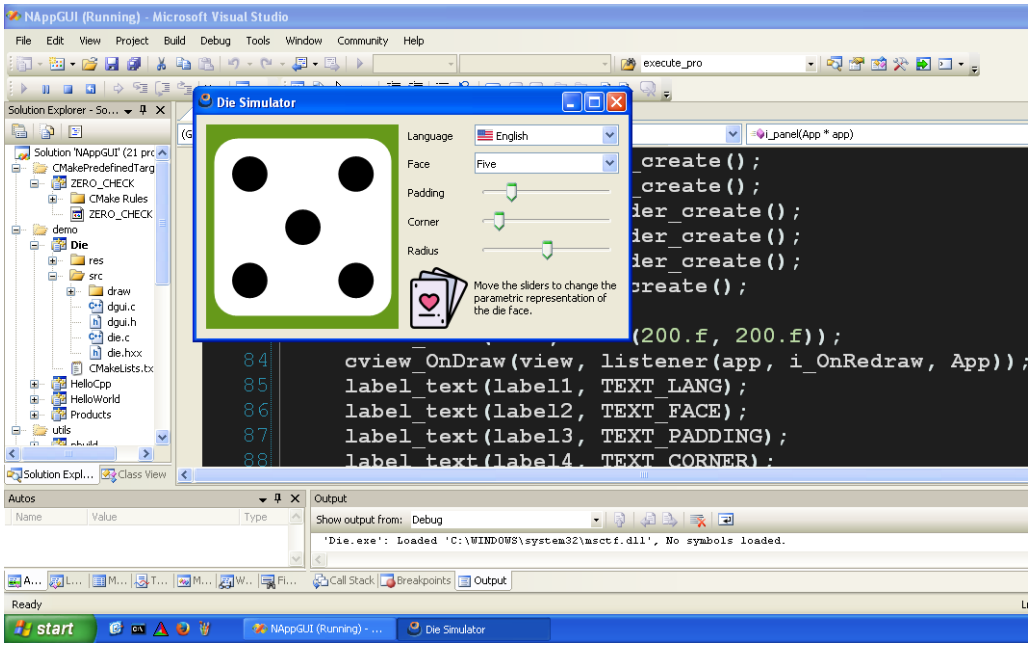


Figure 8.4: Debugging application *Die* on WindowsXP with VS2005 (toolset v80).

Cannot create applications with NAppGUI that work on Windows prior to XP.

8.1.4. SSE support

With the Pentium III, Intel introduced an additional instruction set for floating point operations called SSE *Streaming SIMD Extensions*. This allows you to optimize mathematical calculations at the cost of losing compatibility, since applications that use SSE will not work on Pentium II or earlier models. In NAppGUI the `v80_x86` and `v90_x86` toolsets have been reserved for building applications compatible with older (Table 8.3) processors. Starting with `v100_x86`, SSE2 will be used in all toolsets.

Toolset	SSE	Minimum CPU
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	...

Toolset	SSE	Minimum CPU

Table 8.3: SSE support

SSE support is only disabled on 32-bit (x86) architectures. All 64-bit (x64) CPUs incorporate SSE2.

8.2. macOS compilers

To compile for Apple iMac, macBook and macMini we will need CMake and Xcode[1] starting with version 3.2.6 (Table 8.4). NAppGUI allows you to build applications that work on MacOSX 10.6 Snow Leopard and later:

	Compiler	Minimum O.S.	Platform
	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
	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)

Table 8.4: Xcode versions supported by NAppGUI.

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

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

- `-G` always `"Xcode"`. Use `xcode-select` to toggle if you have multiple versions installed.
- `-DCMAKE_DEPLOYMENT_TARGET`. Minimum operating system that will be supported. Omitting it will set the **Base SDK** included in the Xcode version.

```
-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`. The `arm64` architecture is included starting with SDK 11.0 Big Sur. `i386` was deprecated in macOS 10.13 High Sierra.

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

*NAppGUI does not support the creation of **Apple's Fat binaries**. You must indicate a single value in this field.*

- `-S`: Path where the `CMakeLists.txt` is located. Usually in the `/src` directory of the SDK.
- `-B`: Path where the build projects, binaries and temporary files will be generated.

After running CMake, an Xcode solution will appear in the `/build` folder, `NAppGUI.xcodeproj` or whatever name is configured in `project(NAppGUI)` of the `CMakeLists.txt`. Opening the Xcode solution, we see the different projects that make it up, including *Die* and *Dice*. Select *Die* in the top left dropdown and then click `Play` or `Product->Run`

(Figure 8.5). This will compile the program and launch it in debug mode, where we can set breakpoints to inspect the stack and the values of the variables.

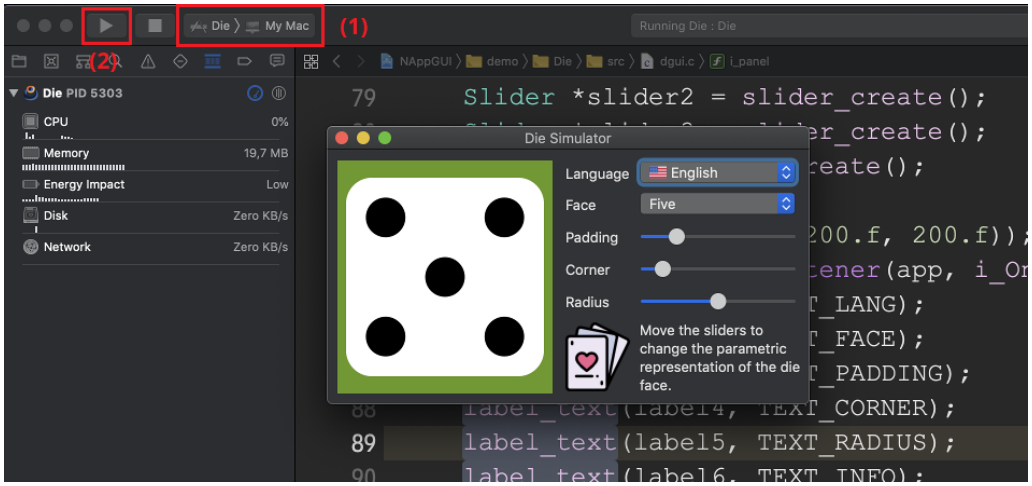


Figure 8.5: Debugging application *Die* in Xcode.

8.2.1. Base SDK and Deployment Target

Every year, Apple releases a new version of macOS, accompanied by a new SDK and an update to Xcode that includes the SDK. This is called the **Base SDK**.

> ***Base SDK*** *is the version included in each new major version of Xcode, which matches the latest version of the macOS system released on the market.*

Apple has a much more restrictive policy than Microsoft regarding the compatibility of applications with previous versions of the operating system. By default, a program compiled with SDK 10.14 (macOS Mojave) will not work on the immediately preceding macOS High Sierra (Figure 8.6).

To avoid this problem, and for applications to work on older macOS, there is the **Deployment Target** parameter. Using it will trigger a macro that will override the new features of the Base SDK. This will allow the program to run on older versions at the cost, of course, of not having access to the latest iMac features. You will be able to select the Deployment Target required by your project through the `-DCMAKE_DEPLOYMENT_TARGET` parameter, as we have already seen in the previous section.

> *Xcode 14 deprecates Deployment Targets below 10.13 (Figure 8.7). Use Xcode 13 if you want compatibility with Mac OSX 10.12 Sierra and earlier.*

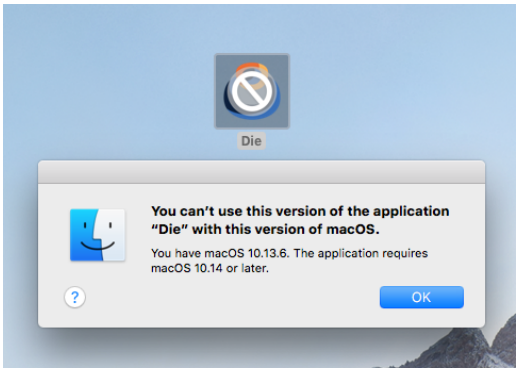


Figure 8.6: *Die* with *Base SDK* 10.14 will not work on High Sierra.

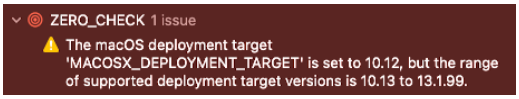


Figure 8.7: Deployment Target 10.12 deprecated as of Xcode 14.

Xcode 8 deprecates Deployment Targets below 10.9 (Figure 8.8). Use Xcode 7 if you want compatibility with Mac OSX 10.8 Mountain Lion and earlier.

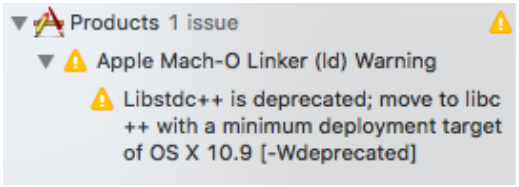


Figure 8.8: Deployment Target 10.8 deprecated as of Xcode 8.

8.2.2. xcode-select

We have already seen that CMake only offers one generator for Xcode (`-G "Xcode"`), although it is possible to have multiple versions installed on the same machine, each within its own *bundle* `Xcode.app`. There will always be a default Xcode on the system (the most recent one) but it can be changed using the `xcode-select` utility:

Query the current version of Xcode.

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

Changing the active version of Xcode.

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

Set the default version of Xcode.

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

*You will need to run **cmake -G "Xcode"...** again each time you use `xcode-select` for your project to update the compiler change.*

8.2.3. macOS ARM

In November 2020 Apple launches its new line of desktop and laptop computers (iMac, macBook and macMini) based on the Apple M1 processor with ARM (Figure 8.9) architecture. Although they are capable of running programs compiled for Intel x64 using the Rosetta 2 (Figure 8.10) program, the ideal would be to compile our applications for the new architecture in order to optimize the executables as much as possible.

Figure 8.9: Procesadores M1 de Apple.

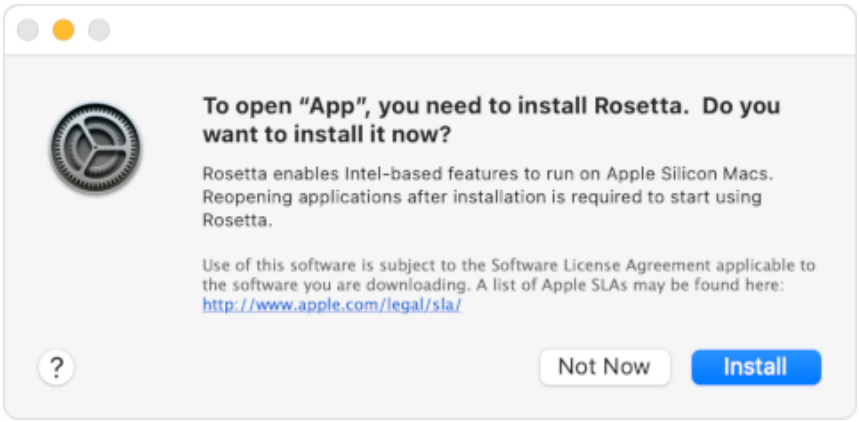


Figure 8.10: Warning Rosetta 2 when trying to run x64 code on an Apple M1.

NAppGUI supports building for the Apple ARM architecture. You just need to include the `-DCMAKE_ARCHITECTURE="arm64"` option in CMake, as we saw in the previous section.

You can compile the M1 architecture from Intel x64 machines, but you won't be able to debug the executables.

M1 architecture is only available for Big Sur system (macOS 11.0) and later.

8.2.4. macOS 32bits

Since the macOS High Sierra release, Apple has declared the 32-bit architecture obsolete[2], issuing notices to users in the case of detecting `i386` (Figure 8.11) executables. As of Xcode 10, (Figure 8.12) cannot be compiled on this architecture.

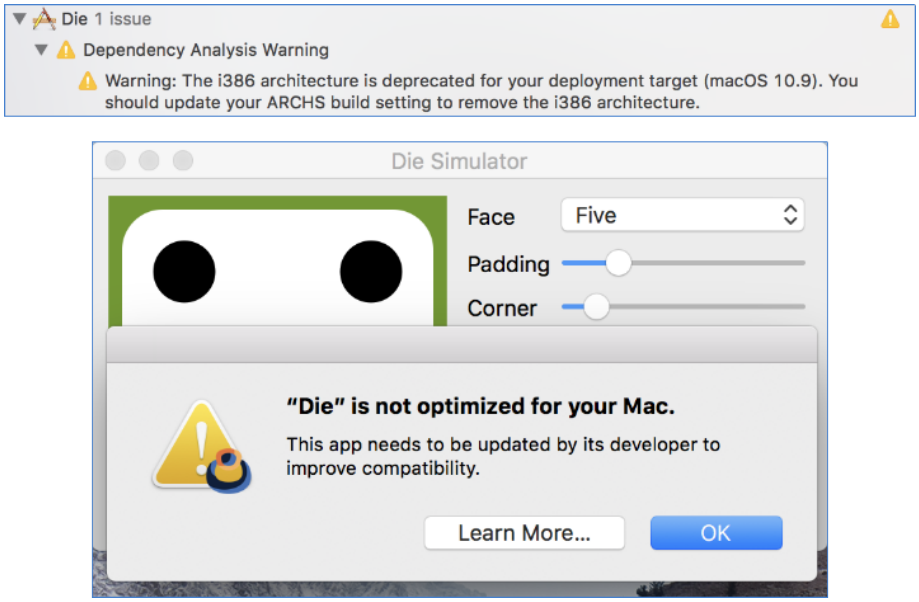


Figure 8.11: macOS warnings in 32bit applications.

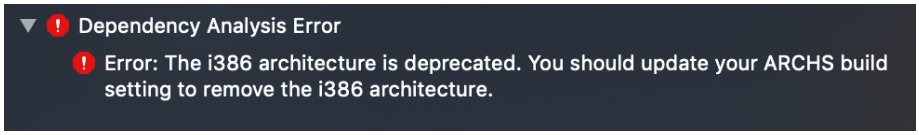


Figure 8.12: Xcode 10 error when trying to compile in 32bit.

*Support for 32bit applications is gone for good in **macOS Catalina**, which only allows running 64bit applications.*

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

This makes some sense since all Intel-based iMac models feature 64-bit processors, except for a few 2006 models in white polycarbonate that mounted the 32-bit Intel Core Duo (Figure 8.13) . These iMacs supported Mac OSX 10.6 Snow Leopard at most, with a 64-bit CPU being a fundamental requirement as of 10.7 Lion. To compile without problems in 32bits you must use, at most, Xcode 6 (Figure 8.14).

Figure 8.13: Only Apple models with Intel 32bit processor.

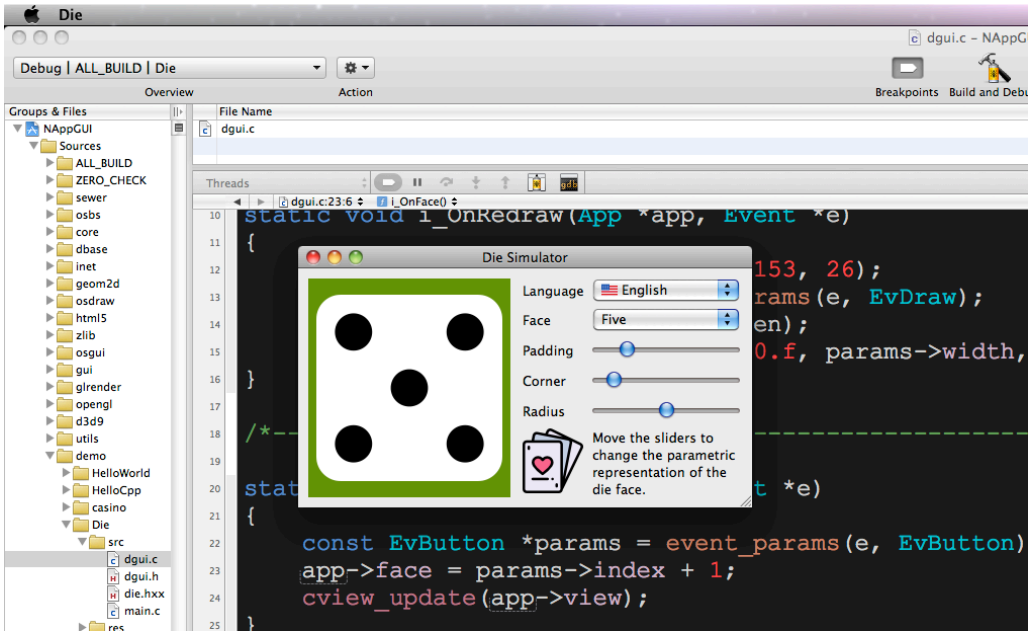


Figure 8.14: Compilación en 32bits con Xcode 3.2.6 (Snow Leopard).

8.3. Linux compilers

For Linux versions, we will use the `gcc` (Table 8.5) compiler and the `make` tool to generate the binaries, but there is no development environment " official" as it happens in Windows and macOS. To carry out an elementary configuration of our equipment, type

the following commands in a 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
```

	Minimum O.S.	Compiler	Toolkit	Platform
	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

Table 8.5: GCC versions supported by NAppGUI.

Just like we did on Windows and macOS, we run `cmake` to generate the build project:

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

- `-G` always `"Unix Makefiles"`. Additionally, you can create projects for the main IDEs available in 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`. Unlike Visual Studio and Xcode, Make does not support the creation of multi-configuration projects. It must be indicated at the time of generation:

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

- `-DCMAKE_ARCHITECTURE`. `x64`, `i386`, `arm`, `arm64`. Cross-compiling is not allowed on Linux. We must select the same architecture as the host machine. This parameter can be omitted, it will be set automatically.

```
-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`. As of today, the only option available is `"GTK3"`, since NAppGUI does not support other graphical toolkits. This parameter can be omitted, it will be set automatically.

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

- `-S`: Path where the `CMakeLists.txt` is located. Usually in the `/src` directory of the SDK.
- `-B`: Path where the build projects, binaries and temporary files will be generated.

After executing `cmake` we will have, in the `/build` folder, a series of `Makefiles` ready to compile the project.

```
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
```

Once the compilation is finished, we can launch the executables directly from the terminal:

Launch application *Die*.

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

If you're fairly comfortable with `gdb`, you can try debugging the code directly from the (Figure 8.15) terminal. Later we will see how to do it using Eclipse and Visual Studio Code.

Debugging *Die* with *gdb*

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

8.3.1. GTK+3

Unlike Windows and macOS, Linux supports a multitude of desktop environments based on different libraries (or *toolkits*), GTK and Qt being the two most famous. NAppGUI uses GTK+3 for the graphical part since it is the base of the Gnome, Xfce, Lxde, etc, (Table 8.6) environments present in many of the most widespread distributions. GTK+3 will be present naturally in all of them, with no other additional dependencies being necessary. Of course, to compile under GTK+3 we will have to install the developer version, as we saw at the beginning of this section.

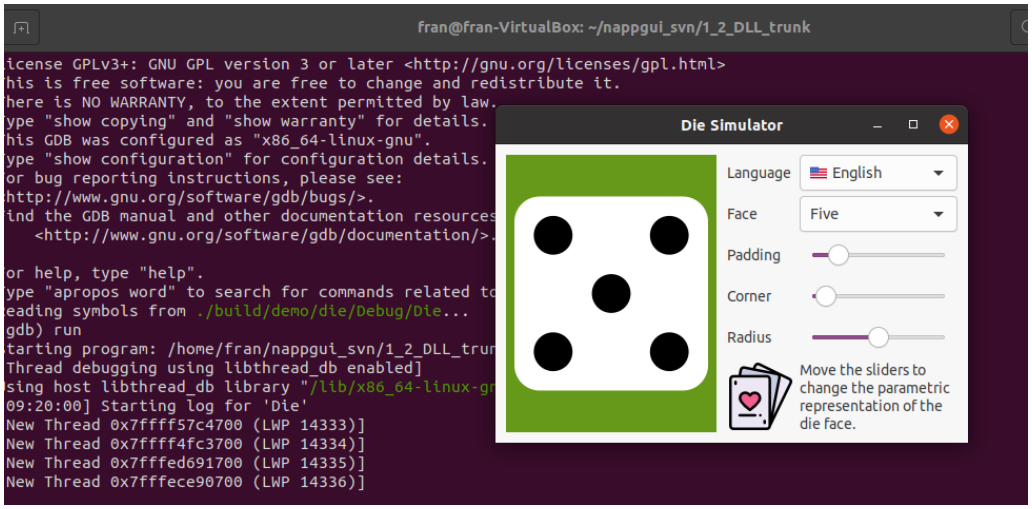


Figure 8.15: Debugging *Die* with GDB from the terminal.

	Environment	Distributions
	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	

Table 8.6: Gtk-based desktop environments.

8.3.2. Multiple versions of GCC

Although every Linux distribution comes with a "canonical" version of GCC, it is possible to have several installed on the same machine and switch between them much like we did on macOS with `xcode-select`. To do this we will use the Linux `update-alternatives` command. We assume that we are on Ubuntu 18.04 LTS:

Version of gcc installed.

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

Install gcc-6

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

Register gcc-7 and 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
```

Switch to gcc-6.

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

Return to the default version of gcc.

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

8.3.3. Linux 32bits

To compile 32bit applications from a 64bit Ubuntu system it is necessary to install the `multilib` package:

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

But there are currently problems[3] with cross-compiling that includes the GTK+ library, so it won't be possible to use the same machine development to build on both architectures, just like it does on Windows. Console applications or libraries that do not access GTK can be compiled in 32bit from a 64bit computer.

It is not possible to compile in 32bits from a 64bit Ubuntu system applications that use GTK+3. You must use a 32-bit Linux system for this.

[3] `https://ubuntuforums.org/showthread.php?t=2038875`

8.3.4. Linux ARM

The ARM [4] *Advanced RISC Machine* architecture is the predominant one in the market for embedded devices such as smartphones and tablets. Currently, NAppGUI does not offer support for the development of iOS/Android mobile applications, but it does support other types of boards that support "desktop" versions of Linux ARM, such as the Raspberry PI. To port our code to the Raspberry Pi we must follow the same steps as in Ubuntu Linux (Figure 8.16). Both distributions are based on Debian, so GCC, CMake and Make are available directly via `apt-get`.

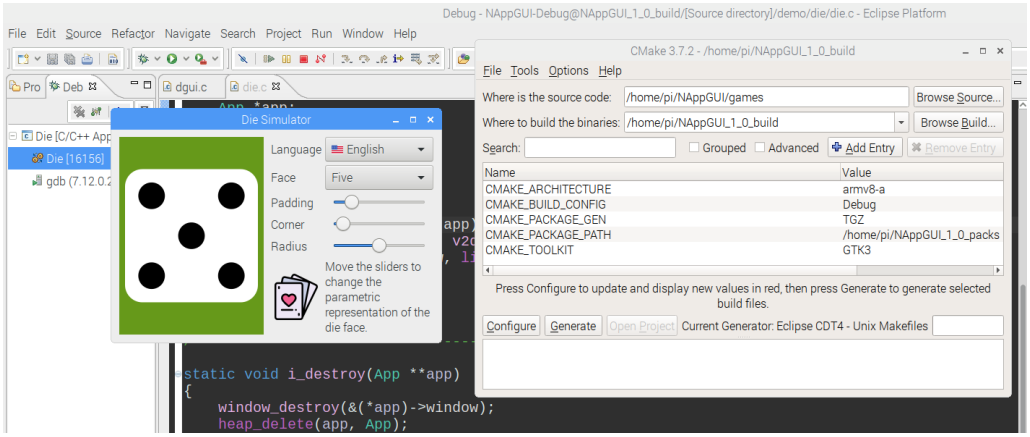


Figure 8.16: Debugging the application *Die* on a Raspberry Pi.

8.3.5. Eclipse CDT

Working directly with the terminal gives us great flexibility when configuring our own tools. Going back to the console and typing `cmake --build ./build -j 4` will recompile everything you need. However, using `GDB` directly will be quite tedious, so the use of an integrated debugger (or IDE) becomes almost essential. For the development of NAppGUI we intensively use Eclipse CDT[5]. This environment will allow us to program with a methodology similar to that of Visual Studio and Xcode: Set breakpoints, inspect the stack and variables, search for files within the code directory, multiple edits, massive searches, etc.

The only difference is that we will have to use the `-G "Eclipse CDT4 - Unix Makefiles "` generator in CMake which, in addition to the `Makefile`, will create the `.cproject` and `.project` required to import the project into Eclipse.

Open Eclipse and do `File->Import->Existing Projects into Workspace`. A di-

[4] `https://en.wikipedia.org/wiki/ARM_architecture`
[5] `https://www.eclipse.org/cdt/`

alog box will appear where we indicate the *build* directory that we have configured in CMake (`/build`). Eclipse will open the project, placing a tree with all the files on the left and we will compile with `Project->Build All`. When debugging (Die in this case) we will create a profile from `Run->Debug Configurations->C/C++ Application`. Click `[Search Project...]` and select *Die* from the dropdown list. Finally we press `[Debug]` to debug the application interactively (Figure 8.17).

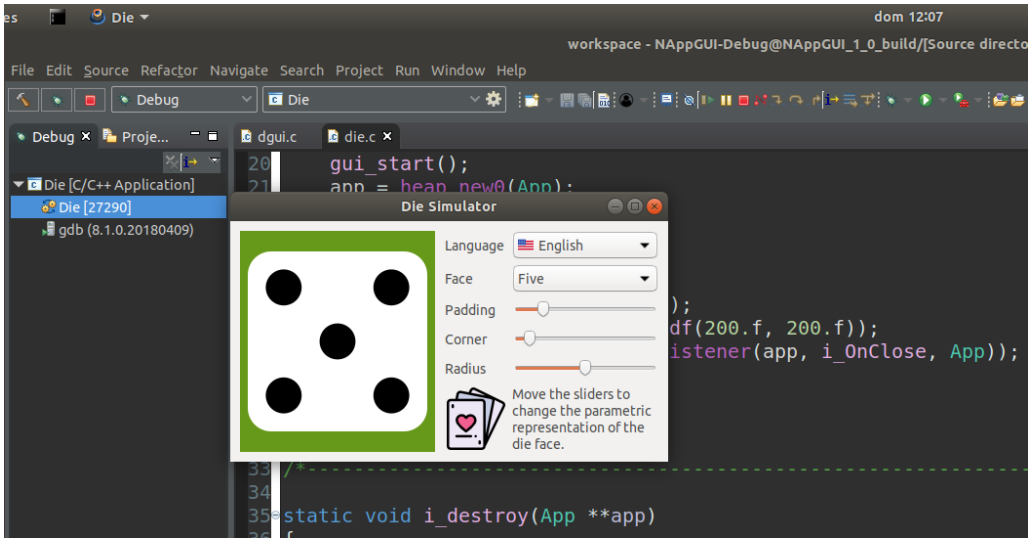


Figure 8.17: Debugging application *Die* with Eclipse.

Some interesting Eclipse CDT options under `Window->Preferences`.

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

Using Eclipse is only a recommendation. You have total freedom to use the tools that you consider best.

8.3.6. Visual Studio Code

Another interesting environment to develop on Linux is Visual Studio Code. With the appropriate extensions, it is possible to work in C/C++ with CMake in a very comfortable and fluid way. To install it:

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

We added, at a minimum, the **C/C++ Extension Pack** which will also include support for CMake (Figure 8.18).

We open our project with `Open Folder`. Later, we run CMake from the environment

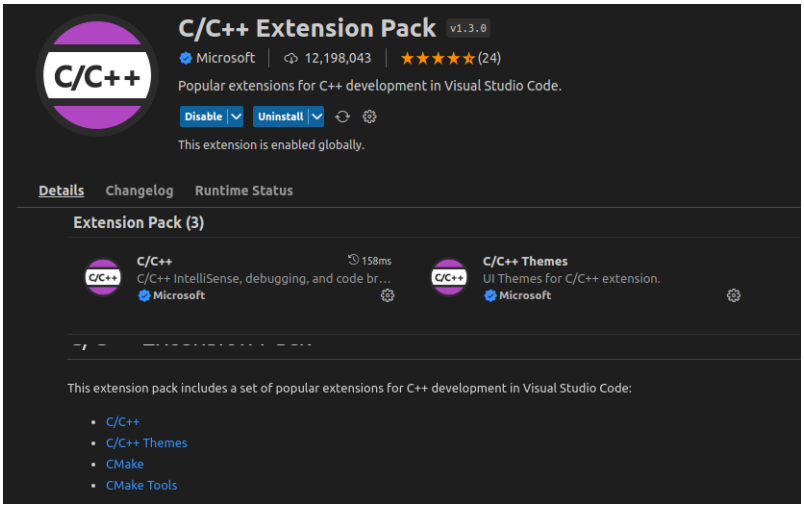


Figure 8.18: C/C++ Extension Pack.

itself: `[F1]->CMake:Configure`. The first time, VSCode will ask for the location of the `CMakeLists.txt` main (Figure 8.19) (`/src/CMakeLists.txt`).

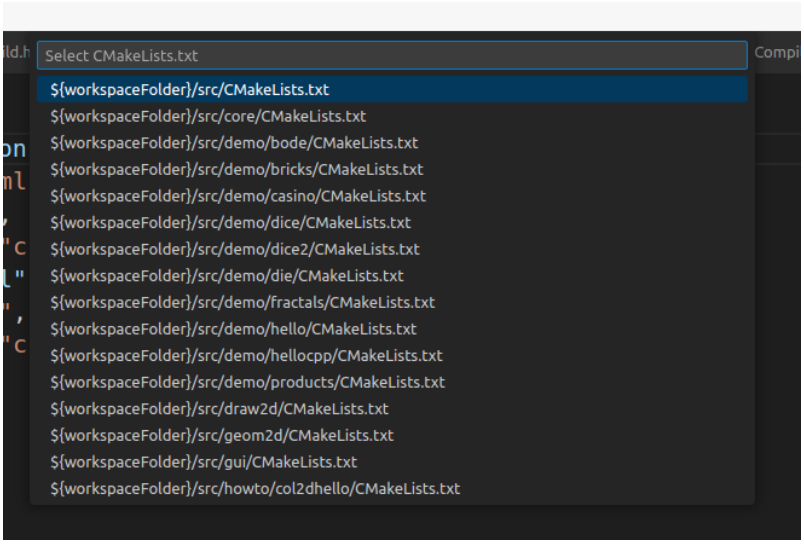


Figure 8.19: Selection of the main CMakeLists.txt of the project.

After the configuration we can compile with `[F1]->CMake:Build`. In the **Output** tab of VSCode we will see the evolution of the process:

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

To debug, the first thing is to select the target (or executable) with `[F1]->CMake:Set Debug Target` (Figure 8.20).

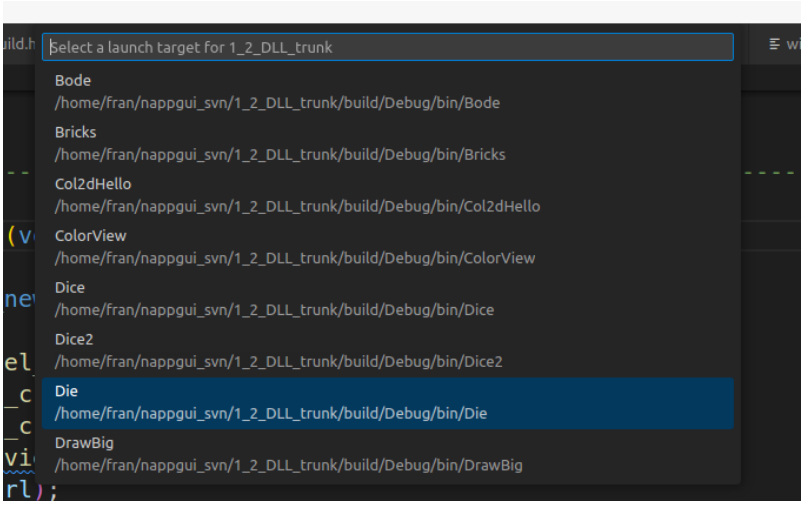


Figure 8.20: Selection of the executable to debug.

And we launch the debugger with `[F1]->CMake:Debug` (Figure 8.21).

8.4. Configurations

A NAppGUI application can be compiled in three different configurations, depending on the level of debugging we need.

- *Debug*: Includes debugging information in the binaries and does not perform code optimizations. It is the developer version.
- *Release*: Remove debug information and perform all possible optimizations. It is the version for the user.
- *ReleaseWithAssert*: It is the Release version, but leaving the *"Asserts"* (page 153) statements active. It is aimed at the end user, but in cases where it is necessary to

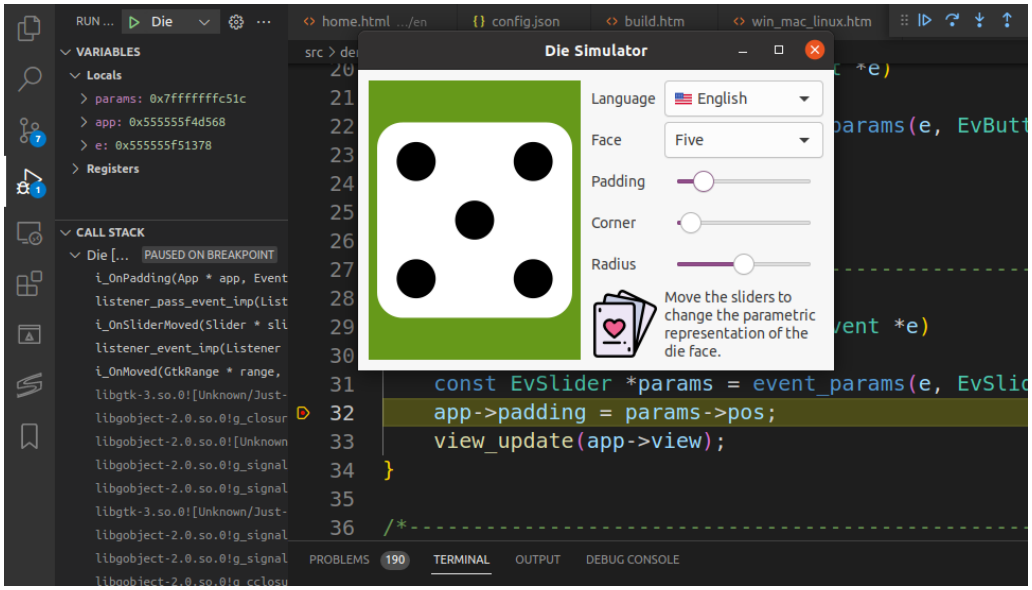


Figure 8.21: Debugging application *Die* from Visual Studio Code.

obtain detailed information on possible anomalies, at the cost of a decrease in the overall performance of the program.

Both Visual Studio and Xcode are multi-configuration environments, that is, we can switch between one and the other directly from the editor itself. In Visual Studio we have a dropdown at the top of the (Figure 8.22) editor.

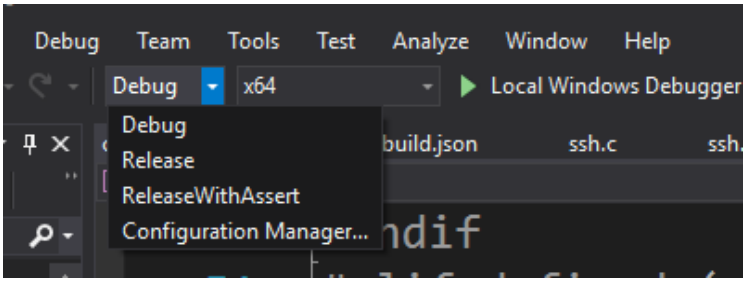


Figure 8.22: Config change in Visual Studio.

In Xcode it is a bit more hidden. We do `Product->Scheme->Edit Scheme`. A popup window will appear. Select `Run->Info->Build Configuration` (Figure 8.23).

Unfortunately, `Unix make` does not support multiple configurations. This forces us to pass the `CMAKE_BUILD_CONFIG` (Figure 8.24) property to set the configuration in CMake before building the `Makefiles`. We must re-run `cmake -S ./src -B ./build` if we change the configuration, for the new configuration to take effect.

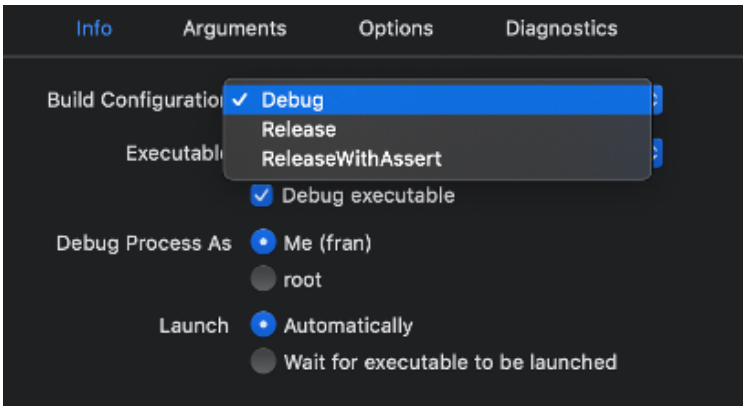


Figure 8.23: Config change in Xcode.

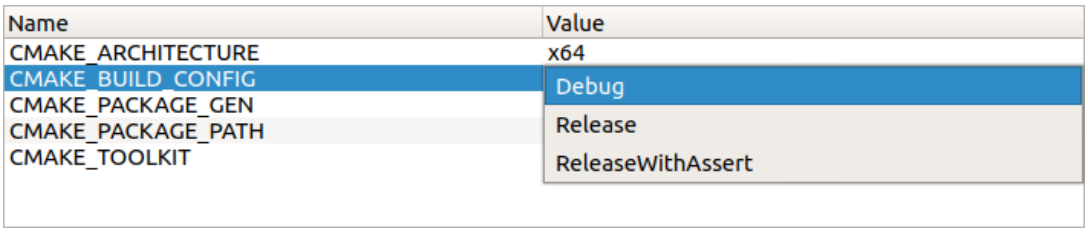


Figure 8.24: Change configuration in CMake (Unix Makefile).

Chapter 9

Create new application

I consider myself a technical person who chose a great project and an excellent way to carry it out.

Linus Torvalds.

In *"Quick start"* (page 5) and *"Generate NAppGUI binaries"* (page 65) we have seen how to obtain the SDK, as well as compile and run the sample applications. Also, in *"Hello World!"* (page 23), we learned the basic structure of an application based on NAppGUI. The time has come to create our own applications, taking advantage of the CMake scripts included in the `/prj` folder of the distribution.

> *If your goal is to use NAppGUI as an external library in your projects, you can skip this chapter.*

9.1. Desktop applications

To create a new desktop application, open the file `/src/CMakeLists.txt` and add the following line after `# Your projects here`!:

src/CMakeLists.txt

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

Then, rebuild the solution with CMake and open it with the corresponding IDE:

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

> *The **cmake –open** command only works with the Visual Studio and Xcode generators. In Linux you will have to open it manually with the editor of your choice.*

In case the solution was already open, it is possible that the IDE warns you that there have been changes, for example Visual Studio (Figure 9.1).

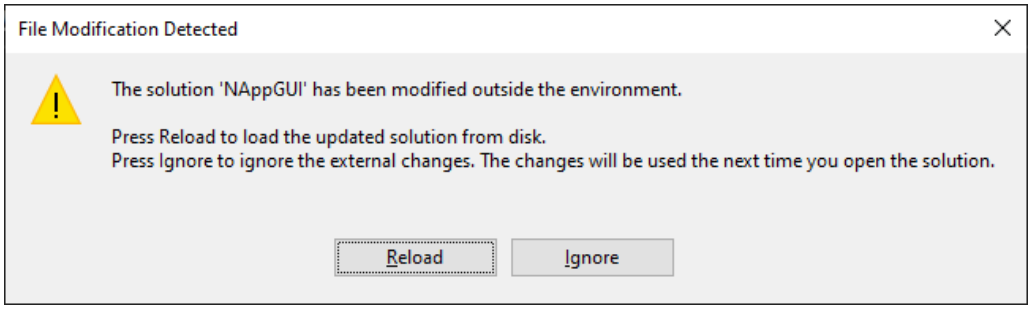


Figure 9.1: Warning Visual Studio to reload the solution.

You will see that CMake has created a new project called `MyNewApp` inside the (Figure 9.2) solution.

If you compile and run `MyNewApp`, you'll notice that it's nothing but the *Hello, World!* (Figure 9.3), since this is the default template for each new desktop application. An *"Application iconApplication icon"* (page 140) has also been assigned by default, but we can customize it later.

Looking in detail at the syntax of the `desktopApp` command that we just added to the `CMakeLists.txt`, we have:

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

desktopApp(appName path depends nrcMode)
```

- `appName`: The name of the application.
- `path`: Path relative to `/src` where the project will be located (in this case `nappgui_src/src/myapp`). Any path depth is supported. For example, `"games/myapp"` will create the project in `nappgui_src/src/games/myapp` and `"demo/games/myapp"` in `nappgui_src/src/demo/games/myapp`.
- `depends`: Libraries on which the application depends. At a minimum you will have to include `osapp` since this is a desktop application. If the application needs additional

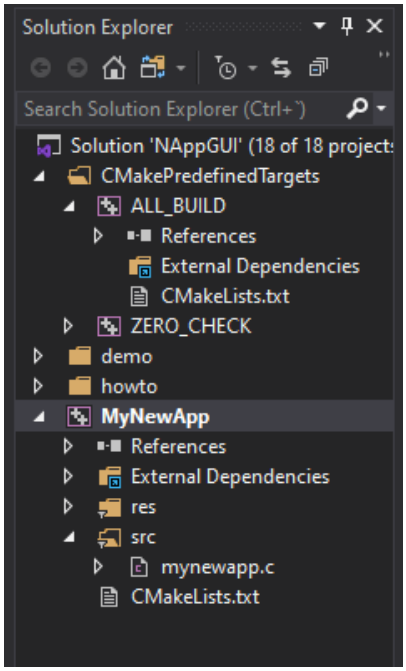


Figure 9.2: Project `MyNewApp` just added to the solution.

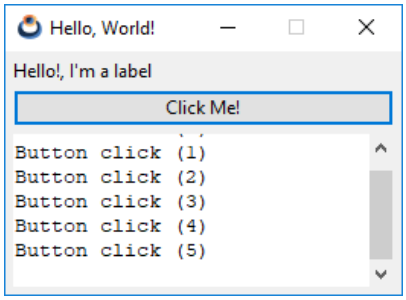


Figure 9.3: Result of compiling and running `MyNewApp`.

dependencies, such as self-created libraries, we will write them below separated by semicolons (eg `"osapp;physics;render"`). In *"Create new library"* (page 107) you have an example of applications with dependencies. **Important:** You only need to indicate the first level dependencies. CMake will recursively add the dependencies of the dependencies.

- `nrcMode`: How the application's resources will be managed. For now, we specify `NRC_EMBEDDED`. We'll go deeper into them in the *"Resources"* (page 129) chapter.

You can create as many applications within the same solution as you want. You just have to repeat the process, adding new `desktopApp()` to the `CMakeLists.txt` script.

9.2. Adding files

Going back to the `MyNewApp` project, we see that by default only one source code file (`mynewapp.c`) is created that contains the entire application. It is very likely that you want to split the code between different files. Create a pair of new files `myfunc.c` and `myfunc.h` inside `nappgui_src/src/myapp` from the IDE or directly from the browser. Open them and add these lines:

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;
}
```

Open `mynewapp.c` and edit the function `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);
}
...
```

Rebuild the solution with `cmake -S ./src -B ./build`. The IDE, Visual Studio in this case, informs us again that there have been changes in the `MyNewApp` (Figure 9.4) project. Just press `[Reload All]`.

Recompile and run `MyNewApp` to see the changes you just made. You can create as many files and subfolders inside the `src/myapp` directory as you need to better organize your code. Always remember to run `cmake -S ./src -B ./build` whenever you add or

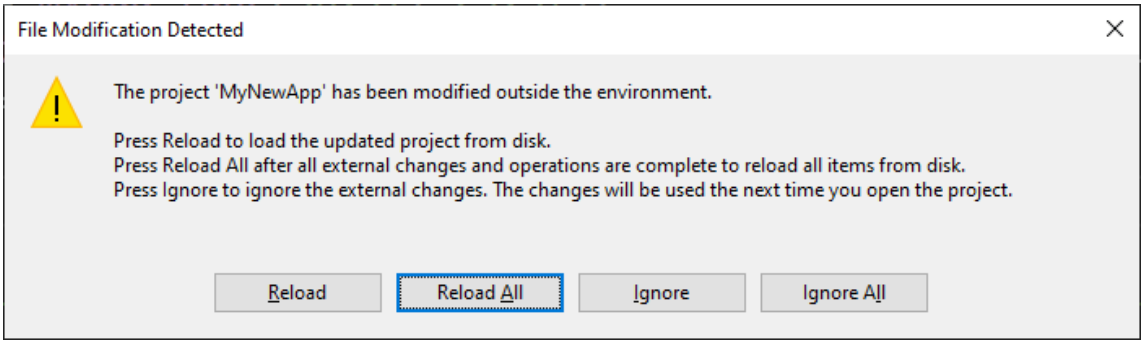


Figure 9.4: Visual Studio warns about new files. Press `[Reload All]`.

remove files from the project. CMake will update the solution by "cloning" the directory structure within the project (`MyNewApp` in this case).

> *At this point we recommend that you spend some time researching, compiling, and testing the examples in the `demo` and `howto` folders.*

9.3. Command line applications

Similar to the desktop apps seen above, you will be able to create console apps. Open `/src/CMakeLists.txt` and add this line after `# Your projects here`!:

src/CMakeLists.txt

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

When regenerating the solution with `cmake -S ./src -B ./build`, Visual Studio will prompt you to reload the solution, just like our previous application did. A new project will have been created in `nappgui_src/src/utils/myutil`, but this time if you compile and run it no window will appear. You will only see a message in the Visual Studio console:

```
Hello world!
```

If you open `myutil.c` you will find the code that generated the above output:

```
/* NAppGUI Console Application */

#include "coreall.h"

int main(int argc, char *argv[])
{
    unref(argc);
    unref(argv);
```

```
    core_start();
    bstd_printf("Hello world!\n");
    core_finish();
    return 0;
}
```

Which is the typical template of a C program, to which the support of the *core* library has been included. From here, we can already modify the code and compile. CMake has already configured everything for us. Let's go back to the `src/CMakeLists.txt` to review the previous line:

src/CMakeLists.txt

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

commandApp(appName path depends nrcMode)
```

- `appName`: The name of the application.
- `path`: Path relative to `/src` where the project will be located (in this case `nappgui_src/src/utils/myutil`).
- `depends`: Dependencies. A command line application does not require any "minimal" dependencies, as desktop applications do. We recommend including a dependency on *"Core"* (page 187) ("core") as it contains a variety of functions that can make our task easier. However, you can set *"Osbs"* (page 166) ("osbs") or even *"Sewer"* (page 149) ("sewer") as minimum requirements.
- `nrcMode`: `NRC_NONE`, `NRC_EMBEDDED` or `NRC_PACKED`. *"Resource distributionResource distribution"* (page 137).

It goes without saying that we can add new files and subfolders to the project in a similar way as we did in desktop applications.

9.4. C/C++ Standard

NAppGUI has been created, almost entirely, in C90 language, adding the fixed type integers `uint32_t`, `int16_t`, `...` (`<stdint.h>`) of C99. For certain parts of the project, C++98 has been used, but always encapsulated under a C90 interface. Therefore, **an application or library can be created using only C90**, which provides great portability between platforms and compilers.

In general, compilers allow you to check that your code conforms to certain C/C++ standards, issuing warnings or errors when it doesn't. By default, each new project created with `desktopApp()` or `commandApp()` will set `C90` and `C++98` as standards. This way,

they will be compatible with the entire list of *"Compilers and IDEs"* (page 73) supported by NAppGUI.

However, you may want to use a more modern standard for your new projects. In this case, you must indicate it at the end of the `desktopApp` or `commandApp` commands:

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

In this case we will indicate that the application `MyNewApp` will use the `C17` and `C++17` standards instead of `C90` and `C++98`, which are the default values.

- For the C compiler, the options will be: `C90`, `C99`, `C11`, `C17`, and `C23`.
- For the C++ compiler, the options will be: `C++98`, `C++11`, `C++14`, `C++17`, `C++20` and `C++23`.

If the compiler does not support the indicated language version, the highest supported one will be set. It is the programmer's responsibility to use the compilers appropriate to the chosen standard.

Chapter 10

Create new library

The only thing that you absolutely have to know, is the location of the library.

Albert Einstein

The use of libraries will allow us to share common code between several projects. Take the NAppGUI SDK for example, which has been organized into various static or dynamic link libraries. For example *"Core"* (page 187) implements functions related to strings, streams and data structures that can be reused in different applications.

10.1. Static libraries

To illustrate the use of libraries, we will use two applications included in the NAppGUI examples: `Die` (Figure 10.1) and `Dice` (Figure 10.2). In both you must be able to draw the silhouette of a dice.

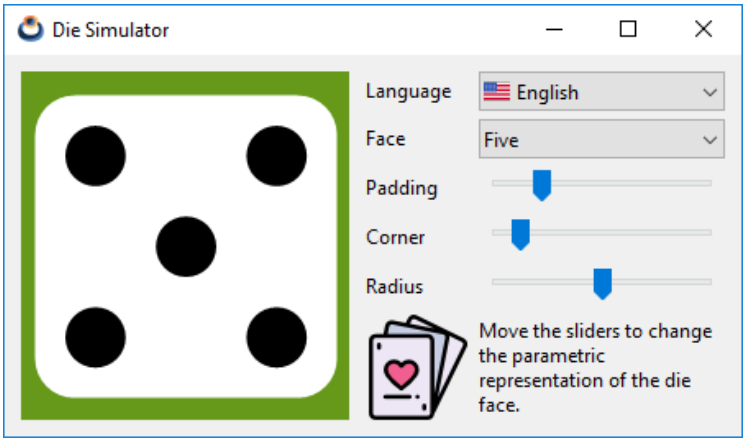


Figure 10.1: Aplicación *Die*.

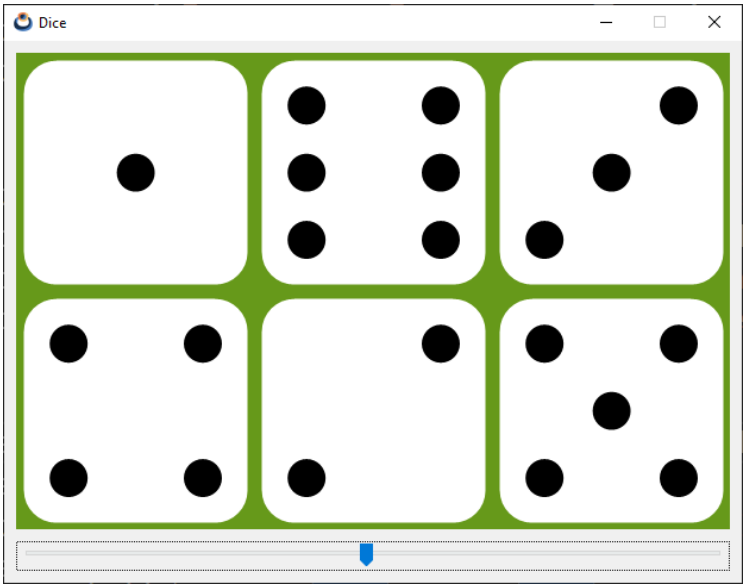


Figure 10.2: Application *Dice*.

The source code for both applications is available at src/demo/die[a] *and 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

It is not very difficult to intuit that we could reuse the parametric drawing routine in both projects. One way to do this would be to copy the routine from *Die* to *Dice*, but this is not recommended as we would have two versions of the same code to maintain.

Another option, the most sensible, is to move the drawing function to a library and link it in both applications. This is very easy to do thanks, again, to CMake. If we open the `src/CMakeLists.txt` we will see these three lines:

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

Where we have used the `staticLib()` command, which is analogous to `desktopApp()`.

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

- `libName`: The name of the library.
- `path`: Path relative to `/src` where the project will be located (in this case `nappgui_src /src/demo/casino`). Just like we saw when creating new apps, any path depth is supported.
- `depends`: Library dependencies. As in applications, it is only necessary to indicate the highest level ones (*draw2d* in this case). Each library is responsible for linking with the ones below it. *draw2d* will include *geom2d* and so on. In *"NAppGUI API"* (page 145) you have the complete dependency graph.
- `nrcMode`: How the library's resources will be managed. For now, we specify `NRC_EMBEDDED` . We'll go deeper into them in the *"Resources"* (page 129) chapter.
- `standard`: Optionally, you can indicate the *"C/C++ StandardC/C++ Standard"* (page 104).

Both *Die* and *Dice* have added a dependency on *casino* (Figure 10.3) via the `depends` parameter of the `desktopApp()` command. In this way, CMake knows that it must link, in addition to *osapp*, the *casino* library, which is where the common code of both projects is found.

Rebuilding with `cmake -S ./src -B ./build` adds the *casino* library to our solution, as well as a link to it in both (Figure 10.4) applications.

As it happened when creating a new application, when a library is created, several files appear by default, which are:

`casino.def`: File that will define the `_casino_api` macro needed to export symbols. More information in *"Symbols and visibilitySymbols and visibility"* (page 124).

Listing 10.1: demo/casino/casino.def

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

#if defined(NAPPGUI_SHARED)
    #if defined(NAPPGUI_BUILD_CASINO_LIB)
```

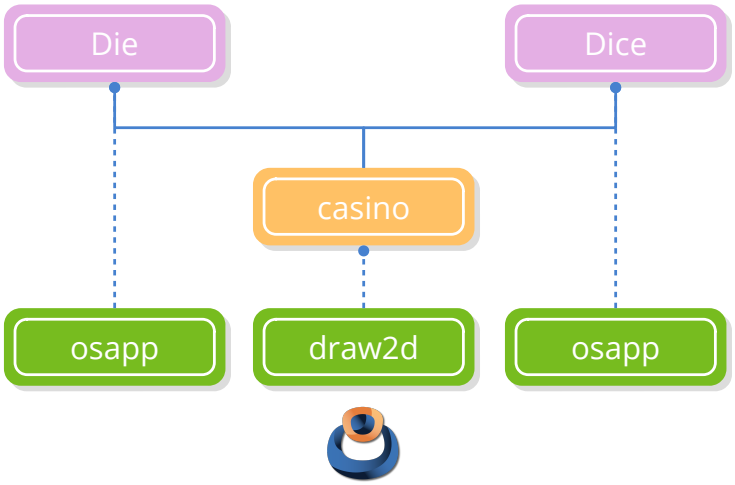


Figure 10.3: Application dependency tree, centered on the *casino* library.

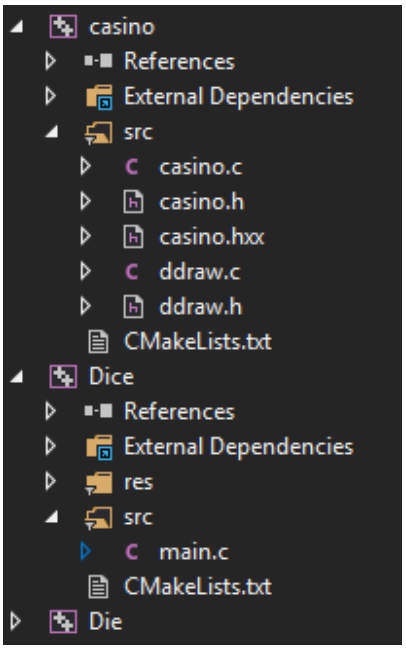


Figure 10.4: Static *casino* library, integrated into the solution.

```
        #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`: Here we will define public types, such as `enum` or `struct`. At the moment `casino` does not contain public types.

Listing 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`: Header file. Here we will write the declaration of general functions. By default, CMake creates two: `casino_start()` and `casino_finish()`, where we would implement global library start and end code, if necessary.

Listing 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`: Implementation of general functions.

Listing 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 */
}
```

Later we create two new files inside `src/demo/casino`, `ddraw.c` and `ddraw.h` where we will implement the drawing function to to share. We already saw how to *"Adding filesAdding files"* (page 102).

Listing 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;
```

Listing 10.6: 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);
    }
}
```

What does it really mean that *Die* and *Dice* have a dependency on *casino*? That from now on none of them can be compiled if there is an error in the *casino* code, since it is a fundamental module for both. Within the build project (Visual Studio, Xcode, Makefile, etc) several things have happened:

- Both applications know where *casino* is located, so they can do `#include "casino.h"` without worrying about its location.
- The binary code of the *casino* functions will be included in each executable in the linking process. CMake has already taken care of linking the library with the executables.
- Any changes made to *casino* will force the applications to be recompiled due to the previous point. Again, the build project will know how to do it in the most efficient way possible. We just have to run `cmake --build ./build` again to update all the binaries.

As we noted before, *casino* also has a dependency on *"Draw2D"* (page 256), NAppGUI's vector drawing library. In turn *draw2d* depends on *geom2d* and so on, up to *sewer*, the lowest package of the SDK. When you develop a new library you should link it with as few dependencies as possible, or, in other words, with the lowest level libraries within the hierarchy that include the necessary functionality. This will improve compilation and distribution, as well as being a very good working practice.

10.2. Dynamic libraries

Dynamic libraries are essentially the same as static libraries. The only thing that changes is the way they link to the (Figure 10.5) executable. In the static link, the library code is added to the executable itself, so the size of the latter will grow. In dynamic linking the library code is distributed in its own file (`.dll`, `.so`, `.dylib`) and is loaded just before the executable program.

The process to create dynamic libraries is exactly the same as the static ones. All we need to do is replace the `staticLib()` command with `dynamicLib()` in `/src/CMakeLists.txt`.

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

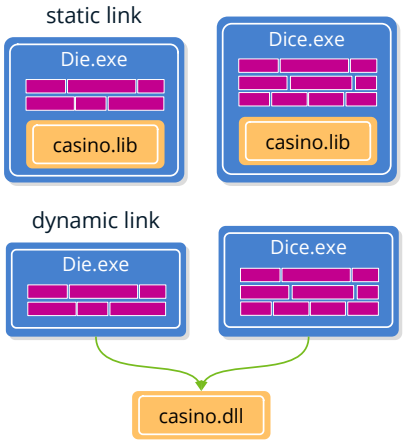


Figure 10.5: Static or dynamic *casino* link.

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

The parameters are exactly the same as in `staticLib`:

- `libName`: The name of the library.
- `path`: Path relative to `/src` where the project will be located.
- `depends`: Library dependencies.
- `nrcMode`: How the library's resources will be managed.
- `standard`: Optionally, you can indicate the *"C/C++ StandardC/C++ Standard"* (page 104).

It is totally valid to create the static and dynamic version of a library. The only condition is to rename one of them, since **it is not possible to have two projects with the same name in the same solution**. Next, we've created two versions of `casino`, linking each with an executable.

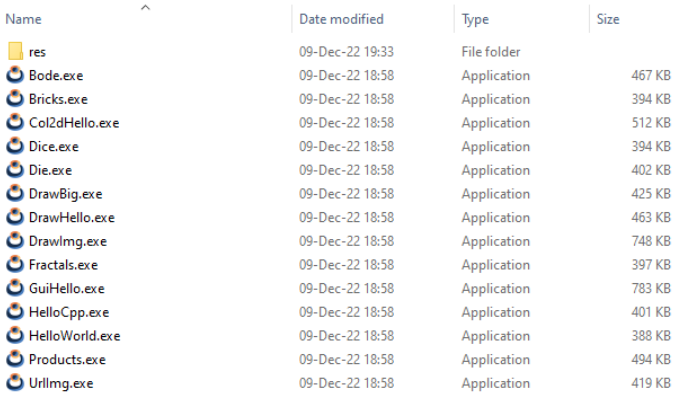
```
staticLib("casino" "demo/casino" "draw2d" NRC_EMBEDDED)
dynamicLib("casino_d" "demo/casino" "draw2d" NRC_EMBEDDED)

// 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)
```

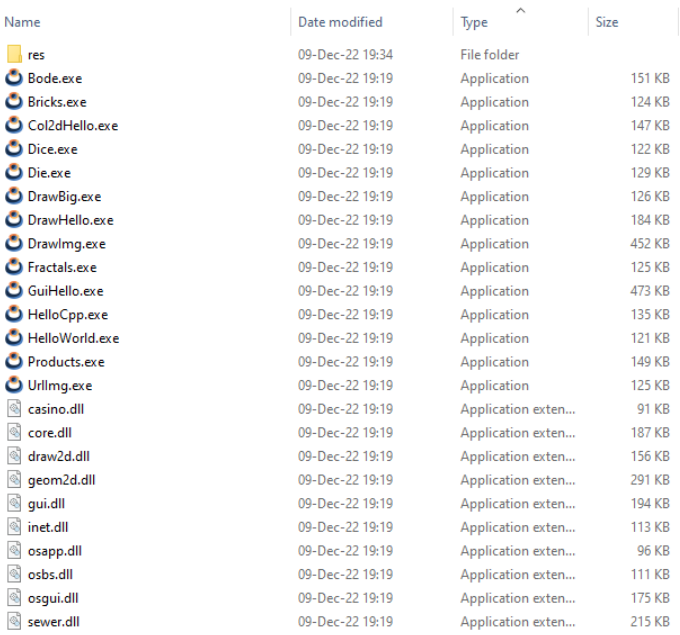
10.2.1. Advantages of DLLs

As we have been able to intuit in the previous example, using DLLs we will reduce the size of the executables, grouping the common binary code (Figure 10.6), (Figure 10.7). This is precisely what operating systems do. For example, `Die.exe` will ultimately need to access Windows API functions. If all applications were to statically link Windows binaries, their size would grow inordinately and a lot of space within the file system would be wasted.



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

Figure 10.6: The programming examples occupy **6.52 Mb** in their static version.



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

Figure 10.7: The programming examples occupy **4.08 Mb** in their dynamic version.

Another great advantage of DLLs is memory savings at runtime. For example, if we load `Die.exe`, `casino.dll` will be loaded at the same time. But if we then load `Dice.`

exe, both will share the existing copy of casino.dll in memory. However, with static linking, there would be two copies of casino.lib in RAM: One built into Die.exe and one from Dice.exe.

10.2.2. Disadvantages of DLLs

The main drawback of using DLLs is the incompatibility that can arise between the different versions of a library. Suppose we release a first version of the three products:

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

A few months later, we released a new version of the Dice.exe application that involves changes to casino.dll. In that case, the layout of our *suite* would look like this:

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

If we have not been very careful, it is very likely that Die.exe no longer works because it is not compatible with the new version of the DLL. This problem is causing many developers head and has been dubbed *DLL Hell*[1]. Since in this example we work on a "controlled" environment we could solve it without too much trouble, creating a new version of all the applications running under casino.dll(v2).

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

This will not always be possible. Now suppose that our company develops only casino.dll and third parties work on the final products. Now each product will have its production and distribution cycles (uncontrolled environment) so, to avoid problems, each company will include a copy of the specific version of the DLL with which their product works. This could lead to the following scenario:

```
/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

Seeing this, we intuit that the benefits of using DLLs are not so good anymore, especially with regard to space optimization and load times. The fact is that it can get even worse. Typically, libraries are written to be as generic as possible and to serve many applications. On many occasions, a given application uses only a few functions from each library it links to. By using static libraries, the size of the (Figure 10.8) executable can be considerably reduced, since the linker knows exactly what specific functions the application uses and adds the code that is strictly necessary. However, using DLLs, we must distribute the entire library for very few functions that the (Figure 10.9) executable uses. In this case, you are wasting space and unnecessarily increasing application load times.

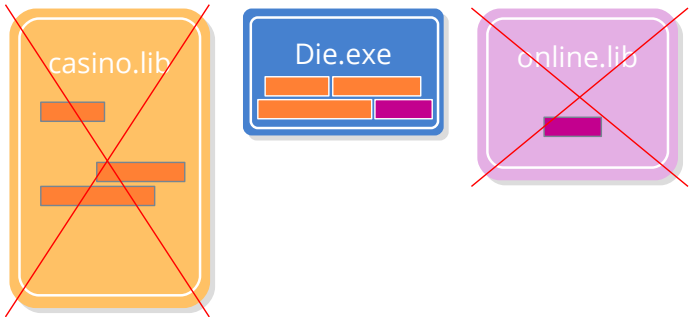


Figure 10.8: With static libraries the space and load times of this application are optimized.

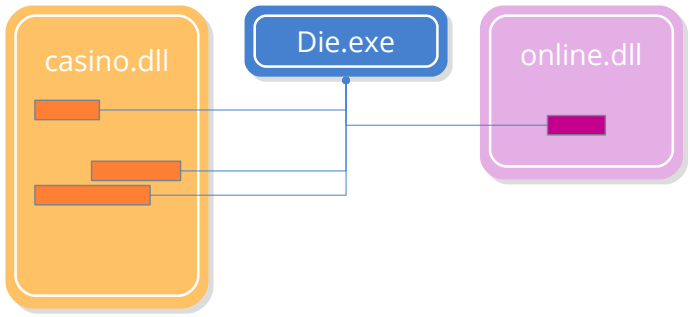


Figure 10.9: With dynamic libraries this application occupies more than it should and its load times increase.

10.2.3. Check links with DLLs

When an executable is launched, for example `Die.exe`, all dynamic libraries linked to it are loaded into memory (if they don't already exist). If there are any problems while loading, **the executable will fail to start** and the operating system will display some kind of error.

Links on Windows

Windows will display a (Figure 10.10) error message when it cannot load a DLL associated with an executable.

If we want to see which DLLs are linked to an executable, we will use the `dumpbin` command.

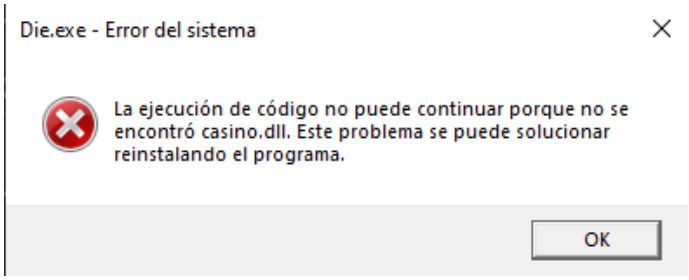


Figure 10.10: Error loading DLL `casino`.

```
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
```

We see, at the beginning, the dependency with `casino.dll`. The rest are Windows libraries related to the kernel and the user interface. In the case that we make a static link of `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
```

`casino.dll` no longer appears, having been statically linked inside `Die.exe`.

Links in Linux

In Linux something similar happens, we will get an error if it is not possible to load a dynamic library (`*.so`).

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

To check which libraries are linked to an executable we use the `ldd` command.

```
~/$ 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)
...
```

Where we see that `Die` depends on `libcasino.so`. The rest are dependencies of the Linux kernel, the C standard library, and GTK.

Links on macOS: We use the `otool` command.

```
% 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. Loading DLLs at runtime

Until now, the importation of DLL symbols is resolved at compile time, or rather at link time. This means that:

- Executables can directly access global variables and functions defined in the DLL. Returning to the code of `Dice.exe`, we have:

```
#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]);
}
```

  - Made a `#include "ddraw.h"`, public header of `casino`.
  - `die_draw()`, `kDEF_PADDING`, `kDEF_CORNER`, `kDEF_RADIUS` have been used.

- The dynamic library `casino.dll` will be loaded automatically just before `Dice.exe`.
- The use of static or dynamic version of `casino` does not imply changes in the code of `Dice`. We would just have to change the dependencies inside `desktopApp()` and recompile the application.

```
// 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)
```

However, there is the possibility that the programmer is in charge of loading, unloading and accessing the symbols of the DLLs at any time. This is known as run-time binding or symbol-less binding. At src/demo/dice2[2] we have a new version of `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);
}
```

- Line 6 loads the `casino_d` library.
- Line 7 accesses the `die_draw` function defined in `casino_d`.
- Lines 11-13 access public variables of `casino_d`.
- Lines 15-20 use `die_draw` via the `func_draw` pointer.
- Line 21 unloads the `casino_d` library from memory.

As we can see, this loading at runtime does imply changes to the source code, but it also brings with it certain advantages that we can take advantage of.

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

- The library is loaded when we need it, not at the start of the program. This is why it is **very important** that `casino_d` does not appear as a dependency of `Dice2`.

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

- We can have different versions of `casino` and choose which one to use at runtime. This is the working mechanism of the *plug-ins* used by many applications. For example, the program *Rhinoceros 3D* enriches its functionality thanks to new commands implemented by third parties and added at any time through a system of plugins (.DLLs) (Figure 10.11).

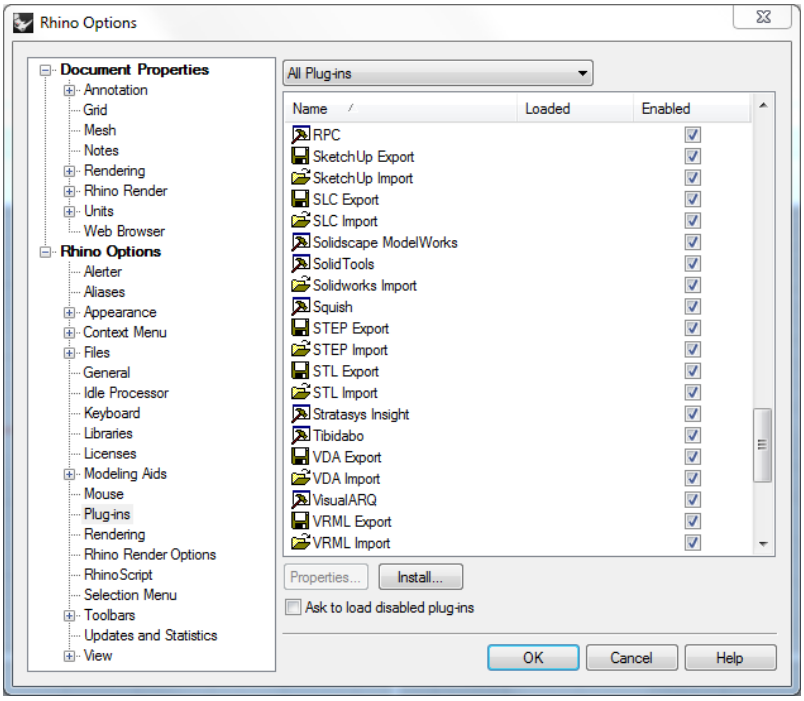


Figure 10.11: Rhinoceros 3D plug-in system, implemented using DLLs.

10.2.5. Location of DLLs

When the operating system must load a dynamic library, it follows a certain search order. On Windows systems it searches in this order:

- The same directory as the executable.
- The current working directory.
- El directorio `%SystemRoot%\System32`.

- The `%SystemRoot%` directory.
- The directories specified in the `PATH` environment variable.

On the other hand, on Linux and macOS:

- The directories specified in the environment variable `LD_LIBRARY_PATH` (Linux) or `DYLD_LIBRARY_PATH` (macOS).
- The directories specified in the `rpath` executable.
- The system directories `/lib, /usr/lib, etc`.

Here we have a big difference between Windows and Unix, since in the latter it is possible to add dependency search directories inside the executable. This variable is known as **`RPATH`** and is not available on Windows. To query the value of the `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)
...
```

Executables generated by NAppGUI's `CMakeLists.txt` automatically set the `RPATH` to find dynamic dependencies in the same directory as executables on Linux or bundles on macOS.

10.3. Symbols and visibility

In the linking process after the compilation of the library, those elements that can generate machine code or occupy space in the final binary are called **symbol**. These are methods, functions, and global variables. Symbols are not considered:

- Type definitions such as `enum`, `struct`, or `union`. They help the programmer to organize the code and the compiler to validate it, but they do not generate any binary code. They do not exist from the point of view of the linker.
- Local variables. These are automatically created and destroyed in the *"Stack SegmentStack Segment"* (page 162) during program execution. They do not exist at link time.

On the other hand, all functions and global variables declared as `static` inside a `*.c` module will be considered **private symbols** not visible in link time and where the compiler is free to perform optimizations. With this in mind, the code within NAppGUI is organized as follows:

- ***.c:** Implementation file. Definition of symbols (functions and global variables).
- ***.h:** Public header file. Declaration of global functions and variables (`extern`), available to the user of the library.
- ***.hxx:** Declaration of public types: `struct`, `union` and `enum`.
- ***.inl:** Declaration of functions and private variables. Only modules internal to the library will have access to these symbols.
- ***.ixx:** Declaration of private types. Those shared between the modules of the library, but not with the outside.

If a function is only needed inside a `*.c` module, it is not included in a `*.inl`. It will be marked as `static` within the `*.c` itself. This way it will not be visible to the linker and will allow the compiler to perform optimizations.

In the same way, types that are only used within a specific module will be declared at the beginning of the `*.c` and not in the `*.ixx`.

In favor of code maintainability and scalability, type and function declarations will be kept as private as possible.

10.3.1. Export in DLLs

When we generate a dynamic link library, in addition to including the public symbols in one or more `*.h` headers, we must explicitly mark them as exportable. The export macro is declared in the `*.def` file of each library. For example in `core.def`, the macro `_core_api` is defined.

Listing 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
```

This macro must precede all functions and variables declared in the `*.h`. Projects based on `/src/CMakeLists.txt` will automatically define the `CORE_IMPORT` and `NAPPGUI_SHARED_LIB` macros whenever dynamic libraries are to be generated (exported) or when they are to be used by an executable (import). In the case of third-party programs (not generated by `/src/CMakeLists.txt`) the import macros must be defined (`CORE_IMPORT`, `GUI_IMPORT`, etc) before including the headers.

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. Checking in DLLs

We can see, from a dynamic library binary, which public symbols it exports. On Windows we will use `dumpbin /exports dllname`, on Linux `nm -D soname` and on macOS `nm -gU dylibname`.

Public symbols from `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
...
```

Public symbols from `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
```

Public symbols from `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
```

Chapter 11

Resources

If we internationalize everything, we end up with rules that stifle freedom and innovation.

Myron Scholes

Resources are data that are required by the application but do not reside in the area of the executable. In other words, they are not directly accessible through program variables, but rather have to be pre-loaded before they can be used. The most common are the texts and images used in the user interface, although any type of file can become a resource (sounds, fonts, 3d models, html pages, etc). To illustrate its use with a real example, we are going to use the `Die` application (Figure 11.1), included in `/src/demo/die`.

11.1. Types of resources

- **Texts**: Although it is very easy to include texts in the code as C variables, in practice this is not advisable for two reasons: The first is that, normally, it is not

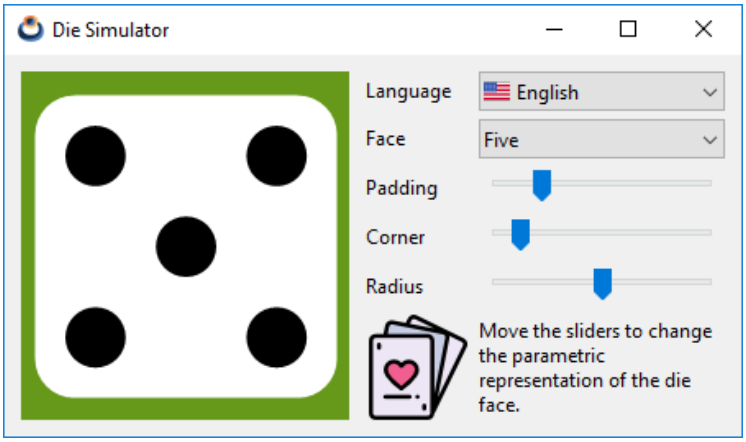


Figure 11.1: Die Application.

the programmers who They compose the messages that the program displays. By separating them into a separate file, other team members can review and edit them without having to directly access the code. The second reason is internationalization. It is an almost essential requirement today to be able to change the language of the program and this can involve several members of the team, as well as the fact that several text strings refer to the same message. Therefore, extracting them from the source code will be almost essential.

- **Images**: It is not usual for the program icons to change depending on the language, although it may be the case. The tricky thing here is transforming a .jpg or .png file into a C variable (Listing 11.1). You have to serialize the file and paste it into the code, something very tedious and difficult for the programmer to maintain. It is preferable to have the images in a separate folder and access them at runtime.

Listing 11.1: Png image embedded in the source code.

```
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,
            ... };
```

- **Files**: Apart from text and images, any file can become a resource. In this case, the application will receive a block of bytes with its content, which it must know how to interpret.

11.2. Create resources

If we go to the source directory of the application (`/src/demo/die`), we see that there is a folder called `/res` added by CMake when creating the project. Inside are several `logo.*` files with the *"Application iconApplication icon"* (page 140).

You can also see a folder called `/res/res_die` which **wasn't created by CMake**, but added later when writing the program. This subfolder is considered a **resource pack** and will contain a set of texts, images or files that will be loaded "in bulk" at some point in the execution. We can create as many packages as necessary depending on the size and logic of our program.

> *In large applications, organize your resources in such a way that it is not necessary to load all of them when starting the application. Certain resources may only be needed when the user performs some action.*

You will see that inside `/res/res_die` there is a `strings.msg` whose content is shown below:

Listing 11.2: Die's message file.

```
/* 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
```

Also contains the `cards.png` image and the `spain.png` and `usa.png` (Figure 11.2) icons.

Each line within the `strings.msg` file defines a new message consisting of an identifier (eg `TEXT_FACE`) followed by the text to be displayed in the program (*Face* in this case). Text is considered from the first non-blank character after the identifier to the end of the line. You don't need to put it in quotes (`"Face"`) like you do in C:

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

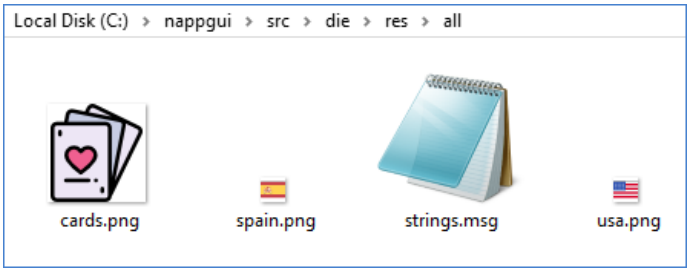


Figure 11.2: Resource bundle at `src/die/res/res_die`.

```
OTHER    Other text.
```

You also don't have to use escape sequences ('\\', '\", ...), with the single exception of `'\n'` for multi-line messages:

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

The message identifier follows the rules for C identifiers, except that letters must be uppercase:

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

Messages accept any Unicode character. We can split the texts into as many `*.msg` files as needed and they must be stored in **UTF8 format**.

> *Visual Studio does not save files in UTF8 by default. Be sure to do so on every `*.msg` that contains non-US-ASCII characters. `File->Save As->Save with encoding->Unicode (UTF8 Without Signature)- Codepage 65001`.*

11.3. Internationalization (i18n)

We have used English as the main language in the program, but we want it to be translated into Spanish as well. To do this we go back to the `/res/res_die` folder, where we see the `/es_es` subdirectory that contains another `strings.msg` file. The identifiers in that file are the same as in `/res_die/strings.msg` but the texts are in another language. Depending on the selected language, the program will use one version or another.

Listing 11.3: Die's message file, translated into Spanish.

```
/* 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
```

We must take into account some simple rules when locating resources:

- If the local version of a resource does not exist, the global version of the resource will be used. CMake will warn if there is **untranslated text** *"nrc warningsnrc warnings"* (page 139).
- Those resources only present in local folders will be ignored. It is imperative that the global version of each exists.
- Resource "subpackages" are not allowed. Only two levels will be processed: `src/res/packname` for globals and `src/res/packname/local` for locals.
- Resource bundles must have a unique name within the solution. One strategy might be to prepend the project name: `/appname_pack1`, `libname_pack2`, etc.
- Existing resources in the root folder (`/res`) will be ignored. All resources must be contained in a package `/res/pack1/`, `/res/pack2/`, etc.
- Localized texts must have the same identifier as their global counterpart. Otherwise they are considered different messages.
- To create a localized version of an image or other file, include it in its corresponding local folder (e.g. `/res/res_die/es_es/cards.png`) using **the same file name** than the global version.
- To name the localized folders, use the two-letter language code ISO 639-1[1] (in, is, fr , de, zh, ...) and, optionally, the two-letter country code 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. Runtime translation

For each resource bundle, CMake creates a `*.h` with the same name as the folder: `res_die.h` in this case (Listing 11.4) . This file contains the resource identifiers, as well as a function that allows us to access them `res_die_respack()`. In (Listing 11.5) we see the actions to be carried out to use these resources in our program.

Listing 11.4: Header file `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
```

Listing 11.5: Load and use of resources.

```
#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);
}
```

- Line 1 includes the (Listing 11.4) resource bundle header, which is automatically generated by CMake.
- Line 3 registers the package in *"Gui"* (page 297), the library in charge of the graphical interface. If the application had more resource packs we would add them in the same way.
- Line 4 sets the default language (English).
- Lines 6 and 7 assign a text and an image to two controls respectively. Identifiers are defined in `"res_die.h"`, as we just saw.
- Line 13 translates the entire interface in response to a change in the *"PopUp"* (page 308) control (Figure 11.3).

Figure 11.3: Translation of the Die application, without destroying the window or rebooting.

Basically, a call to `gui_language`, involves coordinating three actions:

- Load the located resources and replace them with the current ones.
- Assign the new texts and images to all the controls and menus of the program.
- Resize the windows and menus, since changing texts and images will influence the size of the controls.

11.5. Edit resources

To add new resource files or delete any of the existing ones, we just have to go to the `res/res_die` folder through the file explorer and do it there directly. The `*.msg` message files can be edited from within Visual Studio, as CMake includes them within the (Figure 11.4) IDE. After making any changes to the resource folder or editing a `*.msg` file, we must relaunch CMake so that these modifications are integrated back into the project. After each update, the identifiers of the new resources will be created and those whose associated resource has disappeared will be deleted, which will cause compilation errors that will facilitate the correction of the code.

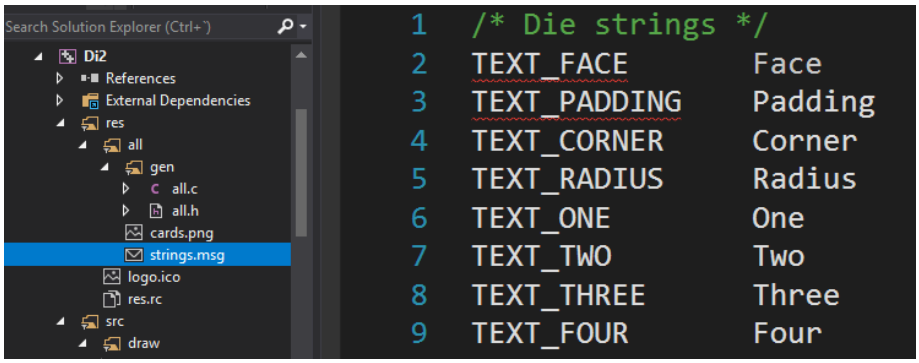


Figure 11.4: Editing resources within Visual Studio.

11.6. Manual management

Although the usual thing will be to delegate the management of resources to the `gui` library, it is possible to access the content of the packages directly, as we see in (Listing 11.6).

Listing 11.6: Direct access to resources.

```
#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);
```

- Line 1 includes the resource bundle header.
- Line 3 creates an object with the content of the package in the Spanish language. Each resource pack will provide its own constructor, whose name will start with the

name of its xxxx_respack() folder.

- Lines 5 and 6 get a text and an image respectively to assign to interface controls.
- Line 8 destroys the resource bundle, at the end of its use.

There is a big difference between allocating resources using ResId or using respack_ (Listing 11.7) functions. In the first case, the label control will be "sensitive" to language changes made by gui_language . However, in cases 2 and 3 a constant text has been assigned to the control, which will not be affected by this function. We will be responsible for changing the text, if necessary.

Listing 11.7: Different ways to access resources.

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

The choice of one access mode or another will depend on the requirements of the program. We remind you that in order to carry out automatic translations, resources must be registered with gui_respack.

11.7. Resource processing

Let's see in a little more detail how NAppGUI generates the resource modules. By setting NRC_EMBEDDED in the desktopApp() command, we tell CMake to process the resources of the Die project. We can also choose the NRC_PACKED option which we will talk about next. When we launch CMake, it traverses the subfolders within the res directory of each project, calling the **nrc** *(NAppGUI Resource Compiler)* utility (Figure 11.5). This program is located in the prj/scripts folder of the SDK distribution. For each resource bundle, *nrc* creates two source files (a .c and a .h) and links them to the project. The .h contains the identifiers and the constructor we've seen in (Listing 11.4). For its part, the .c performs the implementation of the package based on the content of each folder and the nrcMode mode.

> *Files created by nrc are considered generated code and are not stored in the src folder but in the build folder. They will be updated every time CMake is run, regardless of the platform we are working on. In contrast, the original resource files (located in the res folder) are considered part of the source code.*

11.8. Resource distribution

In the previous chapter, when creating the Visual Studio solution, we indicated that the constant NRC_EMBEDDED had to be used in the desktopApp() statement inside the

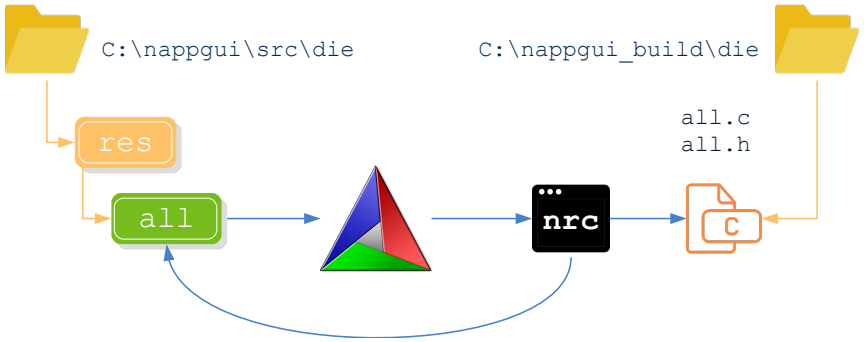


Figure 11.5: Processing resources using CMake and nrc.

`CMakeLists.txt` file. There are two other modes related to resource management that can be configured separately within each `desktopApp()` command:

- `NRC_NONE`: CMake will ignore the contents of the `res` folder, except for the application icon. No resource packs will be generated even if there is content inside this folder.
- `NRC_EMBEDDED`: The resources, with all their translations, are embedded as part of the (Figure 11.6) executable. It is a very interesting option for small or medium-sized applications, since we will supply the entire program in a single `*.exe` file. An installer will not be necessary and we will have the certainty that the software will not fail due to the lack of some external file. The drawback is that, obviously, the size of the executable will grow considerably, so it is not advisable in programs with many resources, very heavy, or with a multitude of translations.
- `NRC_PACKED`: For each resource package, a `*.res` file will be created external to the executable that will be loaded and released at runtime as needed (Figure 11.7). The advantages of this method are the disadvantages of the previous one and vice versa: Smaller executables, but with external dependencies (the `.res` themselves) that must be distributed together. Memory usage will also be optimized by being able to load `*.res` on demand.

Name	Date Modified	Size	Kind
Contents	Today at 18:09	--	Folder
Info.plist	Today at 18:09	1 KB	Property List
MacOS	Today at 18:09	--	Folder
Products	Today at 18:09	948 KB	Unix executable
PkgInfo	Today at 18:09	8 bytes	TextEdit
resources	Today at 18:09	--	Folder
en.lproj	Today at 18:09	--	Folder
logo.icns	Today at 18:09	302 KB	Apple i...n image

Figure 11.6: Distributing a macOS application with embedded resources.

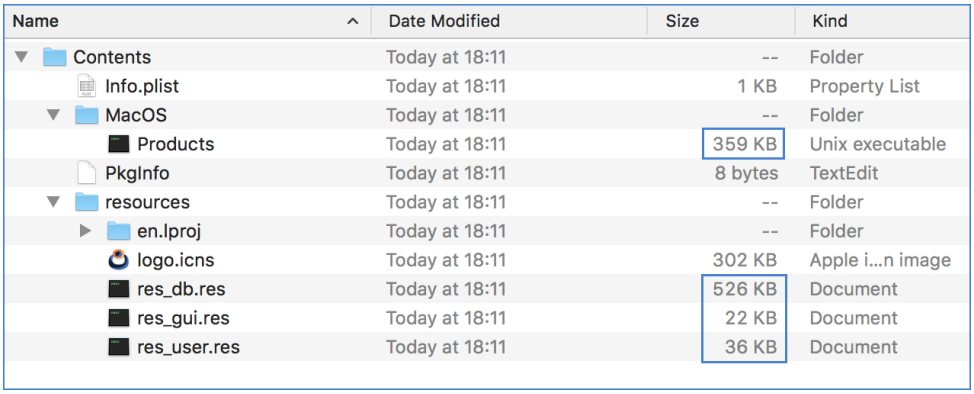


Figure 11.7: A distribution of the same macOS app with packed resources.

CMake manages the location of the resource packages for us. On Windows and Linux applications it will copy all `*.res` into the executable directory. On macOS it will place them in the `resources` folder of the bundle. A very important fact is that **we don't have to modify the source code** when switching from one modality to another. *nrc* already takes care of managing the payload based on the package type. It makes sense to start with `NRC_EMBEDDED`, and if the project grows, change to `NRC_PACKED`. We just have to launch CMake again and recompile the project for the change to take effect.

> *On Windows and Linux the `*.res` files must always be installed in the same directory as the executable. For macOS, CMake generates a distribution-ready bundle and installs the resource bundles in the `/resources` directory of that bundle.*

11.9. nrc warnings

nrc is a silent script whose work is integrated into the CMake *build process*, mostly unnoticed. But there are times when you detect anomalies in the resource directories and you need to let us know in some way. In these cases a red line will appear in the CMake console indicating the affected project and package(s) (Figure 11.8). The details are dumped into the `NRCLog.txt` file located in the generated resources folder (CMake displays the full path).

If the bugs are critical, *nrc* will not be able to generate the `*.h` and `*.c` associated with the package, preventing the application from crashing. can compile (in essence it is still a compilation error). Other times they are mere *warnings* that should be fixed, but they allow you to continue compiling. Specifically, the **critical errors** that affect *nrc* are the following: (we show them in English as they are written in `NRCLog.txt`).

- MsgError (%s:%d): Comment not closed (%s).

```
- 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
```

Figure 11.8: *nrc* encountered anomalies while processing resources.

- 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.

On the other hand, non-critical warnings:

- 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. Application icon

When we create a new project, CMake sets a default icon for the application, which it places in the `/res` directory, with the name `logo*`. This image will be "embedded" in the executable and will be used by the operating system to render the application on the desktop (Figure 11.9). Windows and Linux also use it in the window title bar. We have three versions:

- **logo256.ico**: Version for Windows Vista and later. They must include the resolutions: 256x256, 48x48, 32x32 and 16x16.
- **logo48.ico**: Version for Linux and VisualStudio 2008 and 2005, which do not support 256x256 resolutions. This version only includes: 48x48, 32x32 and 16x16.
- **logo.icns**: Version for macOS. Resolutions 512x512, 256x256, 128x128, 32x32 and 16x16 both in normal resolution (@1x) and Retina Display (@2x).

Figure 11.9: Application icons on the Windows taskbar.

CMake already takes care of using the appropriate version of the icon depending on the platform we are compiling on. To change the default icon, open the `logo*` files with some graphical editor (Figure 11.10), make the changes, and relaunch CMake. **Very important:** do not change the names of the files, they should always be `logo256.ico`, `logo48.ico` and `logo.icns`.

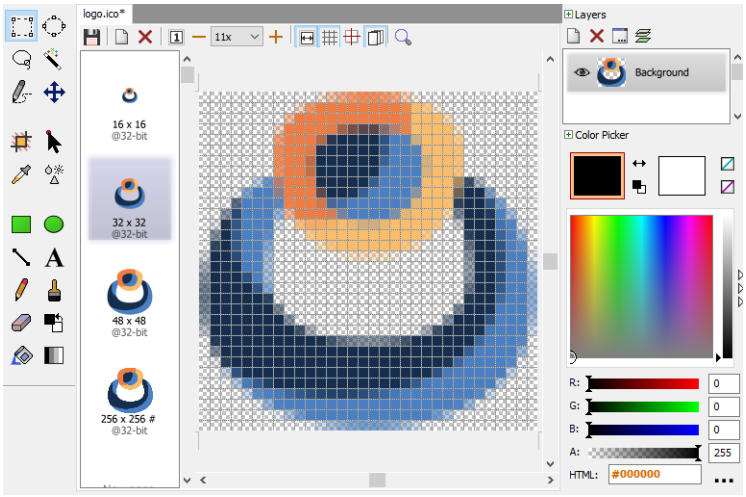


Figure 11.10: Editing `logo.ico`.

Part 2

Introduction to the API

Chapter 12

NAppGUI SDK

While civilians (i.e., nonprogrammers) often fantasize about winning the lottery, the equivalent for many programmers is the rare opportunity to create a new library from scratch, without the constraints that often frustrate their desires to extend and improve an existing library.

Philip J. Schneider - Industrial Light + Magic

12.1. NAppGUI API

The NAppGUI implementation has been split into several libraries written in ANSI-C (C90) with small parts in C++98 (Figure 12.1). The project compiles without problems in all versions of Visual Studio (since VS2005), Xcode (since 3) and GCC (since 4). It can be used for developing high-performance applications written in C on Windows, macOS, and Linux systems. A clear line has been drawn that separates packages oriented to computation and data access *(back-end)* from those intended for the presentation or interface layers *(front-end)*. We have also followed certain *"StandardsStandards"* (page 58) whose bases are centralized in the *"Sewer"* (page 149) library, which, although it does not incorporate much functionality, does define the basic types and configuration macros common to all the project.

- Packages that do not contain platform dependent code.
- Packages that contain platform dependent code under a common interface.

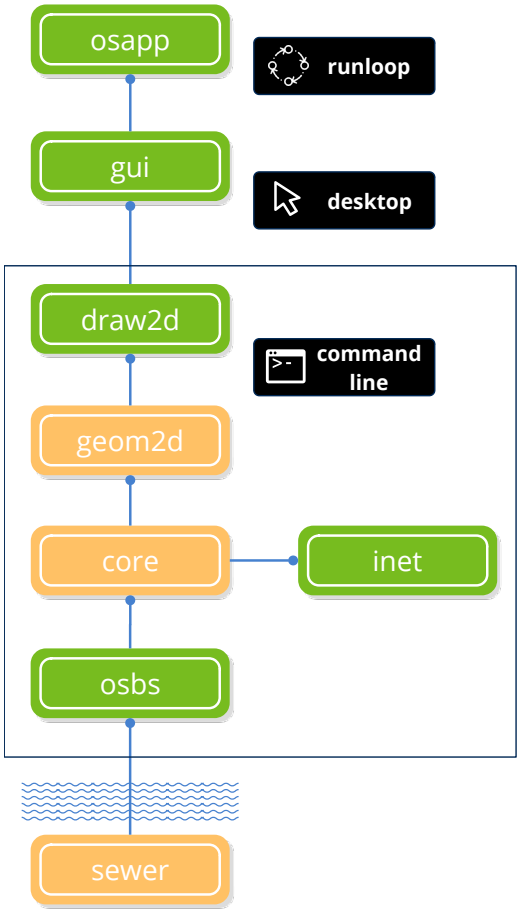


Figure 12.1: NAppGUI architecture.

- *"Sewer"* (page 149): Basic types, assertions, Unicode, standard C library, math functions.
- *"Osbs"* (page 166): Operating system services. Portable API on files, directories, processes, threads, memory, etc.
- *"Core"* (page 187): Commonly used non-graphical utilities. Memory auditor, data structures, I/O channels, lexical analysis, etc.
- *"Geom2D"* (page 235): 2D geometry. Transformations, vectors, polygons, collisions, etc.
- *"Draw2D"* (page 256): Vector drawing API, images and fonts.
- *"Gui"* (page 297): High-level user interface composer.
- *"OSApp"* (page 369): Desktop applications. Message loops.
- *"INet"* (page 377): Internet protocols and services, such as HTTP.

12.2. Online resources

For obvious reasons of space, it is impossible in this book to include a complete reference of each and every one of the functions that make up NAppGUI. On the project's Website[1] you will find a detailed feature-by-feature guide, as well as the source code of several sample applications.

Therefore, please go through this entire section of the book in a leisurely manner, with the sole purpose of getting a general idea of the structure of the software and the different parts that compose it.

12.3. A little history

I started working on this project unconsciously, in the middle of 2008 when I was finishing my studies in Computer Engineering at the University of Alicante. He wanted to develop a physical systems simulator that would work on both PC-Windows computers and Apple iMacs without having to duplicate all the work. The technological alternatives of the time, such as GTK or Qt, did not convince me at all since they were too heavy, complicated to use and slow, so they would end up tarnishing the quality, elegance and effort that I was putting into my mathematical calculation algorithms. After spending several months evaluating different libraries for cross-platform programming, I downloaded some technical manuals from Apple to program directly in Cocoa, the manzanita manufacturer's base technology for developing software on iMac. In the middle of 2010 I started to see the first results and this was encouraging. I had built an application with my simulator prototype in just 500Kb (Figure 12.2), in contrast to the 30+Mb of dependencies required by third-party solutions. The code was compact and clean, the application worked at breakneck speed and, above all, it had a professional appearance that was somewhat reminiscent of iMovie, it allowed 3D views to be manipulated like in a video game and provided technical simulation data in real time. This inspired me to continue working on drawing a barrier between the reusable part of the application and the part that depends on a specific technology. This would allow my simulator to be adapted to different computer models and operating systems.

At the same time, in September 2008 I rejoined the labor market after six years at the University, a market in which I am still currently (May 2021), although the last few years I have been working as a freelancer from home, which allows me to organize the agenda and optimize my time to the maximum. In these years I have not abandoned my personal project, I have continued working on it part-time simply for pure hobby. Its development has allowed me to investigate and delve into interesting areas for me and constantly recycle myself. In 2013 I made my first foray into the world of entrepreneurship as a co-founder of iMech Technologies, a software company with which I am still linked and whose main

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

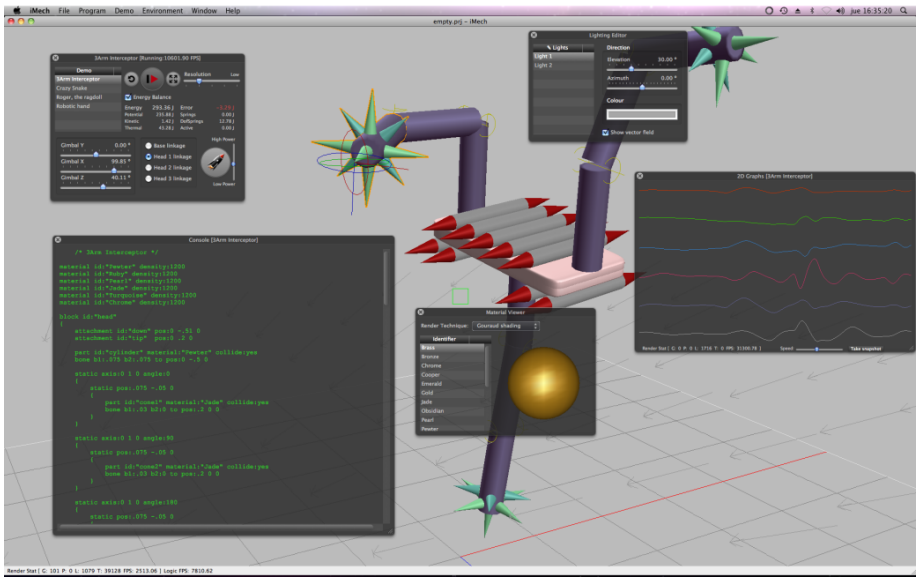


Figure 12.2: iMech simulator, based on a primitive version of NAppGUI.

objective was the sale of the simulation engine that I had previously created. By not coming up with a solid marketing strategy, we didn't achieve our initial goals with iMech, but we were able to turn it around by adding new customers and it's still alive today.

In mid-2015 I began to consider the fact that all the technical effort made during these years has enough entity to become a product by itself. It was then when I created the NAppGUI project and started migrating all the iMech libraries dedicated to cross-platform development. Over the last few years I've completed support for *Cocoa* and included support for *Win32* and *Gtk+*. I have created this documentation in Spanish and English, with the help of Google translation services.

On December 31, 2019, I upload to GitHub the first public pre-compiled version of NAppGUI.

In May 2020 I start the development of the first commercial application programmed entirely with NAppGUI.

On September 8, 2021, I release the source code of NAppGUI on GitHub, making it an Open Source project under the MIT license.

Chapter 13

Sewer library

Even the grandest palaces needed sewers.

Tom Lehrer

13.1. Sewer

Sewer is the first library within the NAppGUI SDK (Figure 13.1). It declares the basic types, the Unicode support, assertions, pointers safe manipulation, elementary math

functions, Standard I/O and dynamic memory allocation. It is also used as a "sink" to bury the unsightly preprocessor macros necessary to configure the compiler, CPU, platforms, etc. As dependencies only has a few headers of the C standard library:

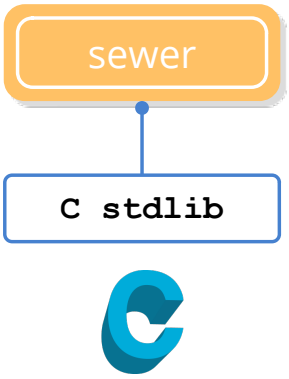


Figure 13.1: Dependencies of *sewer*. See *"NAppGUI API"* (page 145).

13.1.1. The C standard library

The C standard library *(cstdlib)* is not part of the C language, but implements functions of great utility for the developer that solve typical programming problems. Any C programmer has used it more or less and its study is usually linked to learning the language itself (Figure 13.2).

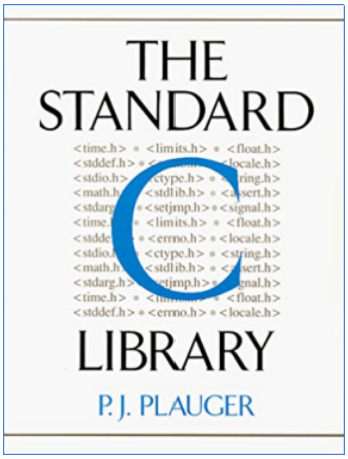


Figure 13.2: A complete reference to the C library is found in the P.J.Plauger book.

This library is located halfway between the application and system calls and provides a portable API for file access, dynamic memory, I/O, time, etc (Figure 13.3). It also implements mathematical functions, conversion, search, string management, etc. In one way or another, NAppGUI integrates its functionality, so it's not necessary (or advisable) to use *cstdlib* directly in the application layer. The reasons that have motivated this design decision can be summarized in:

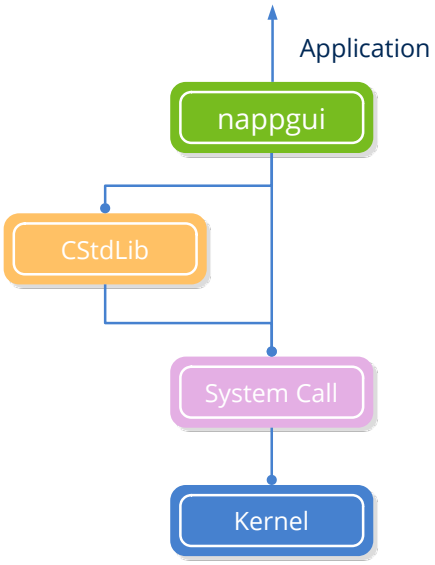


Figure 13.3: The functionality of the C library has been integrated in NAppGUI, avoiding its direct use in applications.

- **Small differences:** Unix-like systems do not support the secure *cstdlib* versions implemented by Microsoft (`strcpy_s()` and others). The use of classical functions (without the suffix `_s`) is insecure and will trigger annoying *warnings* in Visual Studio.
- **Security:** Related to the previous one, avoids *buffer overflow* vulnerabilities in the processing of memory blocks and strings.
- **Duplicity:** Much of the functionality of *cstdlib* *is* already implemented in *osbs* library using direct system calls (files, dynamic memory, I/O, time, etc.)
- **Completeness:** The *cstdlib* functions related to files (`fopen()` and others) do not include support for directory management. *"Files and directories"* (page 177) presents a complete API based on system calls.
- **Performance:** In certain cases, especially in mathematical functions and memory management, it may be interesting to change the implementation of *cstdlib* to an improved one. All applications will benefit from the change, without having to modify your code.
- **Clarity:** The behavior of some *cstdlib* functions is not entirely clear and can lead to confusion. For example, `strtoul` has a very particular functioning that we must remember every time we use it.

```
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
```

- **Style:** The use of *sewer* functions does not break the aesthetics of an application written with 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);
```

- **Independence:** NAppGUI internally uses a very small subset of *cstdlib* functions. It is possible that in the future we will make our own implementations and completely disconnect the support of the standard library.
- **Static link**: If we statically link the standard library, *sewer* will contain all dependencies internally. This will avoid possible incompatibilities with the runtimes installed on each machine (the classic Windows VC++ Redistributables). With this we will be certain that our executables will work, regardless of the version of the C runtime that exists in each case. If all calls to *cstdlib* are inside *sewer*, we free higher-level libraries from their handling and possible runtime errors related to the C runtime.

Static link of the *cstdlib* in Sewer. Doesn't need the C runtime.

```
RUNTIME_C_LIBRARY "static"

dumpbin /dependents dsewer.dll

  Image has the following dependencies:

    KERNEL32.dll
```

Dynamic binding of the *cstdlib* in Sewer. Needs to have a specific runtime installed.

```
RUNTIME_C_LIBRARY "dynamic"

dumpbin /dependents dsewer.dll

  Image has the following dependencies:

    KERNEL32.dll
    VCRUNTIME140D.dll
    ucrtbased.dll
```

To avoid possible bugs or incompatibilities, do not use C Standard Library functions directly in applications. Always look for an equivalent NAppGUI function.

13.2. Asserts

asserts are sentences distributed by the source code that perform an intensive *"Dynamic analysisDynamic analysis"* (page 61), helping to detect errors at runtime. When the *assert* condition becomes `FALSE`, the program execution stops and a warning window is displayed (Figure 13.4).

- Use `cassert` to introduce a dynamic check in your code.
- Use `cassert_no_null` once you have to access the content of a pointer.

```
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");
    ...
}
```

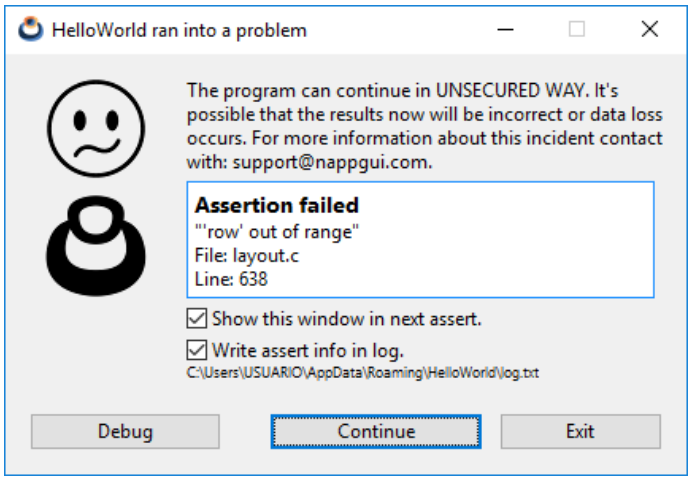


Figure 13.4: *assert* window displayed after a runtime error.

At this time we have three alternatives:

- **Debug**: Debug the program: Access the call stack, inspect variables, etc. More in *"Debugging the programDebugging the program"* (page 63).
- **Continue**: Continue with the execution, ignoring the *assert*.
- **Exit**: Exit the program.

To avoid showing this window in futher *asserts*, deactivate the check `'Show this window in next assert'`. Future incidents will be directed to a *log* file. You can also omit dumps in this log, deactivating `'Write assert info in log'`.

> ***asserts*** *sentences provide very valuable information about program anomalies and should never be ignored.*

In the previous example we have seen a "continuable" *assert*, that is, the execution of the program can continue if we press `[Continue]`. However, as we indicated, they should not be ignored indefinitely. On the other hand we have the **critical asserts** (Figure 13.5). Normally they are related to segment violation problems, where it will not be possible to continue running the program.

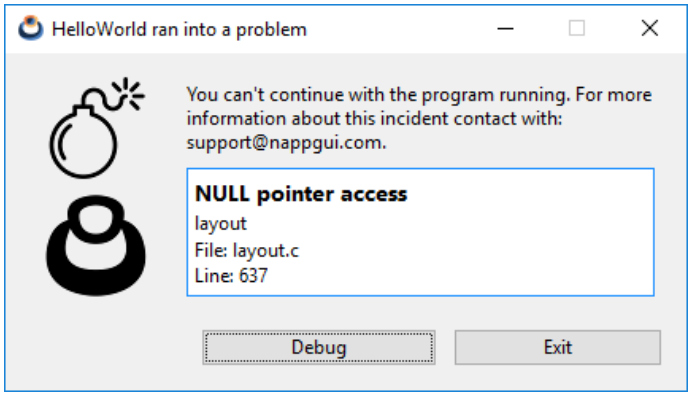


Figure 13.5: Critical *assert* caused by null pointer access.

13.3. Pointers

The *sewer* library provides macros and functions for "safe" pointers manipulation. By "safe" we mean the fact that the SDK will detect improper pointer access just before a *segment violation* occurs. Does it make sense to detect a segment violation if the program is going to crash anyway? Pre-detection plays a very important role when running automated tests. Before the inevitable process closing, it will leave a note in the execution *log.txt*, indicating the reason for the crash.

- Use `ptr_get` to get the content of a pointer.

```
// 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 is a standard in the computer industry, essentially a table, which assigns a unique number to each symbol of each language in the world (Figure 13.6). These values are usually called *codepoints* and are represented by typing **U+** followed by their number in hexadecimal.

- Use `unicode_convers` to convert a string from one encoding to another.
- Use `unicode_to_u32` to get the first codepoint of a string.

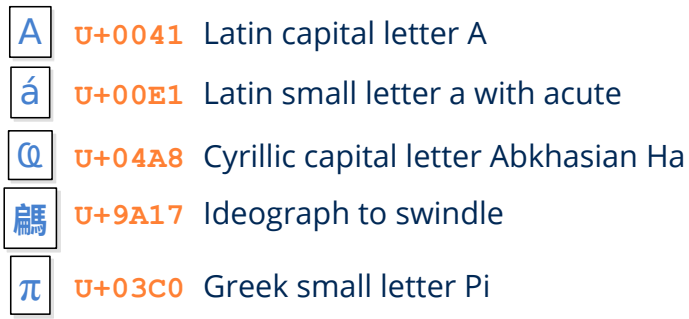


Figure 13.6: Several Unicode *codepoints*.

Related to its structure, it has 17 planes of 65536 *codepoints* each (256 blocks of 256 elements) (Figure 13.7). This gives Unicode a theoretical limit of 1114112 characters, of which 136755 have already been occupied (version 10.0 of June 2017). For real-world applications, the most important one is Plane 0 called *Basic Multilingual Plane* (BMP), which includes the symbols of all the modern languages of the world. The upper planes contain historical characters and additional unconventional symbols.

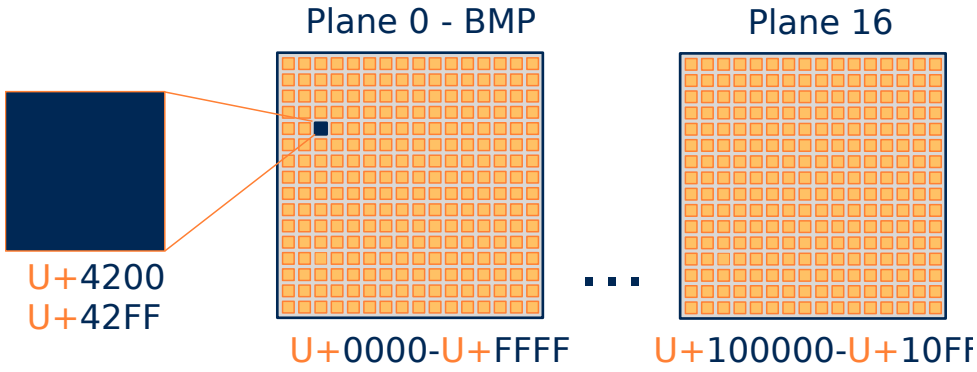


Figure 13.7: Unicode has 17 planes of 256x256 *codepoints* each.

The first computers used ASCII *American Standard Code for Information Interchange*, a 7-bit code that defines all the characters of the English language: 26 lowercase letters (without diacritics), 26 uppercase letters, 10 digits, 32 punctuation symbols, 33 codes

control and a blank space, for a total of 128 positions. Taking the additional bit within a byte, we will have space for another 128 symbols, but still insufficient for all in the world. This results in numerous pages of extended ASCII codes, which is a big problem to share texts, since the same numeric code can represent different symbols according to the ASCII page used (Figure 13.8).

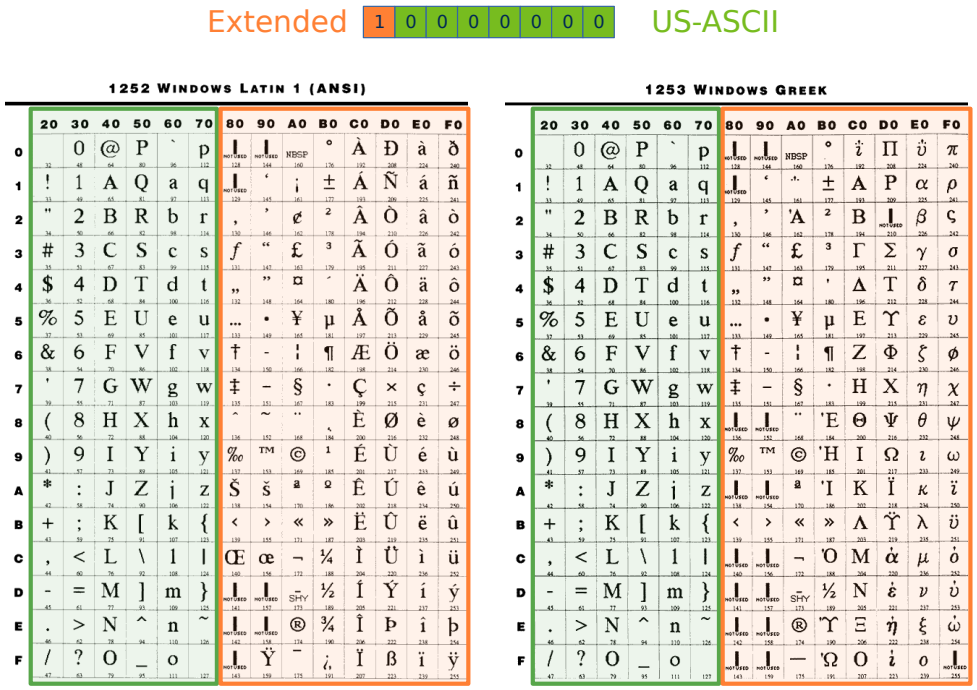


Figure 13.8: On each Extended ASCII page, the top 128 codes represent different characters.

Already in the early 90s, with the advent of the Internet, this problem worsened, as the exchange of information between machines of different nature and country became something everyday. The Unicode Consortium (Figure 13.9) was constituted in California in January of 1991 and, in October of the same year, the first volume of the Unicode standard was published.

Figure 13.9: Full members of the Unicode Consortium.

13.4.1. UTF encodings

Each *codepoint* needs 21 bits to be represented (5 for the plane and 16 for the displacement). This match very badly with the basic types in computers (8, 16 or 32 bits). For this reason, three *Unicode Translation Format - UTF* encodings have been defined, depending on the type of data used in the representation (Figure 13.10).

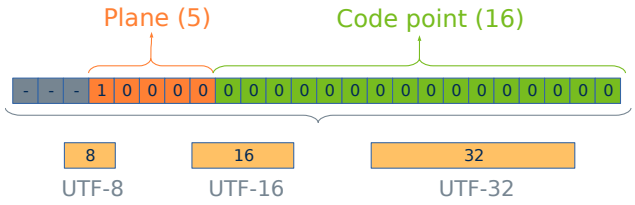


Figure 13.10: Encodings to store 21-bit *codepoints* by elements of 8, 16, or 32.

13.4.2. UTF-32

Without any problem, using 32 bits we can store any *codepoint*. We can also randomly access the elements of an array using an index, in the same way as the classic ASCII C (char) strings. The bad news is the memory requirements. A UTF32 string needs four times more space than an 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 halves the space required by UTF32. It is possible to store a *codepoint* per element as long as we do not leave the 0 plane (BMP). For higher planes, two UTF16 elements (32bits) will be necessary. This mechanism, which encapsulates the higher planes within the BMP, is known as **surrogate pairs**.

```
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;
```

```
}
```

To iterate over a UTF16 string that contains characters from any plane, it must be used `unicode_next`.

13.4.4. UTF-8

UTF8 is a variable length code where each *codepoint* uses 1, 2, 3 or 4 bytes.

- 1 byte **(0-7F)**: the 128 symbols of the original ASCII. This is a great advantage, since US-ASCII strings are valid UTF8 strings, without the need for conversion.
- 2 bytes **(80-7FF)**: Diacritical and Romance language characters, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac and Thaana, among others. A total of 1920 *codepoints*.
- 3 bytes **(800-FFFF)**: Rest of the plane 0 (BMP).
- 4 bytes **(10000-10FFFF)**: Higher planes (1-16).

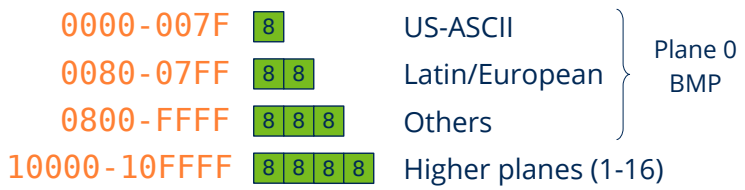


Figure 13.11: Each character in UTF8 uses 1, 2, 3 or 4 bytes.

More than 90% of websites use UTF8 (august of 2018[1]), because it is the most optimal in terms of memory and network transmission speed. As a disadvantage, it has associated a small computational cost to encode/decode, since it is necessary to perform bit-level operations to obtain the *codepoints*. It is also not possible to randomly access a specific character by index, we have to process the entire string.

```
const char_t code1[] = "Hello";
const char_t code2[] = "áéíóú";
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);
}
```

[1] https://w3techs.com/technologies/overview/character_encoding/all

13.4.5. Using UTF-8

UTF8 is the encoding required by all the NAppGUI SDK functions. The reasons why we have chosen UTF-8 over other encodings have been:

- It is the natural evolution of the US-ASCII.
- The applications will be directly compatible with the vast majority of Internet services (JSON/XML).
- In multi-lingual environments the texts will occupy less space. Statistically, the 128 ASCII characters are the most used on average and only need one byte in UTF8.
- As a disadvantage, in applications aimed exclusively at the Asian market (China, Japan, Korea - CJK), UTF8 is less efficient than UTF16.

Within NAppGUI applications they can cohexist different representations (`char16_t`, `char32_t`, `wchar_t`). However, we **strongly recommend the use of UTF8** in favor of portability and to avoid constant conversions within the API. To convert any string to UTF8 the `unicode_convers` function is used.

```
wchar_t text[] = L"My label text.";
char_t ctext[128];
unicode_convers((const char_t*)text, ctext, ekUTF16, ekUTF8, 128);
```

NAppGUI does not offer support for converting pages from Extended ASCII to Unicode.

The `Stream` object provides automatic UTF conversions when reading or writing to I/O channels using the methods `stm_set_write_utf` and `stm_set_read_utf`. It is also possible to work with the `String` type (dynamic strings), which incorporates a multitude of functions optimized for the UTF8 treatment. We can include constant text strings directly in the source code (Figure 13.12), although the usual thing will be to write them in resource files (*"Resources"* (page 129)). Obviously, we must save both the source and resource files in UTF8. All current development environments support the option:

- By default, Visual Studio saves the source files in ASCII format (Windows 1252). To change to UTF8, go to `File->Save As->Save with encoding->Unicode (UTF8 Without Signature)- Codepage 65001`. There is no way to set this configuration for the entire project :-(.
- In Xcode it is possible to establish a global configuration. `Preferences->Text editing->Default Text Encoding->Unicode (UTF-8)`.
- In Eclipse it also allows a global configuration. `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]);
```

Figure 13.12: UTF8 constants in a C source file.

13.5. Maths

`BMath` offers a compact interface on the elementary mathematical functions of the C standard library. It also defines some of the most used constants, such as the number Pi, conversions between degrees and radians or the root of 2.

- Use `bmath_cosf` to calculate the cosine of an angle (*wrapper* over cstdlib `cosf()`).
- Use `bmath_sqrtf` to calculate the square root (*wrapper* over cstdlib `sqrtf()`).

13.5.1. Random numbers

`BMath` includes a seed-based pseudo-random number generator. From the same seed, the sequence of numbers generated will always be the same. The sequences produced by two different seeds will be radically disparate. Hence they are called pseudo-random.

- Use `bmath_rand_seed` to set the random number seed.
- Use `bmath_randf` to get a random floating point number, within an interval.

In the case of multi-threaded applications, this sequence may vary depending on the order of execution of the threads, since these functions **are not re-entrant**. You must use an "environment" of random numbers for each thread in question, in case you need to always ensure the same sequence (deterministic algorithms).

- Use `bmath_rand_env` to create a random number safe environment.
- Use `bmath_rand_mtf` to get a random number from an environment.

13.6. Standard functions

`BLib` includes useful functions from the C standard library that don't fit in other modules like `BMath` or `BMem`. As in `<stdlib.h>` we find text conversion functions, algorithms

or interaction with the environment.

- Use `blib_strcmp` to compare two text strings.
- Use `blib_qsort` to sort a vector of elements.
- Use `blib_bsearch` to perform a dichotomous search on an ordered vector.
- Use `blib_abort` to end program execution.

13.7. Standard I/O

All processes have input and output channels by default, without the need to create them explicitly. By channels we mean *streams* or data flows.

- Use `bstd_printf` to write text to standard output.
- Use `bstd_read` to read bytes from standard input.

Each running process has three standard communication channels:

- **stdin:** data input. The process will read data that comes from outside.
- **stdout:** data output. The process will write results on this channel.
- **stderr:** error output. The process will write on this channel information regarding errors.

It's like having three perpetually open files where the program can read and write without limits. When we execute a process from the Console or the Terminal, `stdin` automatically connects to the keyboard and `stdout`/`stderr` to the screen (Figure 13.13). However, these standard channels can be redirected to use files as input sources or output destinations:

```
dir > out.txt
ls > out.txt
sort < out.txt
```

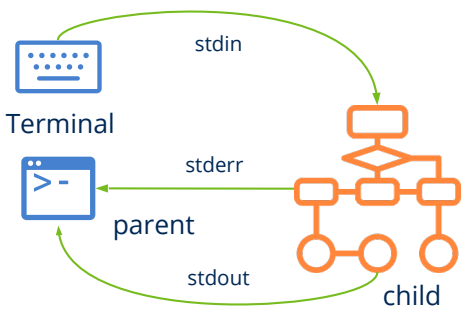


Figure 13.13: Executing a process from the Terminal.

In this code snippet, the result of the command `dir` (`ls` in Unix) has been redirected to the file `out.txt`, so we will not see anything on the screen. On the other hand, the command `sort` it does not wait for the user to enter through the keyboard. Simply taking the file `out.txt`, sorting its lines. Therefore, whenever we write applications on the command line, we should conveniently use these standard channels without making presumptions from where the information processed by the application comes from or where it goes.

13.8. Memory

From the programmer perspective, access to memory is done through variables and manipulated through the language operators (+, -, *, =, ...) and always in the same way, regardless of how the variables were created or in what memory zone they are hosted. Within `bmem.h` we have several functions to make copies, assignments or checks of generic memory blocks. This module also defines functions for dynamic memory manipulation *(Heap)*.

- Use `bmem_malloc` to reserve a dynamic memory block.
- Use `bmem_free` to free a block of dynamic memory.
- Use `bmem_copy` to copy the contents of two memory blocks, previously reserved.

13.8.1. Stack Segment

The memory of a compiled and running C program is divided into several segments. One of them is the *stack*, a space of variable but limited size, where local variables and function calls *(call stack)* are stored. It grows and shrinks as the process enters and leaves areas or functions (Figure 13.14). It is automatically managed by the compiler as a LIFO *Last-in First-out structure*, so it goes unnoticed most of the time, since it does not require extra attention from the programmer. We are aware of its existence when receiving the *Stack Overflow* error, usually caused by infinite recursion or the reservation of very large C vectors (Listing 13.1). The debugger allows us to inspect the state of the stack at execution time *"Debugging the programDebugging the program"* (page 63).

Listing 13.1: Two simple cases that cause the stack overflow.

```
int func(int n) { func(n); } // Stack Overflow

float v[2000000]; // Stack Overflow
```

While the use of the *stack* is ideal due to its high performance, security and ease of use, sometimes falls short. On the one hand, it is necessary to foresee in the design time the amount of memory needed and define it statically (eg. `struct Product pr[100];`),

```
void func(int a, int b)
{
   int i;
   char str[10];

   i = 5;
(b) str[0] = 'A';
}

int main()
{
(a) int g = 5;
   func(1, 2);
(c) g = 10;
}
```

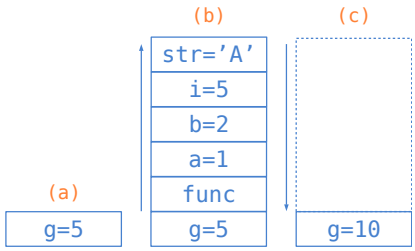


Figure 13.14: *Stack* state in different points of the program.

something very inflexible when it comes to building real applications. On the other hand, variables are destroyed when closing a scope or leaving a function, which prevents sharing data globally.

13.8.2. Heap Segment

The *heap* is a memory zone that the process can request on demand, through calls to the system. It is complementary to the *stack* and is characterized by:

- It can be accessed globally, from any point of the program through a pointer.
- The amount of available memory is practically unlimited.
- It is less efficient than the *stack*.
- Requires management. Operating systems provide functions for requesting dynamic memory blocks (`HeapAlloc()`, `sbrk()`), being the responsibility of the process, or rather the programmer, to release these blocks when they are no longer needed.

As allocations and de-allocations can be made in any order, internal fragmentation occurs as the program progresses (Figure 13.15). Here would come into play the so-called **memory manager**, which are algorithms that allow optimizing the use of the heap by compacting it and reusing the released blocks. The standard C library provides the familiar functions `malloc()/free()`, which implement a generic memory manager through system calls.

NAppGUI implements its own dynamic memory manager/auditor "Heap - Memory manager" (page 188) very optimized to serve numerous requests of small size, which is what applications demand normally. `bmem_malloc/bmem_free` *connect to the operating system through system calls and should not be used directly.*

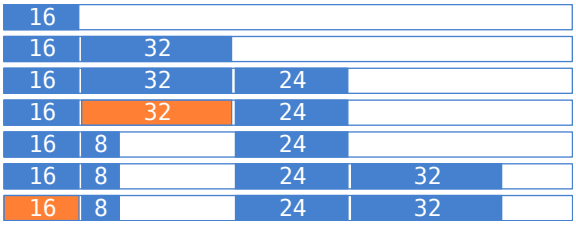


Figure 13.15: Fragmentation of the *heap* during the execution of the process.

Chapter 14

Osbs library

There is no neat distinction between operating system software and the software that runs on top of it.

Jim Allchin

14.1. Osbs

osbs *(Operating System Basic Services)* is a portable wrapper that allows applications to communicate with the operating system core at the level of processes, memory, files and networks. This communication is carried out through a series of **system calls** (Figure 14.1) which vary according to the operating system for which we are programming. It is the non-graphic lowest level API to communicate with hardware devices and access the machine resources. Below are the device drivers managed directly by the kernel, to which applications have access denied.

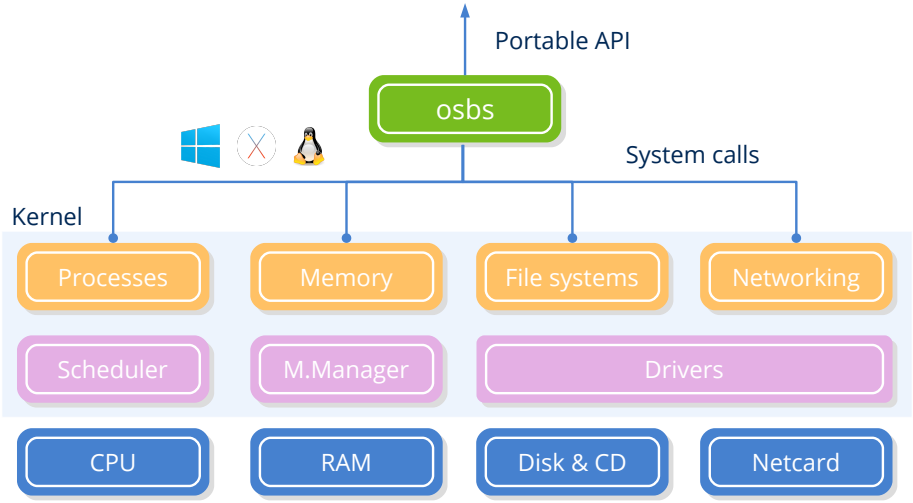


Figure 14.1: System calls are the gateway to the operating system kernel.

Darwin, the macOS kernel, and Linux are Unix-like systems, therefore, they share the same system calls (with subtle differences). But Windows presents a radically different architecture and function set. The NAppGUI **osbs** library is nothing more than a small wrapper that internally handles these differences and provides a common way to access the same resources on different platforms (Figure 14.2). It only depends on *"Sewer"* (page 149) and its functionalities have been divided into different modules:

- *"Processes"* (page 167), *"Threads"* (page 170), *"Mutual exclusion"* (page 175).
- *"Loading libraries"* (page 175).
- *"Files and directories"* (page 177).

- *"Sockets"* (page 179).
- *"Time"* (page 183).

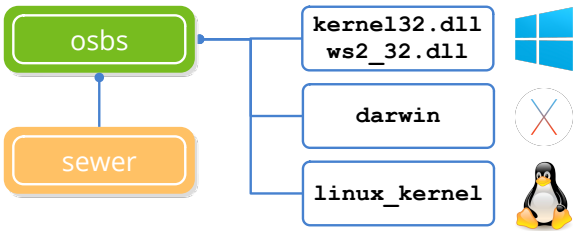


Figure 14.2: *osbs* dependencies. See *"NAppGUI API"* (page 145).

14.2. Processes

From the programmer perspective, multi-processing is the possibility of launching and interacting with other processes (children) from the main process (parent). The operating system can execute the child process in another CPU core *(true multitasking)* or in the same as the parent *(context switch)*. This is a system decision in which the programmer can not influence and will depend on the processor type and its workload. The final effect will be that both processes (parent and child) run in parallel.

- Use `bproc_exec` to launch a new process from the application itself.
- Use `bproc_read` to read from the standard output of the process.
- Use `bproc_write` to write to the standard input of the process.

14.2.1. Launching processes

`bproc_exec` will launch a process from our own C program in a similar way as the Terminal does (Figure 14.3). In this case, the *"Standard I/O"* (page 161) `stdin`, `stdout` and `stderr` will be redirected to the `Proc` object through anonymous pipes. From here, we can use `bproc_write` to write on the son `stdin` channel and `bproc_read` to read from his `stdout`. The rules of reading/writing are those that govern the operating system *pipes* and that we can summarize in:

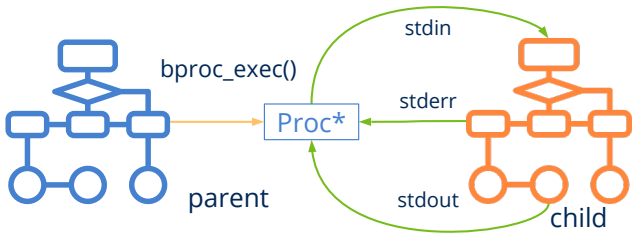


Figure 14.3: Launching a process from our own C code.

- If the parent calls `bproc_read` and the child has not written anything (empty buffer), the parent will be blocked (wait) until there is information in the child's output channel.
- If the child ends and parent is waiting to read, `bproc_read` will return `FALSE` and the parent will continue his execution.
- If the parent calls `bproc_write` and the writing buffer is full, the parent will block (wait) until the child reads from his `stdin` and free space in the channel.
- If the child ends and the father is blocked by writing, `bproc_write` will return `FALSE` and the parent will continue his execution.
- Some commands or processes (eg `sort`) will not start until reading the entire `stdin` contents. In these cases, the parent process must use `bproc_write_close` to indicate to the child that the writing on his `stdin` has finished.
- When the parent calls `bproc_close`, all the I/O channels will be closed and both processes will continue their execution independently. To finish the execution of the child process *(kill)* use `bproc_cancel`.
- `bproc_wait` will stop the parent process until the child completes. To avoid overloading the child output buffer `stdout`, close the channel through `bproc_read_close`.
- `bproc_finish` will check, in a non-blocking way, if the child has finished running.

14.2.2. Multi-processing examples

Let's look at some practical examples of IPC *Inter-Process Communication* using the standard I/O channels in linked parent-child processes. In (Listing 14.1) we will dump the child process `stdout` output in a file. In (Listing 14.2) we will redirect both channels, we will write in `stdin` and we will read from `stdout` using disk files. Finally, we will implement an asynchronous protocol where the parent and child exchange requests and responses. In (Listing 14.4) we show the code of the child process, in (Listing 14.3) the parent process and in (Listing 14.5) the result of the communication, written by the parent process.

Listing 14.1: Reading from a process `stdout` and saving it in a file.

```
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);
```

The shell commands are not portable in general. We use them only as an example.

Listing 14.2: Redirecting the `stdin` and `stdout` of a process.

```
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);
```

Listing 14.3: Asynchronous protocol (parent process).

```
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);
```

Listing 14.4: Asynchronous protocol (child process).

```
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;
        }
    }
}
```

Listing 14.5: Parent process execution result.

```
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. Threads

The **threads** are different execution paths within the same process (Figure 14.4). They are also known as **light processes**, since they are more agile to create and manage than the processes themselves. They share code and memory space with the main program, so it is very easy to exchange information between them through memory variables. A thread starts its execution in a method known as *thread_main* and, at the moment it is launched, it runs in parallel with the main thread. Like the processes, they are objects controlled by the core of the system that will dictate, ultimately, whether the threads will be executed

in another CPU core *(true multitasking)* or will share it *(context switch)*.

- Use `bthread_create` to create a new thread.
- Use `bthread_wait` to force the main thread to wait for the thread to execute.

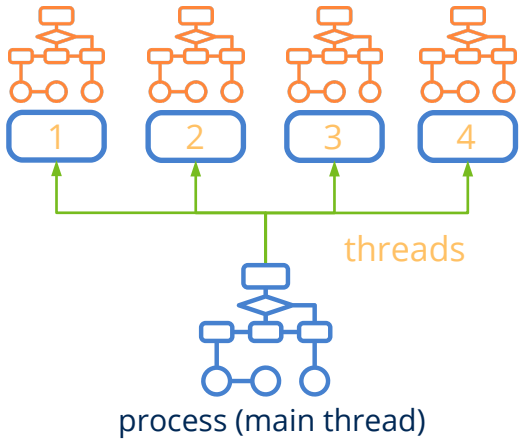


Figure 14.4: A process with multiple execution threads.

14.3.1. Throwing threads

Each call to `bthread_create` will create a new thread in parallel starting at the function passed as a parameter *(thread_main)*. The "natural" way to end it is by returning from *thread_main*, although it is possible to abort it from the main thread.

Basic code to launch a parallel execution thread.

```
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. Shared variables

Each new thread has its own *"Stack SegmentStack Segment"* (page 162) therefore, all automatic variables, function calls and dynamic allocations will be private to said thread. But it can also receive global data from the process through the *thread_main* `data` parameter. We must be careful when accessing global data through multiple concurrent

threads, since modifications made by other threads can alter the logical code execution, producing errors that are very difficult to debug. The program (Listing 14.6) is correct for single-thread programs, but if the variable `vector` is accessed by two simultaneous threads, can lead to a *Segmentatin Fault* error if thread-1 frees memory while thread-2 is executing the loop.

Listing 14.6: Dangerous access to shared variables.

```
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;
}
```

To avoid this problem, we will have to protect the access to shared variables through a `Mutex` (Listing 14.7). This *"Mutual exclusion"* (page 175) mechanism guarantees that only one thread can access the resource in a moment of time. A thread will be stopped if it intends to execute the code located between `bmutex_lock` and `bmutex_unlock` if another thread is within this *critical section*.

Listing 14.7: Secure access to shared variables.

```
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. Multi-thread example

The tricky part of multi-threaded programming is to decompose a solution into parts that can run in parallel and organize the data structures so that this can be carried out in the most balanced way possible. In (Listing 14.8) the program will run four times faster (x4) since a perfect division of the problem has been made (Figure 14.5). This is just a theoretical example and this result will be very difficult to achieve in real situations. We must also minimize the number of shared variables and the time of the critical sections, otherwise the possible inter-blocks will reduce the gain.

Listing 14.8: Multi-threaded processing of a very large vector.

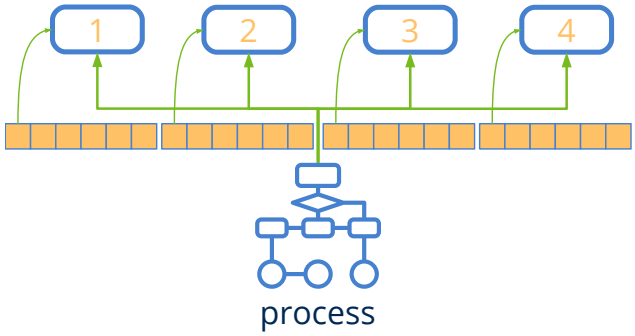


Figure 14.5: Collaboration of four threads in a vector calculation.

```
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);
```

Listing 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. Mutual exclusion

In processes with multiple threads, mutual exclusion guarantees that only one of them can execute a **critical section** at a specific moment of time. The critical section is a block of code that normally protects a shared resource that does not support concurrent access.

- Use `bmutex_create` to create a lock.
- Use `bmutex_lock` to lock a critical section.
- Use `bmutex_unlock` to unlock a critical section.

14.4.1. Locks

Locks or `Mutex` are synchronization objects managed by the operating system that mark the beginning and end of a critical section (Figure 14.6). When a thread is going to access a certain share, you must call the method `bmutex_lock` to guarantee exclusive access. If another thread is using the resource (it has previously called `bmutex_lock`), the current thread will stop until the resource is released through `bmutex_unlock`. Blocking and unblocking threads is handled by the operating system itself. The programmer should only worry about identifying and protecting the critical sections. *"Multi-thread exampleMulti-thread example"* (page 172).

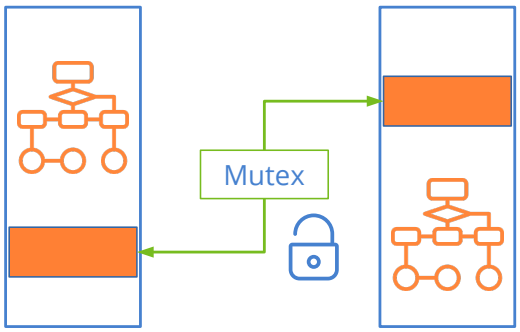


Figure 14.6: A mutex protecting the critical sections of two threads, which can not be executed concurrently. The rest of the code can run in parallel.

14.5. Loading libraries

The usual, in projects of relative size, is to divide the program code into libraries in order to be able to reuse them in different projects. The link of these libraries within the final executable can be done in three ways:

- **Compile time:** The library code is copied into the executable, forming an inseparable part of it (static libraries) (Figure 14.7) (a).
- **Load time:** The library code is distributed separately (dynamic libraries) and is loaded together with the main program, at the same time (Figure 14.7) (b).

- **Runtime:** Dynamic libraries that the program loads when it needs them (Figure 14.7) (c).

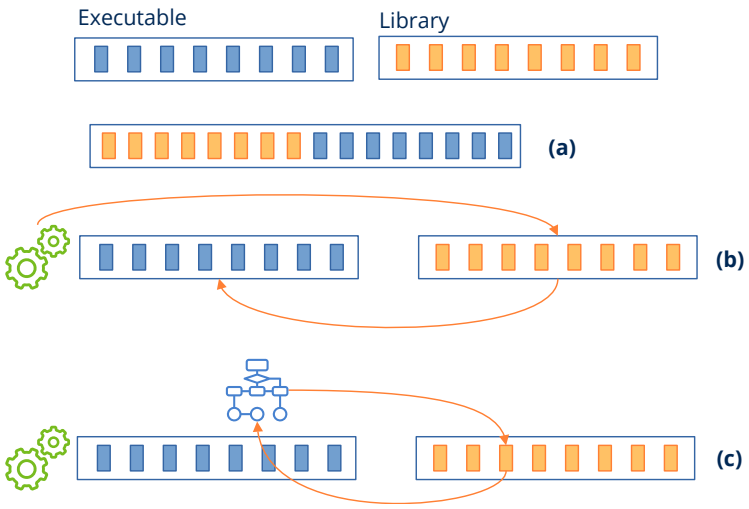


Figure 14.7: Library link and dynamic loading.

The linking process is relatively complicated and is handled automatically by the compiler and operating system's loader. The programmer should only intervene in the third case, since it is necessary to include code to load the libraries and access the appropriate methods or variables at all times.

- Use `dlib_open` to load a library at runtime.
- Use `dlib_proc` to get a pointer to a library function.
- Use `dlib_var` to get a pointer to a library variable.

14.5.1. Library search paths

A dynamic library is in a different file than the executables that can make use of it. Each operating system implements different search strategies that we must know to install and/or configure the programs correctly.

14.5.2. Search order in Windows

- Directory `path` of `dlib_open`.
- The same directory as the executable.
- The current directory `bfile_dir_work`.
- Directory `%SystemRoot%\System32`.

- Directory `%SystemRoot%`.
- The directories specified in the environment variable `PATH`.

14.5.3. Search order on Linux/macOS

- The directories specified in the environment variable `LD_LIBRARY_PATH` (Linux) or `DYLD_LIBRARY_PATH` (macOS).
- The directories specified in the executable `rpath`.
- System directories `/lib, /usr/lib, etc`.

14.6. Files and directories

14.6.1. File System

The file system *(filesystem)* is the hierarchical structure composed of directories and files that allows organizing the persistent data of the computer (Figure 14.8). It is something with which computer users are very familiar, especially after the emergence of graphic systems that introduced the analogy of desktop, folder and document. It starts in a directory called root (`/` on Unix or `C:\` on Windows) and, from here, all sub-directories and files hang down forming a tree that grows deep. At the programming level, the file system is managed through system calls that allow directories to be created, browse their content, open files, delete them, obtain attributes, etc.

- Use `bfile_create` to create a new file.
- Use `bfile_dir_create` to create a directory.
- Use `bfile_dir_open` to open a directory to explore its contents.
- Use `bfile_dir_get` to get information about a directory entry.

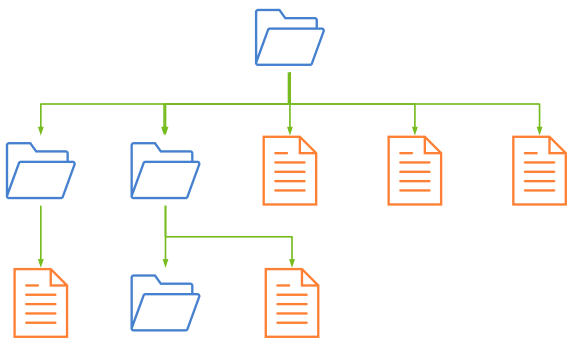


Figure 14.8: Typical structure of a file system.

14.6.2. Files and data streams

A process can read or write data to a file after opening an I/O (*"Streams"* (page 193)) which provides a stream of binary data to or from the process itself (Figure 14.9). There is a pointer that moves sequentially each time data is read or written. It is initially in byte 0, but we can modify it to access random positions in the file without reading the content (Figure 14.10). This can be very useful when working with large files whose data is indexed in some way.

- Use `bfile_open` to open an existing file.
- Use `bfile_read` to read binary data from a file.
- Use `bfile_write` to write binary data to a file.
- Use `bfile_seek` to modify the file pointer.

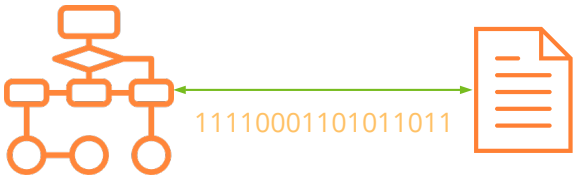


Figure 14.9: After opening a file, the process has an I/O channel to read or write data.

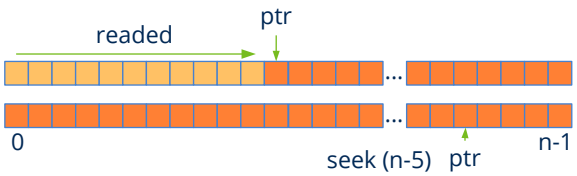


Figure 14.10: Sequential read or random access.

14.6.3. Filename and pathname

These two concepts are recurrent and widely used by API functions that manipulate files. When we navigate through the contents of a directory `bfile_dir_get`, we obtain a sequence of *filenames* that is the "flat" name of the element (file or subdirectory) without including its path within the file system (without characters `'/'` or `'\'`). On the other hand the *pathname* is a sequence of one or several *filenames* separated by `'/', '\'`, which indicates the way forward to locate a certain element. This path can be **absolute** when it starts with the root directory (`C:\Users\john\docs\images\party.png`) or **relative** (`docs\images\party.png`) when it indicates the partial route from the process current *working directory*.

- Use `bfile_dir_work` to get the current working directory.
- Use `bfile_dir_set_work` to set the working directory.

14.6.4. Home and AppData

These are two typical directories used by applications to store files relative to a particular user. On the one hand, *home* indicates the personal directory of the user currently registered in the system, typically `C:\Users\john` (Windows), `/home/john` (Linux) or `/Users/john` (macOS). On the other hand *appdata* is a directory reserved for saving temporary or configuration data of applications. Typical locations can be `C:\Users\john\AppData\Roaming` (Windows), `/home/john/.config` (Linux) or `/User/john/Library` (macOS). The usual thing will be to create a sub-folder with the name of the application `/User/john/Library/TheApp`.

- Use `bfile_dir_home` to get the user home directory.
- Use `bfile_dir_data` to get the application data directory.
- Use `bfile_dir_exec` to get the current executable directory.

14.7. Sockets

We can define a **socket** as a communication channel between two processes that are running on different machines. They use as a base the family of TCP/IP protocols that govern Internet communication from the first prototypes of the big network back in 1969. For its part, the IP protocol *(Internet Protocol)* is responsible for sending small data packets between two remote computers through the network. As there are packets that can be lost or take different paths when crossing the Internet nodes, TCP *(Transmission Control Protocol)* will be in charge of sorting them sequentially and re-ordering those that have been lost. Another important aspect that TCP adds is the concept of a **port**, which allows the same machine to have multiple connections open at the same time. The conjunction of TCP/IP provides the process of a reliable bidirectional communication channel *(full-duplex)* with the remote process and is the basis of the client/server model (Figure 14.11).

- Use `bsocket_connect` in the client process to create a communication channel with a remote server.
- Use `bsocket_server` in the server process to listen for client requests.
- Use `bsocket_accept` to accept a client's request and start communication.
- Use `bsocket_read` to read data from a socket.
- Use `bsocket_write` to write data to a socket.

Sockets are the lowest-level communication primitive accessible by applications. They are extremely fast but, in general, their functions are blocking, that is, they will stop the process until the other party responds.

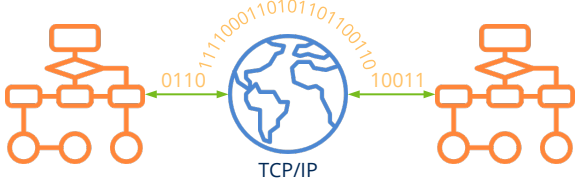


Figure 14.11: TCP/IP sockets allow two processes to be connected through the Internet.

- `bsocket_connect` will stop the client process until the server responds or the `timeout` expires.
- `bsocket_accept` it will stop the server process until a request from a client arrives or the `timeout` is fulfilled.
- `bsocket_read` will stop the process until the other interlocutor writes data to the channel or the `timeout` is fulfilled.
- `bsocket_write` will stop the process until the other peer reads data from the channel and frees the intermediate buffer or the `timeout` is fulfilled.

Apart from these indications, working with *sockets* is very similar to working with files on disk. The TCP/IP implementation is complicated and is part of the operating system, so the establishment of the connection has been simplified through the system calls seen above. Since a socket only allows sending and receiving bytes, both partners need to define a **protocol** that indicates the order, sequence and type of data to be shared in such a way that communication is satisfactory and free of deadlocks. Some of the most used protocols on the Internet are: HTTP, SMTP, FTP, SSH, etc.

14.7.1. Client/Server example

As an example we are going to see how two processes exchange information through sockets. The protocol is extremely simple. After connection, the client (Listing 14.11) will send a series of numerical values to the server (Listing 14.10) and it will respond by resending the same value. When the client sends the value `UINT32_MAX` the communication will end.

Listing 14.10: Simple socket-based server.

```
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);
```

Listing 14.11: Client process.

```
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. Time

The operating system measures the passage of time using an internal clock, typically implemented by a counter of the *ticks* that have passed since an initial moment called *epoch*. In Unix-like systems this counter represents the number of seconds elapsed since January 1, 1970 UTC. However, in Windows it represents the number of 100 nanosecond intervals since January 1, 1601 coinciding with the beginning of the Gregorian calendar. In NAppGUI these values have been unified to work with *Unix Epoch* on all platforms.

- Use `btime_now` to get the number of micro-seconds elapsed since January 1, 1970 UTC.
- Use `btime_date` to get the system date.
- Use `btime_to_micro` and `btime_to_date` to convert dates to Unix Time and vice versa.

The difference between two instants will give us the time elapsed during the execution of a task.

```
uint64_t ed, st = btime_now();

// Do something...
...
```

Figure 14.12: Unix Epoch Instant 0.

```
ed = btime_now();
bstd_printf("Total elapsed micro-seconds: %lu\n", ed - st);
```

14.9. Log

A *log* or diary is a record of anomalies that occur at runtime and that help to further debug the program or determine the cause of an error (Figure 14.13). This report is aimed more at programmers or software administrators and not at the end user, so it is advisable to include specific technical information on the cause of the problem. The messages addressed to the end user must be written in a more friendly tone, far from technicalities and sent to the standard output (`stdout stderr`) or to the window system, if we are facing a desktop application.

- Use `log_printf` to write a message to the execution log.

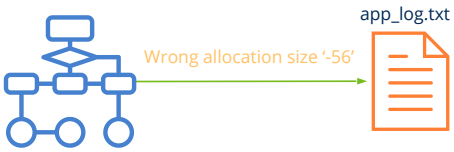


Figure 14.13: Messages related to internal anomalies of the program, can be sent to a *log*.

Chapter 15

Core library

A strong core will improve your technique, strength, and stamina, and compliment everything you do.

Susan Trainor

15.1. Core

Just as a building needs a strong foundation, any application or library must be based on robust and efficient pillars. It is useless to invest hours and hours in a nice interface if the internal engine is broken. For this purpose, the *core* library has been developed (Figure 15.1). Provides structures, utilities and algorithms commonly used in programming, which will facilitate the program development guaranteeing maximum efficiency and portability. *Core* is the third level within the NAppGUI SDK and still has no knowledge about the operating system graphics capabilities, so it can be used to implement any kind of project.

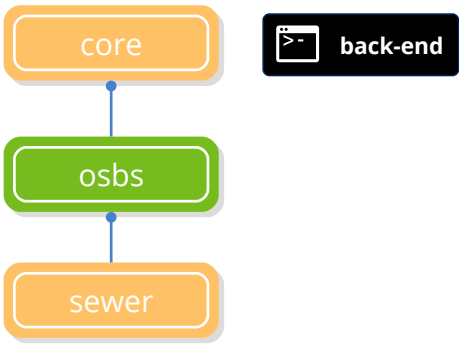


Figure 15.1: *core* dependencies. See *"NAppGUI API"* (page 145).

The services provided by *core* have been divided into several modules.

- *"Heap - Memory manager"* (page 188).
- *"Buffers"* (page 192).
- *"Strings"* (page 192).
- *"Streams"* (page 193).
- *"Arrays"* (page 208).
- *"Binary search trees"* (page 217).
- *"Regular expressions"* (page 222).
- *"Data binding"* (page 225).
- *"Events"* (page 230).
- *"File operations"* (page 231).
- *"Clocks"* (page 233).

15.2. Heap - Memory manager

Heap is a very efficient dynamic memory manager and auditor included in the *core* library and available for all projects based on NAppGUI (libraries and applications). It is common for applications to request a large number of small memory blocks to hold different objects (character strings, interface controls, structure instances, I/O buffer, etc). The strategy behind this manager is just to ask the operating system for memory pages of a certain size (64kb or more) using `bmem_malloc` and use them to solve several requests very efficiently.

- Use `heap_new` to dynamically create an object.
- Use `heap_malloc` to reserve a memory block.
- Use `heap_delete` to destroy an object.
- Use `heap_free` to free up a memory block.

```
Product *product = heap_new(Product);
byte_t *memblock = heap_malloc(1024, "MyOwnBlock");

// Do something
...

heap_delete(&product, Product);
heap_free(&memblock, "MyOwnBlock");
```

Using **Heap** instead of system calls will provide us with certain benefits:

- Performance: A call to `heap_malloc` is solved only by increasing the value of a counter. `heap_free` it only updates the header of the affected page.
- Locality: Two consecutive calls to `heap_malloc()` are located in contiguous physical memory positions. This reduces the number of cache failures because, according to the locality principle, there is a high probability that two objects that are created together will be used together.
- Memory leaks: *heap* points reservations and releases by object type. If necessary, will notify the programmer through *"Asserts"* (page 153) or *"Log"* (page 184) that there are objects not released. The great advantage of this auditor over other tools is that it is always being executed as part of the program. This exploits the temporal coherence, because if after a program change *leaks* are detected where there was not before, it is very likely that we can limit and detect the error, since it will be something we have just worked on.
- Statistics: We can obtain memory usage profiles (time/bytes). This can help us detect bottlenecks (especially at startup) or optimize page size.

15.2.1. Multi-thread memory

By default, *heap* is configured to work optimally in single-threaded applications. If we want several threads of the same process to reserve or release dynamic memory concurrently and safely, we must use:

- `heap_start_mt` to start multi-thread support.
- `heap_end_mt` to end multi-thread support.

The moment `heap_start_mt` is called, the synchronization mechanisms within the heap are activated to guarantee mutual exclusion to the memory manager until a call to `heap_end_mt` is received which will return to single-threaded operation mode. Successive calls to `heap_start_mt` will accumulate, so it will remain in multi-threaded mode until all open blocks are closed (Listing 15.1). It is the responsibility of the programmer to use this pair of functions at those points of the program that require it.

Any section that begins with `heap_start_mt` must be closed with `heap_end_mt`.

There is no problem in activating multi-threaded support in single-threaded sections, except for a slight performance penalty.

Listing 15.1: Multi-thread sections.

```
// 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. How Heap Works

When a program starts, *heap* creates a default memory page. The first bytes are reserved as a header, a small structure that controls the state of the page. Each request is assigned sequentially within the same page, increasing the value of a pointer (Figure 15.2). When

the page runs out of space, a new one is created `bmem_malloc`, which is linked to the previous one and labeled as the new **default page** (Figure 15.3). Each call to `heap_free` update the header with the number of blocks/bytes released (Figure 15.4). These blocks **are not reused**, otherwise the logic of *heap* would be complicated by slowing it down. The address of the header is stored at the end of each block, so do not have to iterate to locate it. When all the blocks on the page have been released, the entire page is destroyed by `bmem_free` and the pointers between neighboring pages restored (Figure 15.5).

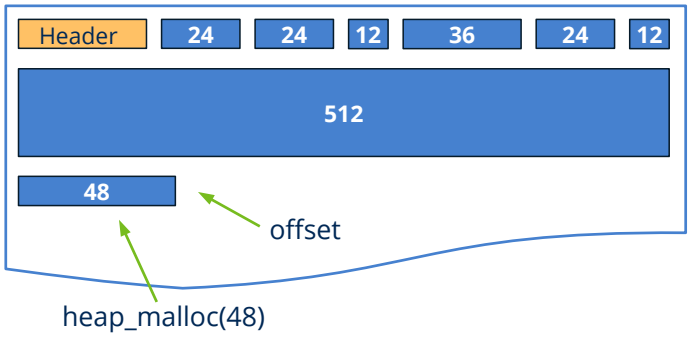


Figure 15.2: Reserve a new memory block with `heap_malloc()`.

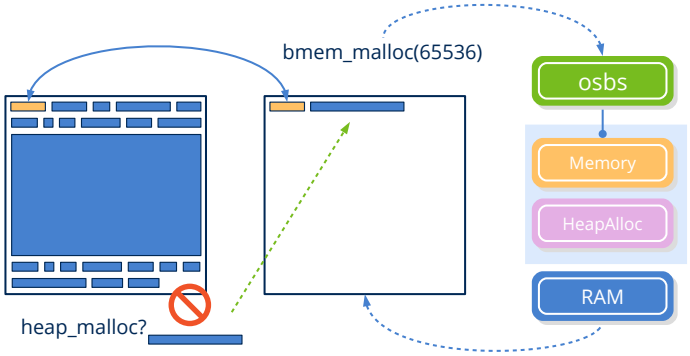


Figure 15.3: Request to the operating system of a new empty page.

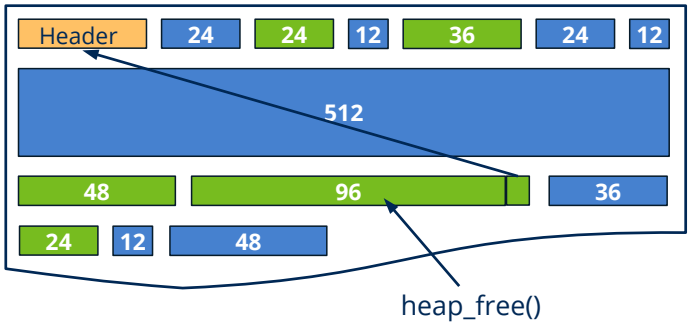


Figure 15.4: Releasing a block of memory (only updates the header).

Heap also counts the number of alloc/dealloc per object type using the parameter `name` of `heap_malloc`. At the end of the execution of the program, if the application lacks *memory leaks*, it will write in *"Log"* (page 184) a message like this:

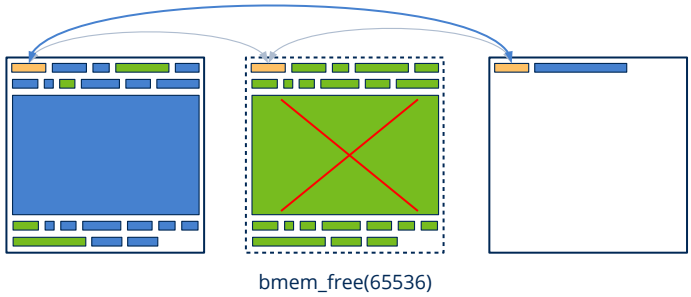


Figure 15.5: Destroying the entire page.

```
[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
[12:58:08] Effective reallocations: (0/34)
[12:58:08] Real allocations: 2 pages of 65536 bytes
[12:58:08] ============================
```

But if after the execution, the application has memory to be released, the message will be different:

```
[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] ==================================
```

That warns that we have an object `App` and four `String` without releasing. If in the previous execution there were no *leaks*, it is very likely that we can narrow the error without too much difficulty.

The heap auditor does not intend to replace more advanced memory testing tools, it is only a first filter that constantly alerts us during the development and test phase. Although the overhead that occurs at runtime is minimal, the auditor is completely disabled in the `Release` *configuration.*

15.3. Buffers

Buffer objects are simply dynamically stored memory blocks and stored in the *"Heap SegmentHeap Segment"* (page 163). They are useful for sharing generic data between different functions or threads. For the latter case, they must be protected by a `Mutex` if several threads can access it concurrently (they are not *thread-safe*). They are of fixed size. Once created, they can not be resized, although they can be rewritten as many times as necessary.

- Use `buffer_create` to create a dynamic memory block.
- Use `buffer_destroy` to free up a block of dynamic memory.
- Use `buffer_data` to get a pointer to the memory block.

15.4. Strings

String objects contain *"UTF-8UTF-8"* (page 158) characters strings dynamically allocated. Although it is possible to insert static text strings directly into the source code or access them through the resource packages (`respack_text`), it is usually necessary to compose texts at runtime or dynamically store strings received by some input channel (keyboard, files, network, etc). The NAppGUI `strings.h` module offers a multitude of functions to work with UTF8 text strings, both static and dynamic.

- Use `str_c` to create a dynamic copy of a static C string.
- Use `str_printf` to compose a dynamic string using the same format as C `printf`.
- Use `tc` to get a `const char_t*` pointer to the content of a `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."
```

Do not confuse `String` objects with C strings `const char_t *str` or `char_t str[128]`. The first ones contain a pointer to the dynamic memory area and an integer with the number of reserved bytes.

In the case that it is necessary to create more extensive texts from loops, the most efficient way is to create a `Stream` and, later, obtain the associated `String`.

```
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

A *stream* is a data flow that runs from a source to a destination. Think of a phone call. We have an origin (the person who speaks), a destination (the person who listens) and a channel (the line itself). In programming, the stream is the equivalent to the telephone line, it is the pipe that joins the application with a data source or destination (Figure 15.6) and through which binary information, bit sequences, run. As with any other communication channel, the information is volatile, available for a very limited time. Once it reaches the receiver, it disappears.

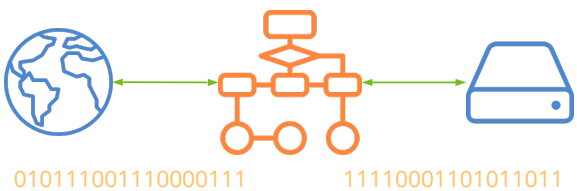


Figure 15.6: Streams connect the process with the machine and the world.

In essence, there are three elementary operations to perform when working with streams: Create the channel, read data and write data.

- Use `stm_memory` to create a read/write memory stream.
- Use `stm_read_r32` to read a `float` from the stream.
- Use `stm_write_r32` to write a `float` to the stream.
- Use `stm_close` to close the channel and free up resources (destructor).

15.5.1. Stream Types

Actually, it is more correct to talk about types of extremes (origin and destination) than of stream types. From the perspective of the programmer, a stream is an abstract

type that presents the same functionality regardless of the ends it connects. Therefore, when talking about *stream types* we are referring to the type of constructor.

15.5.2. File stream

In *File streams* (Figure 15.7), the source is the process memory and the destination is a disk file. The opposite can also happen: that the source is the file and the destination the memory, it will depend on how we create the channel. It will not be possible to perform write operations on an open file for reading or vice versa (Listing 15.2). *"Files and directories"* (page 177).

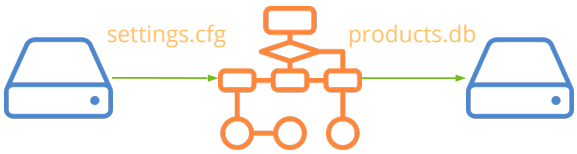


Figure 15.7: *File streams* allow communication with the file system.

- Use `stm_from_file` to open a file and read from it.
- Use `stm_to_file` to create a file and write to it.
- Use `stm_append_file` to add content to an existing file.

Listing 15.2: Example of writing to a file.

```
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

A *socket* is a communication channel between two processes over the Internet (Figure 15.8). Unlike *file streams*, sockets allow bidirectional *full-duplex)* communication, that is, both ends can send and receive information. The sequence of message exchange between partners is determined by the protocol (Listing 15.3), being HTTP, FTP, SMTP or LDAP some of the most used for Internet transmissions. See *"Sockets"* (page 179).

- Use `stm_socket` to create a communication channel with a remote process.

Listing 15.3: Downloading a web page, using the HTTP protocol.

Figure 15.8: A *socket stream* opens a communication channel over the Internet.

```
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

Block streams are used to read formatted data from a generic memory block. (Figure 15.9). This memory area is considered read-only and will not be modified, so write operations will not be allowed in this type of stream. When the end of the block is reached, the `ekSTEND` state will be activated.

- Use `stm_from_block` to read data from a memory block.

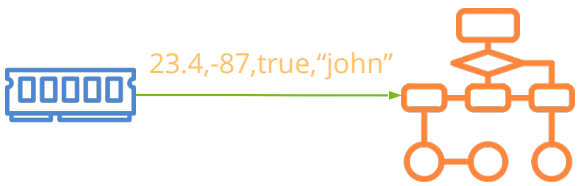


Figure 15.9: With *block streams* we will read formatted data from memory areas.

15.5.5. Memory stream

Memory streams are read/write channels that allow implementing the producer/consumer model (Figure 15.10). First, the information reaches the stream through write operations and is stored in an internal memory buffer. Subsequently, said information can be read by another function, thread or process. After each reading the information read will disappear from the channel. The concept is similar to that of IPC-pipes, except that

there is no size limit for the buffer, but it will grow on demand. Read and write operations can be done simultaneously depending on the established protocol.

- Use `stm_memory` to create a stream in memory.
- Use `stm_buffer` to access the internal buffer.

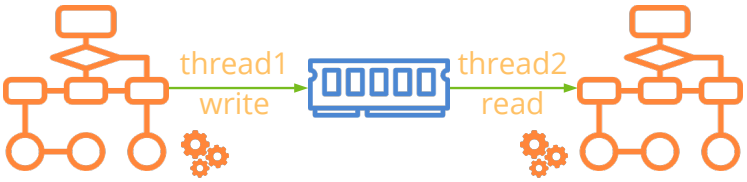


Figure 15.10: Producer/consumer model implemented with *memory streams*.

Although this type of stream supports read and write operations it is not considered full-duplex. The reading is done on previously written data, but cannot "answer" the interlocutor. It is not a "conversation".

15.5.6. Standard stream

The *"Standard I/O"* (page 161) can be managed by *streams* using three predefined objects (Figure 15.11). These objects are created when the program starts and will be automatically released when finished.

- `kSTDIN`: To read from the standard imput.
- `kSTDOUT`: To write in standard output.
- `kSTDERR`: To write in the error output.

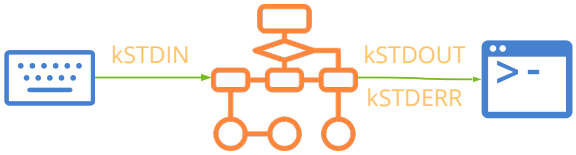


Figure 15.11: Access to standard I/O through streams.

```
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

Sometimes it can be useful to have a "sink" that ignores all write operations (Figure 15.12). Think of debugging tasks where we want to activate or deactivate the output of information but deleting or commenting on the code is cumbersome. The idea is similar to the Unix `/dev/null`.

- Use `kDEVNULL` to write to a sink that will ignore all received data.

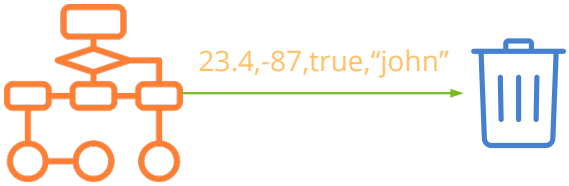


Figure 15.12: With *null streams* everything that is written will be ignored.

```
#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);
```

Cannot read from ***kDEVNULL****.*

15.5.8. Binary stream

Generic binary data always travels through a stream as bytes. How these data are interpreted depends on the interlocutors and their communication protocol. But by emphasizing "binary data" we mean that numeric values are written to the channel as they appear in the CPU registers using binary, two's complement, or IEEE754 (Figure 15.13) code. In multi-byte types we must take into account the *"Byte orderByte order"* (page 206). In `stream.h` several functions are defined to read and write binary types.

- Use `stm_read_u32` to read a 32-bit unsigned integer.
- Use `stm_write_r64` to write a real 64bits (`double`).
- Use `stm_write_bool` to write a boolean.

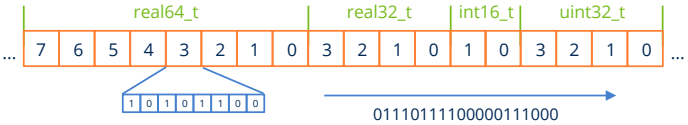


Figure 15.13: Numbers in binary format.

15.5.9. Text stream

Text streams are a particular case where the binary information is assumed to represent Unicode character codes *(codepoints)* (Figure 15.14) (Listing 15.4). This means that the content of the stream will be readable directly by a human, but it will require a post-processing (*parsing*) in destination to interpret these texts and translate them into binary. You do not have to do anything special when creating a stream to indicate that it is of type text, you just have to use the appropriate functions.

- Use `stm_printf` to write text in a stream.
- Use `stm_read_char` to read a character from a stream.
- Use `stm_read_line` to read a text line from a stream.
- Use `stm_col` to get the column number of the last character read.
- Use `stm_row` to get the row number of the last character read.

Figure 15.14: In text streams the information can be read directly.

Listing 15.4: Reading a text file using 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` and other reading functions will always return the text in `UTF8`. But if the data inside the stream were in another format, we must use `stm_set_read_utf` , in order to carry out the conversion correctly. See *"UTF encodingsUTF encodings"* (page 157).

On the other hand, `stm_printf` also receives the text in UTF8, but the receiver may need it in another format. We will use `stm_set_write_utf` to set the output encoding. We will write in UTF8, but the channel will be sent in UTF16 or UTF32.

> *Streams do not have to be "pure" text or binary. They can combine both types of representations.*

15.5.10. Tokens

When reading from text streams, an interpretation (*parsing*) of the content is necessary in order to transfer the data to memory variables (in binary). The first step is to break the text into symbols (or words) called *tokens*. Internally, the streams incorporate a simple **lexical analyzer** that recognizes the tokens of the C language, very common in countless grammars and file formats (Figure 15.15). It is implemented as a finite state machine and will greatly facilitate the processing of these text flows. In (Listing 15.5) we see the code necessary to read one by one all the tokens from a `.c` file. We have the result of processing the file (Listing 15.6) in (Listing 15.7).

- Use `stm_read_token` to read a token.
- Use `stm_token_lexeme` to obtain the string associated with the last token read.
- Use `stm_read_r64_tok` to read a `real64_t` from text.
- Use `stm_token_col` to get the column of the last token.
- Use `stm_token_row` to get the row of the last token.

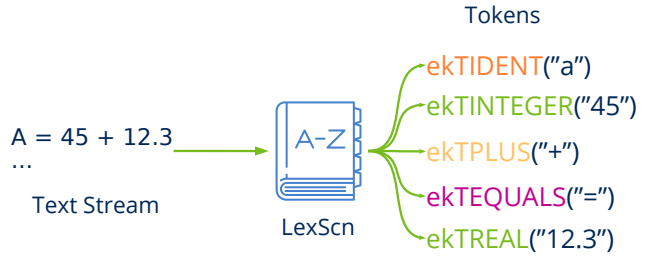


Figure 15.15: Decomposition of a text into *tokens*.

Listing 15.5: Reading *tokens* from a file in 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
        ...
    }
}
```

Listing 15.6: File *source.c*.

```
void func(int a)
{
    int i;
    char *str = "Hello";

    i = 5 + 2.5;
}
```

Listing 15.7: Lexical analysis of *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. Identifiers

An identifier is an alphanumeric "word" that must begin with a letter or `'_'` and contains numbers, letters, or `'_'`. It is used to name variables, functions, reserved words, etc. They do not allow spaces or symbols. (Listing 15.8) (Figure 15.16).

Listing 15.8: Correct and incorrect identifiers.

```
OK: while cos _reSult a56B _06_t aG h9 _12AcVb
NO: 045 ?er "_5G _tg(
```

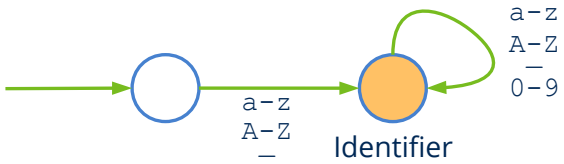


Figure 15.16: Finite automata that recognizes an identifier.

Certain identifiers can be reserved to act as language **keywords**. For example `for`, `while` or `if` are C keywords and cannot be used for the naming of variables or functions. Being general purpose, our scanner does not recognize any type of reserved word, but must be expressly tagged after reading the token (Listing 15.9).

Listing 15.9: Recognizing the **while** keyword.

```
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. Strings

A text string is a series of Unicode characters enclosed in quotation marks (`""`) (Figure 15.17). The parser recognizes C escape sequences to represent non-printable codes or unavailable characters on the keyboard (Listing 15.10).

- Use `stm_token_escapes` to make escape sequences effective when reading strings.

Listing 15.10: Escape sequences accepted in `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
```

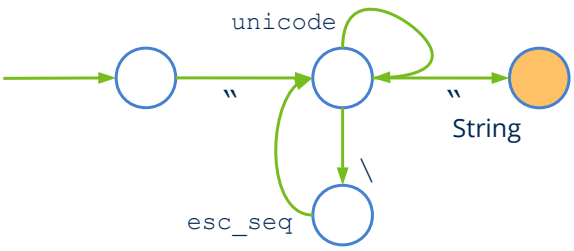


Figure 15.17: Finite automata that recognizes a text string.

15.5.13. Numbers

In the case of numerical *tokens* the thing is complicated a bit due to the different numerical bases and the exponential representation of real numbers (Figure 15.18). We briefly summarize it, although it is common to many programming languages (C included).

- If the number starts with `0` it will be considered octal (base 8), therefore, the following digits are limited to `0-7`, eg: `043, 001, 0777`.
- If the number starts with `0x` will be considered hexadecimal (base 16) with digits `0-9 a-f A-F`, eg: `0x4F, 0XAA5, 0x01EAC`.
- At the moment a decimal point appears `'.'` will be considered real number. A point at starting is valid, eg: `.56`.
- An integer or real number allows exponential notation with the character `'e'('E')`, eg: `12.4e2, .56e3, 1e4`.

15.5.14. Symbols

The symbols are single-character *tokens* that represent almost all US-ASCII punctuation marks and are often used as operators, separators or limiters within grammars. (Listing 15.11) (Figure 15.19).

Listing 15.11: Symbols recognized as *tokens* by `LexScn`.

```
< > , . ; : ( ) [ ] { } + - * = $ % # & ' " ^ ! ? | / \ @
```

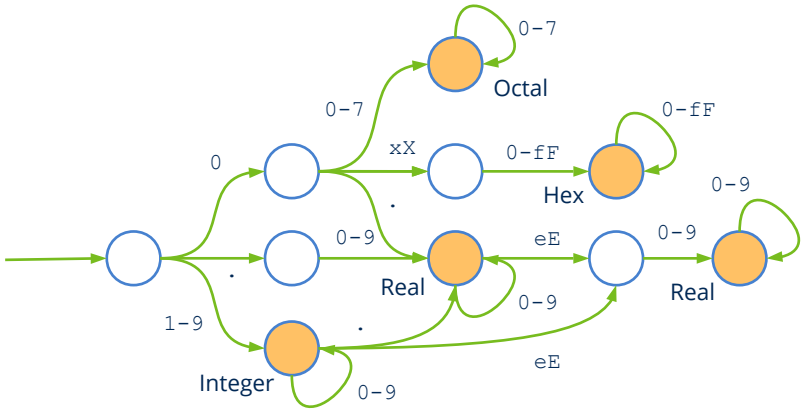


Figure 15.18: Finite automata that recognizes numbers.

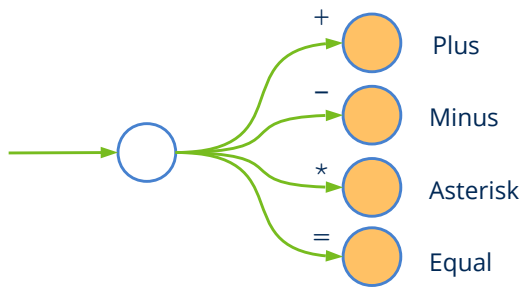


Figure 15.19: Finite automata that recognizes some symbols.

15.5.15. Comentarios

By default, C `/*Comment */` or C++ `//Comment` are ignored by `stm_read_token`.

- Use `stm_token_comments` so that it returns `ekTSLCOM` or `ekTMLCOM` if it finds any.
- Use `stm_token_spaces` to return `ekTSPACE` when it finds blank spaces.

15.5.16. Stream advantages

Although it is possible to read or write directly to the I/O channels (*"Memory"* (page 162), *"Files and directories"* (page 177), *"Sockets"* (page 179), *"Standard I/O"* (page 161)), do it through `Stream` objects has certain advantages. Therefore, we recommend using them instead of low-level APIs for the following reasons:

15.5.17. Unify serialization

Streams offer a uniform interface, regardless of the origin and destination of the data (Figure 15.20). For the object serialization, we just have to write a reader and a writer, without worrying if the object will be saved to disk, transmitted over the Internet or stored

temporarily in memory (Listing 15.12).

Listing 15.12: (De)serialization of an object through 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);
}
```

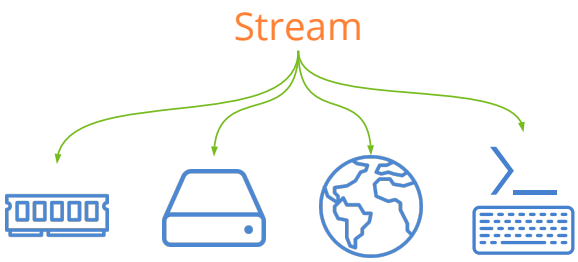


Figure 15.20: Through streams we manage all I/O channels with the same interface.

15.5.18. More elegance

The I/O channels only work with byte blocks. Streams implement high-level functions for texts and binary types. This will make our code much more readable. (Listing 15.13).

Listing 15.13: Writing an object to disk directly or through a 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. Higher productivity

Related to the previous one, streams can "parse" text strings directly. You can get characters, words or lines without having to scan the entry character by character (Listing 15.14).

Listing 15.14: Read a line of text directly or through a 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. Higher performance

File streams and *socket streams* implement an internal cache. This allows less access to the channel with a higher volume of data, which means faster processing speed. (Figure 15.21).

- Use `stm_flush` to clear the cache and dump the data in the channel.

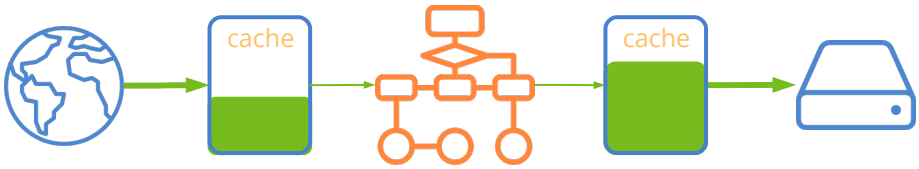


Figure 15.21: Streams implement cache memory, which increases performance.

15.5.21. Byte order

When reading or writing binary data from an I/O channel, special attention must be paid to the order of the bytes in 16, 32 or 64 bit data types, which is known as *endianness*. On *litte endian* machines, as is the case with the Intel x86/x64 family processors, the lowest order byte will be located at the lowest memory address. In the case of the *big endian* (Motorola 68000, PowerPC) it happens on the contrary, it will go in the highest. For example, if we write a 32-bit integer in a file or *socket* from a *little endian* machine and read it from a *big endian,* the data will be corrupted by altering the internal order of bits (Figure 15.22). The `Stream` objects automatically adjust the *endianness* in each read/write operation. Default is set `ekLITEND`, except in *sockets* that will be `ekBIGEND` for being the accepted agreement for network communications. However, it can be changed if necessary.

- Use `stm_set_write_endian` to establish the *endianness* of the output channel. The data will pass from *endian CPU* to *Stream endian* before being written.

- Use `stm_set_read_endian` to establish the *endianness* of the input channel. The data will pass from *Stream endian* to *CPU endian* at the time of being read.

Figure 15.22: We must take into account *endianness* when sharing data between machines of different architecture.

Endianness does not influence "UTF-8UTF-8" (page 158) text strings, but it does in the "UTF-16UTF-16" (page 157) and "UTF-32UTF-32" (page 157).

15.5.22. Stream state

A stream can be affected by two types of problems. On the one hand the **data corruption** that is evident when we read binary data from the stream. A clear example would be to read a Boolean by `stm_read_bool` and get a value of `129` when obviously this value should be `0` (`TRUE`) or `1` (`FALSE`). Typically, a stream becomes corrupted due to lack of coordination between writer and reader and is usually due to a programming error. This situation should be resolved by debugging and correcting the serialization of objects or reviewing the data protocol. On the other hand, there may be "physical" errors in the channel (file deleted, loss of Internet connection, permissions, etc.). In both cases, the stream will be blocked and subsequent read or write operations that we carry out on it will be ignored. We can also ask the total number of bytes read and/or written in the channel, in case we need to know if there is information available for reading.

- Use `stm_state` to know the current status of the channel.
- Use `stm_file_err` to get extended error information on disk streams.
- Use `stm_sock_err` to get extended error information in *sockets*.
- Use `stm_corrupt` to mark a stream as `ekSTCORRUPT`. Sometimes it is the application itself that detects that the data is not correct (eg out of range).
- Use `stm_bytes_written` to get the total number of bytes written to the stream.
- Use `stm_bytes_readed` to get the total number of bytes read from the 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

Being able to work with data collections is essential when designing our model. In addition to the basic types and the `struct`, `union` or `class`, the C language offers us the *array* construction, which allows to store several elements under the same variable name (Listing 15.15):

Listing 15.15: C Arrays.

```
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];
```

Or, dynamically (Listing 15.16):

Listing 15.16: Dynamic arrays.

```
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);
```

The **C arrays** store elements in contiguous positions of memory and, although they are very quick to consult, they lack the functionality to insert, delete, search or sort. In many cases, the data is not available when the container is created, but they are entering or leaving dynamically during the program execution, so we cannot anticipate in advance a maximum number with which to make the memory reservation. The `Array` type implemented in NAppGUI is, in essence, a dynamic C array and a series of methods to manipulate it. By dynamic we understand that the structure adjusts its size to the actual amount of elements, keeping the main premise that **all remain in memory together**.

When an `Array` is created, memory is reserved for a few records (Figure 15.23). Later, we can add new elements at the end (typical) or insert them in any random position in case we already have data in the container. In the latter case, the rest of the elements will be shifted to the right. As soon as the number of reserved records is exceeded, the internal dynamic block will be doubled to accommodate the new positions. In the same way it is possible to eliminate any element of the collection, moving the rest to the left to maintain the spatial coherence of the structure. If the number of items decreases by half, the memory block will be reduced. In this way, during the life of the container, the memory will be adjusted by multiplying or dividing by 2 the number of reserved elements.

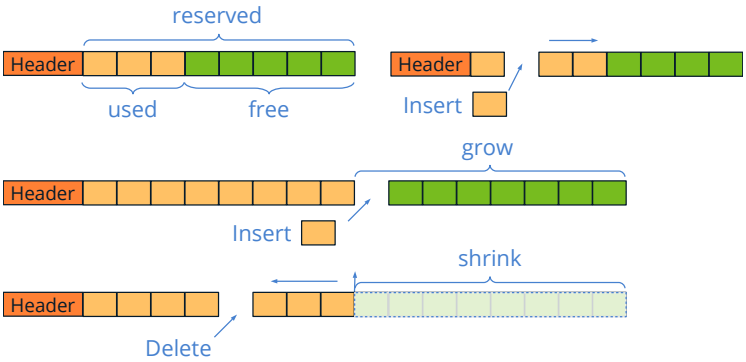


Figure 15.23: The `Array` adapt their internal memory to the number of elements.

15.6.1. Registers or pointers

An object of type `Product`, our example structure, needs 20 bytes on 32-bit systems (Figure 15.24). The `code`, `description` and `image64` fields are pointers that point to other memory areas, where the `String` and `Image` type fields reside, dynamically reserved.

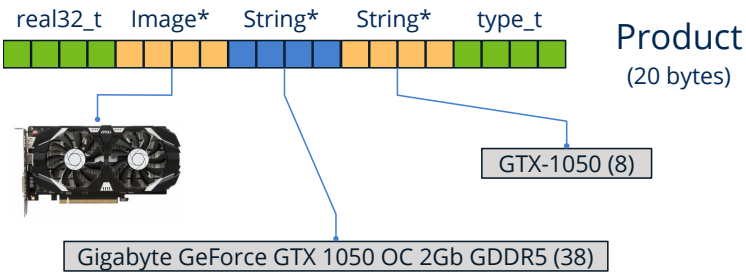


Figure 15.24: Product object.

Depending on what is stored inside the container, we can use two kinds of array (Listing 15.17). The array of records will keep the entire object (20 bytes) inside and the array of pointers only a reference to it (4 bytes), the actual object being located in another

memory address (Figure 15.25). Although the internal structure management is the same, access to the elements differs slightly.

- Use `arrst_create` to create an array of records.
- Use `arrpt_create` to create an array of pointers.

Listing 15.17: Create an array.

```
ArrSt(Product) *arrst = arrst_create(Product);
ArrPt(Product) *arrpt = arrpt_create(Product);
```

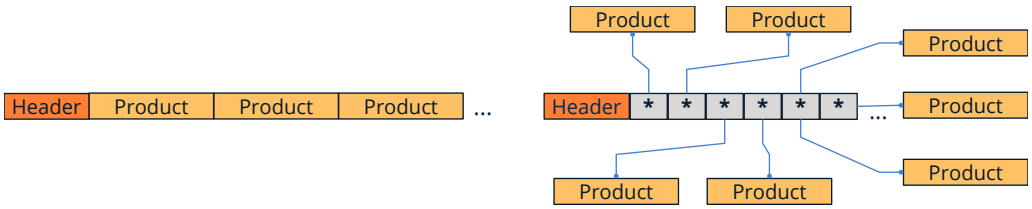


Figure 15.25: Arrays of registers and pointers.

Use `ArrSt` can slightly improve performance, thanks to spatial consistency, which reduces cache failures, and saving calls to the memory manager *"Arrays vs Sets comparativeArrays vs Sets comparative"* (page 221). But this will not always be possible, and we cannot use them in these cases:

- Opaque objects: If the type definition is not public, the container cannot calculate the space required for each element, so we can only work with pointers to them.
- Shared objects: If other structures of the model keep pointers to the elements of the container, we will have *Segmentation Fault* problems due to the change of memory addresses when relocating the internal container block (Figure 15.26).

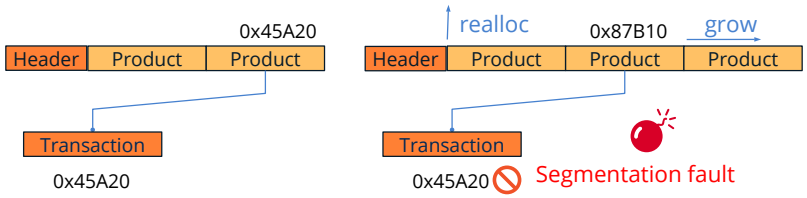


Figure 15.26: Register arrays are dangerous with external references.

15.6.2. Type check

You will have noticed in (Listing 15.15) that two statements appear just after the definition of the `struct Product`: `DeclSt` and `DeclPt`. These are two macros that enable compile-time type checking, defining a custom interface in the containers for this new type

(Listing 15.6.2). All things considered, they mimic the C++ `template<>`. `DeclSt` enables record containers and `DeclPt` pointer ones.

```
Product *p1 = arrst_new(Product);
Product *p2 = arrst_get(arrst, 5, Product);
const Product *p3 = arrst_get_const(arrst, 5, Product);
```

Although it is not advisable, you can dispense with the use of these macros and use the "raw" interfaces of the containers, defined in `array.h` and `rbtree.h`. In this case your code will be much less readable and you will not have compiler support.

Headers `array.h` and `rbtree.h` are not documented.

15.6.3. Constructors

When memory is reserved for an object, either in the *"Stack SegmentStack Segment"* (page 162) as **automatic variables**

```
Product product;
```

at *"Heap SegmentHeap Segment"* (page 163) through dynamic memory

```
Product *product = heap_new(Product);
```

or in a container

```
Product *product = arrst_new(array, Product);
```

its initial content is garbage, understood as undetermined bytes. Initializing an object is assigning valid and consistent values to each field of the object (Listing 15.18).

Listing 15.18: Initializing an object `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;
}
```

For its part, a constructor is an initializer that previously reserves memory dynamically to store the object (Listing 15.19).

Listing 15.19: Constructor of object `Product`.

```
static Product *i_create(void)
```

```
{
    Product *product = heap_new(Product);
    i_init(product);
    return product;
}
```

When we use register arrays, we will only need to initialize the object, since the space to store it has been reserved by the container itself (Listing 15.20). However, in pointer arrays, the memory for the object must be explicitly reserved, since the container will only save a reference.

Listing 15.20: Insert correctly initialized objects.

```
// 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);
```

> *Use `arrst_new`, `arrst_insert_n` or `arrst_prepend_n` whenever possible to insert into record arrays, as they avoid having to copy the object.*

15.6.4. Array loops

To iterate over all the elements of the array, we can choose between two types of syntax to implement the loop.

```
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. Copy objects

Similar to constructors, there are two methods for copying objects (Listing 15.21). In the first one, we generate dynamic memory for the object's fields, but not for the object itself, either because it is an automatic variable or is stored in an array of records. In the second case, we reserve dynamic memory for both the object and its elements.

Listing 15.21: Copying an object `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. Serialization

A special case of the constructor are the **readers** (de-serializers). When we create an array from the content of *"Streams"* (page 193) (Listing 15.22), we need a method capable of creating or initializing an element from the stream itself. Depending on the type of container it will be necessary to reserve memory for each item or not.

Listing 15.22: Reading an array from a 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);
```

In the same way we can write (serialize) the contents of an array in a write stream (Listing 15.23). In this case, a single write function is sufficient for both types of containers, since each one knows how to access its elements.

Listing 15.23: Writing an array in a 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. Destructors

In programming many times we are confused by the verbs: *'delete', 'destroy', 'free', 'erase', 'remove', 'clear'* since they essentially mean the same thing but with subtle differences. In NAppGUI we will use one verb or another depending on concrete actions:

- **Free:** Only free dynamic memory allocated to an object (Listing 15.24). You need a double pointer, since the object will be invalidated (=NULL) after freeing it, avoiding references to free memory areas.

  Listing 15.24: Freeing the memory of an object.

  ```
  Product *product = heap_new(Product);
  ...
  heap_free(&product, Product);
  // product = NULL
  ```

- **Remove:** It destroys the fields of an object, but does not free the memory of the object itself. It is the opposite of the *initializer* (Listing 15.25).

Listing 15.25: Freeing memory from object fields.

```
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:** The combination of the previous two. Destroy the fields of the object and free its memory (Listing 15.26). It is the opposite of the constructor. Obviously, it requires a double pointer to invalidate the reference.

Listing 15.26: Free the object's memory and all its contents.

```
static void i_destroy(Product **product)
{
    i_remove(*product);
    heap_free(product, Product);
}

arrpt_destroy(&arrpt, i_destroy, Product);
```

- **Delete:** Delete an element from an array or other type of container (Listing 15.27). It may have associated a destructor or remover, although it is not mandatory.

Listing 15.27: Delete an item from a container.

```
// 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:** Delete all the elements of a container, but do not destroy it, just leave it to zero (Listing 15.28). Like `arrst_delete`, optionally can free objects memory.

Listing 15.28: Clear a container, deleting all its items.

```
// 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. Sort and search

The usual way to use arrays will be to add elements at the end by `arrst_new` or `arrpt_append` then iterate over all. This "natural" order will be enough in most cases, but we may need to organize the elements following another criterion for:

- Present the information ordered by one or several fields of the structure.
- Optimize searches. To locate a certain element, there is no choice but to travel the entire array, with linear cost `O(n)`. But we can solve the search in logarithmic time `O(logn)` if the array is sorted, dramatically increasing performance especially in large sets (Figure 15.27).

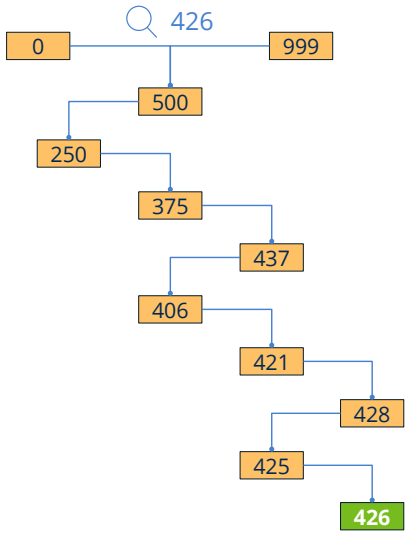


Figure 15.27: In a maximum of 10 steps we will find an element among a thousand (20 steps for a million).

- Use the function `arrst_sort`, to sort an array. We will have to pass a comparison function, which will determine the order relationship (Listing 15.29).

Listing 15.29: Sort arrays by product code.

```
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);
```

To search for an element within an array, we must also provide a function that compares the object with a key. This key contains the search criteria and is usually smaller than the element itself. Many times it is just a simple number or a text string (Listing 15.30).

- `arrst_search` Slow method. It will search for elements in a linear way, one by one `O(n)`.
- `arrst_bsearch` Fast method. It will search elements in logarithmic way, `O(logn)`. The array must be sorted according to the same criteria as the search.

Listing 15.30: Search for an item by its code.

```
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 of basic types

The basic types are a particular case of single-field structure, so we will use it `ArrSt`. In the specific case of `enum` we must create an alias by `typedef`, as `ArrSt(type)` does not support the keyword `enum`, just as does not support `struct` keyword. In C++ this alias is not necessary. When destroying the array we will pass `NULL` to the destructor parameter, since the basic types do not generate dynamic memory.

```
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 (pointers)

15.8. Binary search trees

Like *arrays* **binary search trees (BST)**, also known as sets or maps, are containers that allow us to work with a collection of objects. The main difference with the first ones

is that the elements are not stored linearly in contiguous positions of memory, but use a tree-shaped structure where each node has two descendants (Listing 15.31) (Figure 15.28).

Listing 15.31: Creation of arrays and 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);
```

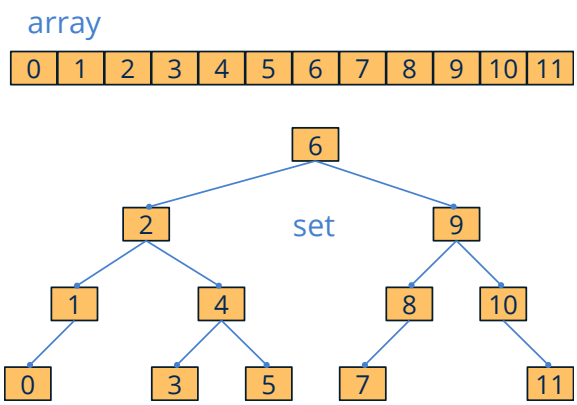


Figure 15.28: Array and set representation.

BSTs are structures optimized for cases where insertions, deletions and searches are very frequent. They are permanently sorted, hence it is possible to insert, delete or locate any element in logarithmic time `O(logn)`, no need to use sort functions like `arrst_sort` (Figure 15.29). For maintenance to be carried out efficiently, the tree that supports the structure must meet a number of characteristics:

- **Binary** : Each node can only have 0, 1 or 2 children.
- **Sorted** : All descendants to the left of a node are of lesser value and those to the right of greater value. The order and search criteria are set in the constructor by a

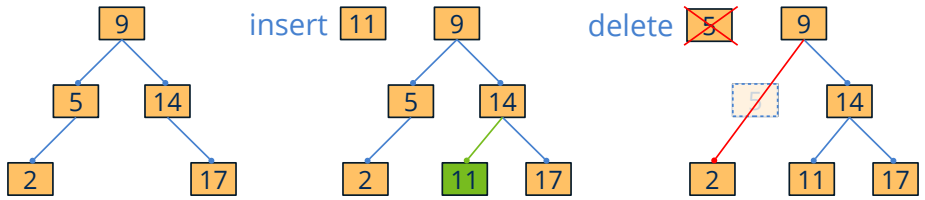


Figure 15.29: In search trees the insertion or deletion does not break the order of the set.

comparison function (`i_compare` in the previous example) and cannot be changed during the lifetime of the container. The new elements will be inserted in their correct position according to this order. It does not support duplicate elements or in arbitrary positions.

- **Balanced**: A tree can fulfill the two previous properties, but have degenerated to a list where searches can no longer be resolved in logarithmic time (Figure 15.30). Internally, the NAppGUI `Set` containers are implemented with the so called *red-black trees*, where a maximum height of `2log(n+1)` is guaranteed. This is achieved by restructuring the tree after each insertion or deletion, so adding a new element (or removing it) is resolved in a maximum of `O(logn)`. This is much faster than in arrays, where have to move all the elements to insert a record in a specific position, with an associated cost of `O(n)`.

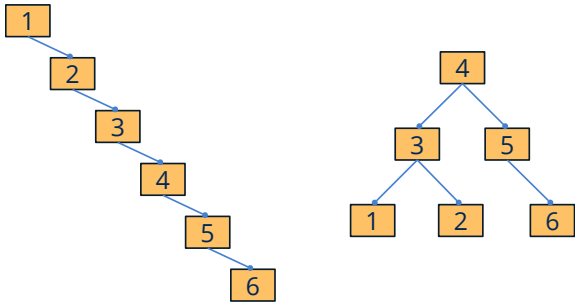


Figure 15.30: Degenerated and balanced search tree.

As we saw in *"Registers or pointersRegisters or pointers"* (page 209), we have two modalities when creating sets (Figure 15.31). The register-based version is more efficient than the pointer-based version, although less flexible.

- Use `setst_create` to create a set of registers.
- Use `setpt_create` to create a set of pointers.

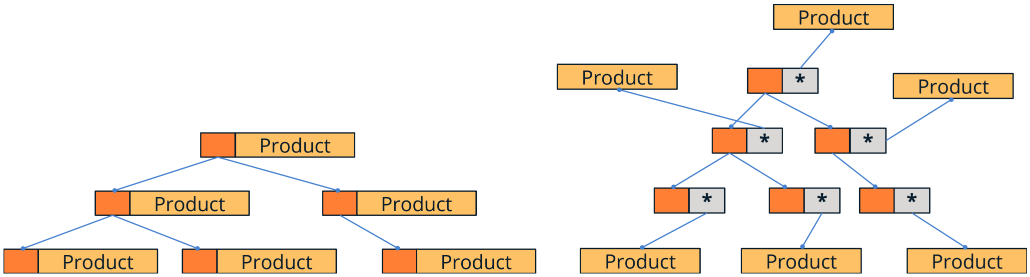


Figure 15.31: Sets of registers and pointers.

15.8.1. Iterators

We cannot access the elements of a set using a random index, as was the case with arrays. The nodes are dispersed in different memory areas, which prevents calculating the position of a particular element from a base address. An iterator is nothing more than a pointer within the set that acts as a marker for the currently selected element (Figure 15.32). From a specific position, we can move to the previous or subsequent element, but never make arbitrary jumps. We can control the position of the iterator with different functions (Listing 15.32):

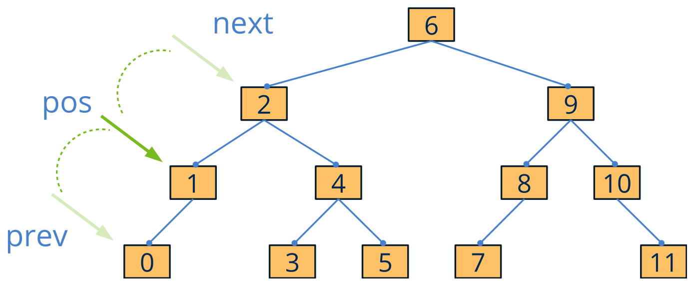


Figure 15.32: The iterators allow us to move through the structure.

- Use `setst_get` to search for an item. The iterator will be fixed on it.
- Use `setst_next` to move the iterator to the next item.
- Use `setst_prev` to move the iterator to the previous item.
- Use `setst_first` to move the iterator to the first element of the set.
- Use `setst_last` to move the iterator to the last element of the set.

Listing 15.32: Iterating over the elements of a *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. Arrays vs Sets comparative

We have performed a test to see the behavior of these two types of structures in real situations, apart from mere theory (Table 15.1). The structure used has been `Product` described in (Listing 15.31). We will compare six types of containers `ArrSt(Product)` and `ArrPt(Product)` (unsorted), `ArrSt(Product)` and `ArrPt(Product)` (sorted), `SetSt(Product)` and `SetPt(Product)`.

- The items will be sorted by `code` field using the method `i_compare` described in (Listing 15.31).
- The elements have been created previously and reside in memory. Times only reflect the management performed by the containers.
- Field `code` take values from `"0"` until `"n-1"`, where `n=100,000` is the number of elements. The elements have been previously messed up using the function `bmem_shuffle_n`.
- The tests have been performed on a **Raspberry Pi 3 Model B** with NAppGUI compiled in Release version (*"ConfigurationsConfigurations"* (page 95)). We have chosen this platform because of its clear technical inferiority with respect to others. In this way the asymptotic difference is more evident.

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

Table 15.1: Results of the comparison (in seconds).

In view of these data, we can reach the following conclusions:

- Linear searches `O(n)` are tremendously slow.
- Keeping an array sorted after each insertion or deletion is expensive. It is more efficient to add all the elements and then order, although this will not always be possible. If the elements enter or leave arbitrarily but the set must always be ordered, it is better to use Sets.
- Register-based containers are more efficient in queries, but less when inserting or deleting. However, this test does not include the time to create or release dynamic memory, something inherent in pointer containers.
- Iterating in arrays is almost free, but iterating in sets has a small cost due to the logic of jumping between nodes.
- We cannot say that one container is better than another in general. It will depend on each specific case.
- For small groups (less than 1000 elements) the differences are practically imperceptible.
- For extremely small groups (up to 100 items) always use arrays. The asymptotic Sets improvement is marred by the much more efficient implementation of the Arrays.

15.9. Binary search trees (pointers)

15.10. Regular expressions

Regular expressions define a text pattern that can be used to find or compare strings.

- Use `regex_create` to create a regular expression.
- Use `regex_match` to check if a string matches the pattern.

Listing 15.33: Using regular expressions.

```
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);
```

Result of (Listing 15.33).

```
YES: file01.txt
NO:  image01.png
NO:  sun01.jpg
YES: films.txt
NO:  document.pdf
```

15.10.1. Define patterns

We can build a regular expression from a text string, following these simple rules:

- A string pattern corresponds only to that same string.

```
"hello" --> {"hello"}
```

- A period `'.'` is equivalent to "any character".

```
"h.llo" --> {"hello", "htllo", "hällo", "h5llo", ...}
```

- A dash `'A-Z'` sets a range of characters, using the ASCII/Unicode code from both ends.

```
"A-Zello" --> {"Aello", "Bello", "Cello", ..., "Zello"}

'A-Z': (65-90) (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
'0-9': (48-57) (0123456789)
'á-ú': (225-250) (áâãäåæçèéêëìíîïðñòóôõö÷øùú)
```

Like `String` objects, patterns are expressed in "UTF-8UTF-8" (page 158), therefore the entire Unicode set can be used to create regular expressions.

- The brackets `'[áéíóú]'` allow you to switch between several characters.

```
"h[áéíóú]llo" --> {"hállo", "héllo", "híllo", "hóllo", "húllo"}
```

- The asterisk `'*'` allows the last character to appear zero or more times.

```
"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",...}
```

- The parentheses `'(he*llo)'` allow grouping a regular expression, so that it behaves as a single character.

```
"[(hello)(bye)]" --> {"hello", "bye" }
"[(red)(blue)(1*)]" --> {"red", "blue", "", "1", "11", "111", ... }
"(hello)*" --> {"", "hello", "hellohello", "hellohellohello", ... }
"(he*llo)ZZ" --> {"hlloZZ", "helloZZ", "heelloZZ", "heeelloZZ", ... }
```

- For `'.', '-', '[]', '*', '()'` to be interpreted as characters, use the *backslash* `'\'`.

```
"\(he\*\-llo\)" --> {"(he*-llo)"}
```

Remember that for expressions inserted as constants in C code, the backslash character is represented by a double slash `"\\(he\\(*\\(-llo\\()"`.

15.10.2. Regular languages and automata

Regular languages are those that are defined recursively using three basic operations on the set of characters (or symbols) available. They can be described using the regular expressions discussed above.

- Each character **'a'** is a regular language **'A'**.
- The union of two regular languages, is a regular language **A B**.
- The concatenation of two regular languages, is a regular language **A · B**.
- The closure of a regular language is a regular language **A***. This is where recursion comes in.

In this context the symbols are all Unicode characters. But you can define languages based on other alphabets, including the binary {0, 1}.

To recognize whether or not a string belongs to a certain regular language, it is necessary to build a **Finite Automata** based on the rules reflected in (Figure 15.33).

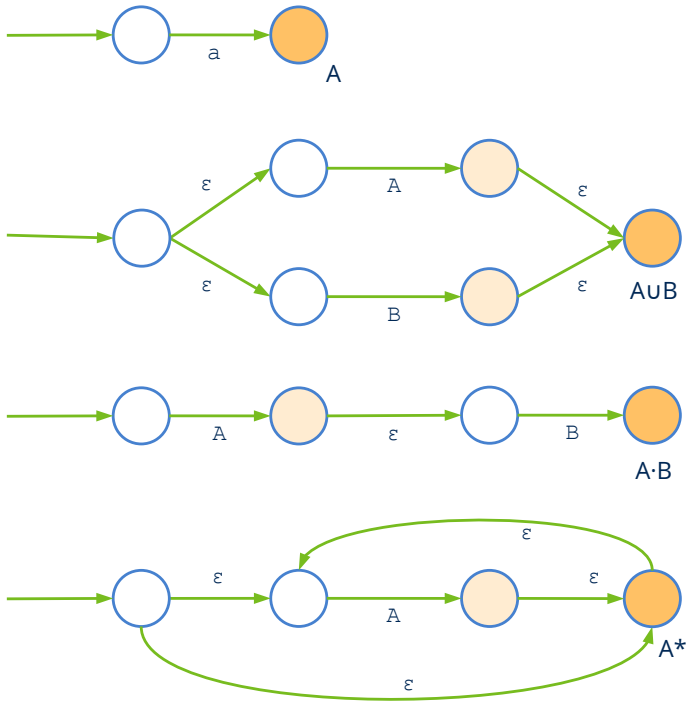


Figure 15.33: Construction of finite automata to filter regular expressions.

15.11. Data binding

We understand by *Data Binding* the possibility of automatically synchronizing the structures of the program with different input/output sources. We start from the simple model (Listing 15.34) that we present in *"Arrays"* (page 208) composed of a `struct` and an `enum`.

Listing 15.34: Simple data model based on `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;
};
```

The first thing we have to do is register this model in **dbind**, a kind of general "database" within our application (Listing 15.35). It is only necessary to carry out this process once when starting the program. In this way, internal tables will be created with the description of each structure of the data model (Figure 15.34), leaving the program ready to automate certain tasks when working with objects of said classes.

- Use `dbind` to register the fields of a structure.
- Use `dbind_enum` to register the different values of enum types.

Listing 15.35: Registering the data model of (Listing 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);
```

```
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

Figure 15.34: Internal tables created by *dbind* when registering the data model.

15.11.1. Synchronization with graphical interfaces

One of the most widespread uses of data binding is the possibility of synchronizing the graphical interface with the objects that make up the data model. This paradigm is known as MVVM *(Model-View-ViewModel)* (Figure 15.35)and we will delve deeper into it *"GUI Data binding"* (page 352).

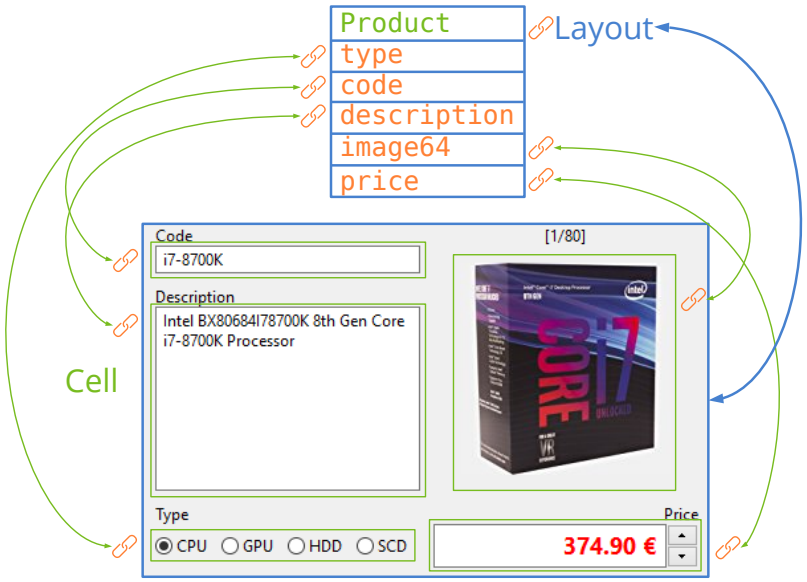


Figure 15.35: Automatic synchronization of data and interface.

15.11.2. Read and write JSON

The parsing of JSON scripts can also be automated thanks to *dbind* (Figure 15.36). In *"JSON"* (page 379) you will have detailed information on how to do it.

```
{
    "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....
    },
    ...
}
```

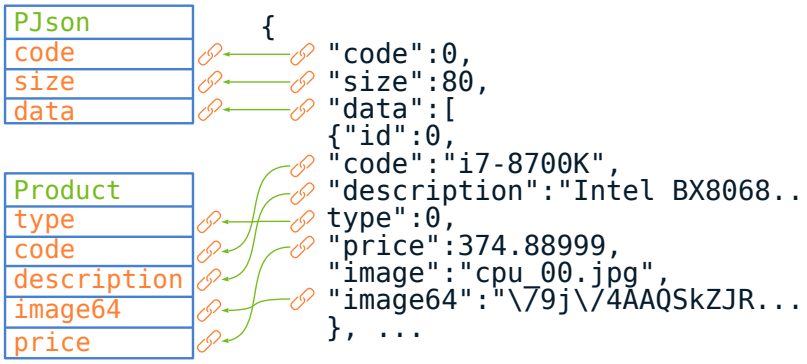


Figure 15.36: Data Binding in the analysis of JSON scripts.

15.11.3. Serialization with DBind

As we saw in *"SerializationSerialization"* (page 213) and *"Unify serializationUnify serialization"* (page 203) we need to define object reading and writing functions to send or receive them through streams. Fortunately, *dbind* knows the detailed composition of each registered object, so it's possible access the I/O without having to explicitly program these functions (Listing 15.36) (Figure 15.37).

Listing 15.36: Objects serialization with *dbind*.

```
ArrPt(Product) *products = dbind_read(stream, ArrPt(Product));
...
dbind_write(stream, products, ArrPt(Product));
```

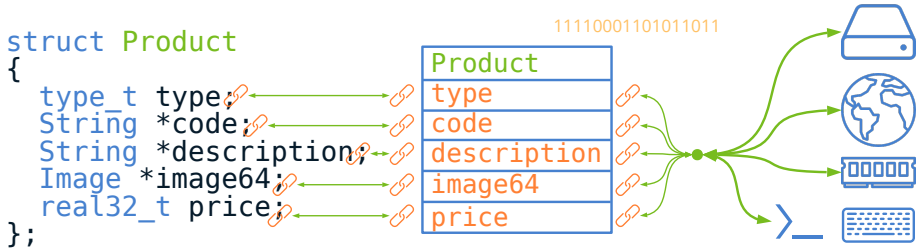


Figure 15.37: Object reading/writing via *dbind*.

15.11.4. Default constructor

Thanks to *dbind* we can also create objects initialized with default values without creating specific constructors (Listing 15.37). They can also be destroyed guaranteeing the correct recursive release of the memory of all their fields.

- Use `dbind_create` to create an object *"ConstructorsConstructors"* (page 211).

- Use dbind_init to initialize an object.
- Use dbind_destroy to destroy an object *"DestructorsDestructors"* (page 214).

Listing 15.37: Construction and destruction without additional methods.

```
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));
```

The default values when initializing object fields are 0 for numbers, FALSE for booleans, "" for Strings and empty containers in the case of arrays or sets. If the object contains nested sub-objects, they will also be created/initialized recursively. These default values can be changed if necessary (Listing 15.38).

- Use dbind_default to set the default value.

Listing 15.38: Changing the default values.

```
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. Numerical ranges

It is possible to configure the numeric fields uint32_t, int8_t, real64_t, etc to limit the accepted values (Listing 15.39). dbind will be responsible for validating the data every time it reads values from any data source (GUI, JSON, Streams, etc).

- Use dbind_range to set a maximum and minimum numerical values.
- Use dbind_precision to set numerical precision. For example 0.01 in monetary values.
- Use dbind_increment to set the value of discrete increments.
- Use dbind_suffix to set a suffix that will be added when converting numbers to text.

Listing 15.39: Value range and accuracy of price value.

```
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. Events

An event is an action that occurs during the program execution, usually asynchronously or unpredictably and on which a given object must be notified. In applications with a graphical interface, many events are constantly occurring when the user interacts with the different controls. However, they can also occur in console applications, for example, when finish the writing of a file to disk or when downloading a page from Internet. In a system of events two actors intervene: The sender, which has evidence when the action occurs and the receiver who is notified that such action has occurred. To connect both ends we must perform these simple steps (Listing 15.40) (Figure 15.38):

- Create a `listener` indicating the receiving object and the *callback* function to which the sender should call.
- Said *listener* is assigned to the sender by the appropriate method. For example, the `Button` type provide the method `button_OnClick` to notify of a click.
- When the event occurs, the sender calls the *callback* function, indicating the receiving object (parameter of `listener`) and detailed information about the event collected in the object `Event`.

Listing 15.40: *Callback* function and button click event.

```
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));
}
```

Events are used in bulk in GUI applications, but can also be useful in command line applications. See for example `hfile_dir_loop` in "File operations" (page 231).

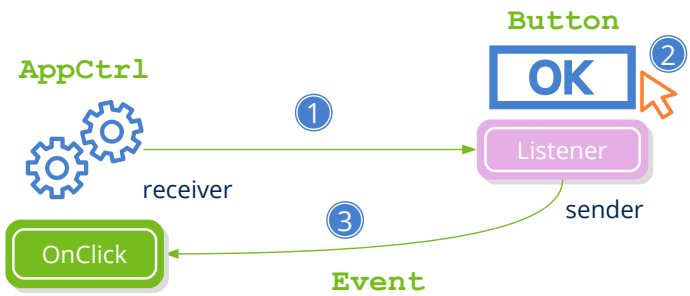


Figure 15.38: `OnClick` event.

15.13. Keyboard buffer

The operating system generates events related to the pressing or releasing keys `view_OnDown` `view_OnUp`. Sometimes we need to know the state of a key without having to be aware of the events they generate. `KeyBuf` offers a very simple query mechanism using only the value of the `vkey_t` key.

- Use `keybuf_create` to create the buffer.
- Use `view_keybuf` to assign the buffer to any generic view, which will be in charge of capturing events and updating it. The status may be consulted from any program function at any time.

15.14. File operations

Although in *"Files and directories"* (page 177) we already saw how to access the file system at a low level, sometimes certain high-level operations are necessary on the data on disk. The mere act of completely deleting a directory has many individual low-level operations associated with it. The *Core* library, through `<hfile.h>` provides certain utilities that can simplify our lives at certain times.

- Use `hfile_dir_create` to create a directory, also creating its predecessors if they don't exist.
- Use `hfile_dir_destroy` to recursively delete a directory and all its contents.
- Use `hfile_dir_sync` to synchronize the contents of two directories. Something similar to Unix `rsync`.
- Use `hfile_dir_loop` to go deep through a directory (Listing 15.41).
- Use `hfile_buffer` to load the contents of a file into memory.

Listing 15.41: Using **hfile_dir_loop** to loop through a three-level directory.

```
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);
```

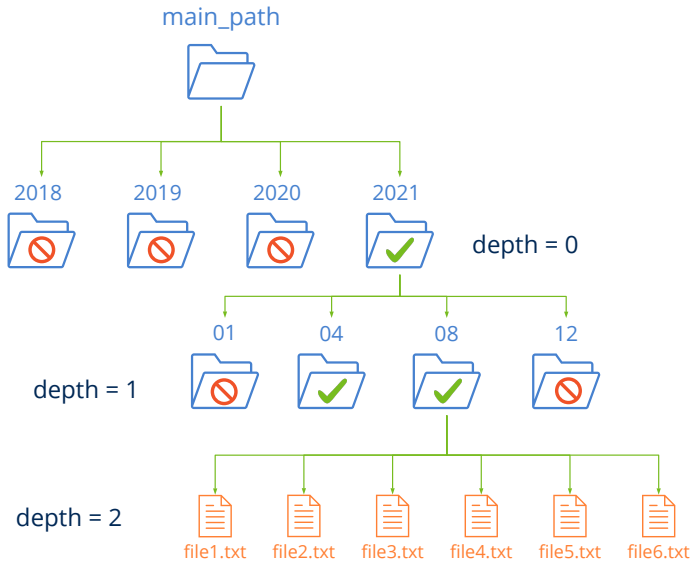


Figure 15.39: Representation of directory of (Listing 15.41).

15.15. Resource packs

15.16. Dates

A series of functions are included within *core* to work with dates.

- Use `date_system` to get the system date.
- Use `date_add_seconds` to increment a given date.
- Use `date_cmp` to compare two dates.

15.17. Clocks

Simple objects that allow us, in a comfortable way, to measure the time span between two instants. They are also useful for launching events at regular time intervals (Listing 15.42).

Listing 15.42: 25fps animation.

```
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);
```

Chapter 16

Geom2D library

16.1. Geom2D

We are facing a geometric calculation library in two dimensions. Geom2D allows working with primitives in the real plane: Points, vectors, transformations, curves and surfaces.

It offers only mathematical functionality, that is, it does not define any type of representation or drawing operation. It only depends on *"Core"* (page 187) library (Figure 16.1), so it can be used in both desktop applications and command line utilities. All types and functions are defined in simple `(float)` and `double` precision, in addition to being able to make use of C++ *"Math templatesMath templates"* (page 53).

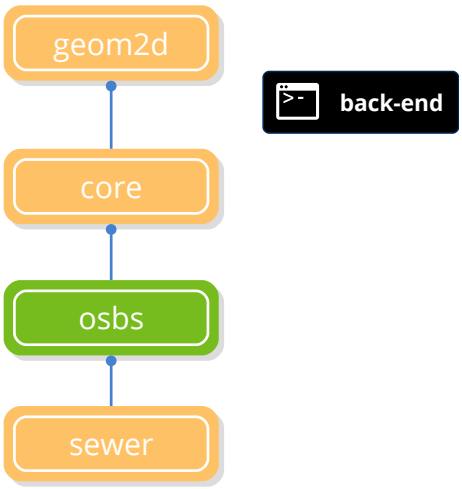


Figure 16.1: Dependencies of *geom2d*. See *"NAppGUI API"* (page 145).

All geometric elements are based on **(x, y)** coordinates in the plane. Geom2D does not assume how these coordinates will be interpreted. That will depend on the reference system defined by the application. The most used are the Cartesian and the screen (Figure 16.2), although others systems could be used where appropriate (Figure 16.3).

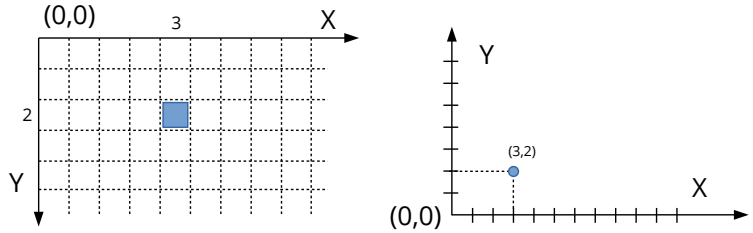


Figure 16.2: Interpretation of the coordinate (3,2) on monitors (left) and on the Cartesian plane (right).

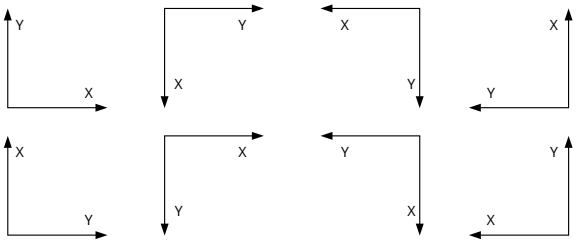


Figure 16.3: Different 2D coordinate systems.

16.2. 2D Vectors

Vector (`V2Df`, `V2Dd`) is the most elementary geometric element. It represents a point, a direction or displacement by its two components **x** and **y** (Figure 16.4).

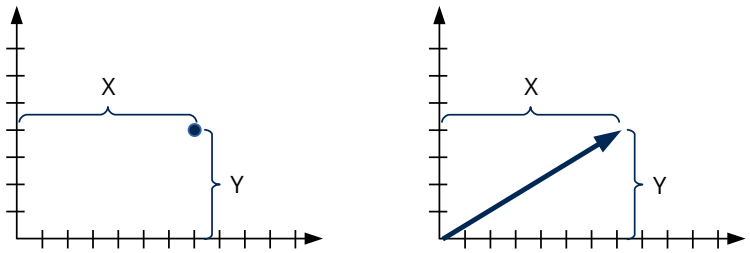


Figure 16.4: A 2D vector represents a position or a displacement in the plane.

The Vectorial Albegra defines a series of basic operations: Addition, negation, multiplication by a scalar, module and normalization (Formula 16.1). The visual representation of these operations is in (Figure 16.5).

$$
\begin{aligned}
\vec{v} &= \vec{a} + \vec{b} \\
&= (a.x + b.x, a.y + b.y) \\
\vec{v} &= p2 - p1 \\
&= (p2.x - p1.x, p2.y - p1.y) \\
-\vec{a} &= (-a.x, -a.y) \\
\vec{v} &= s \cdot \vec{a} \\
&= (s \cdot a.x, s \cdot a.y) \\
|\vec{a}| &= \sqrt{a.x^2 + a.y^2} \\
\hat{a} &= \left(\frac{a.x}{|\vec{a}|}, \frac{a.y}{|\vec{a}|} \right)
\end{aligned}
$$

Formula 16.1: Elementary vector albegra.

- Use `v2d_addf` to add two vectors.
- Use `v2d_subf` to subtract two vectors.
- Use `v2d_mulf` to multiply by a scalar.
- Use `v2d_lengthf` to calculate the modulus of a vector.
- Use `v2d_normf` to normalize a vector.

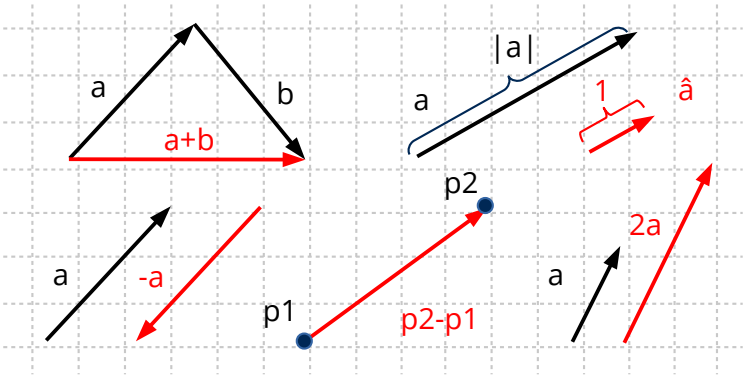


Figure 16.5: Geometric interpretation of basic operations with vectors.

16.2.1. CW and CCW angles

The angle of rotation of a vector will always be expressed in **radians** and the **positive direction** corresponds to the rotation from the **X axis to the Y axis**. Normally the counterclockwise direction is associated as positive and the clockwise direction negative. This is true in Cartesian coordinates but not in other types of reference systems, such as images or monitors (Figure 16.6). We must bear this in mind to avoid confusion, something that happens relatively frequently. The same criterion is applied when calculating the perpendicular vector, differentiating between positive and negative.

- Use `v2d_anglef` to get the angle between two vectors.
- Use `v2d_rotatef` to apply a rotation to a vector.
- Use `v2d_perp_posf` to calculate the positive perpendicular vector.

*To avoid confusion, remember that the positive direction is the one that rotates from the X axis to the Y axis. It will be **counterclockwise direction** in Cartesian coordinates and **clockwise direction** in screen coordinates.*

16.2.2. Vector projection

Another operation used quite frequently in geometry is the projection of points onto a vector. Intuitively, we can see it as the point on the vector closest to the original point and that it will always be on the perpendicular line. We will calculate it with the dot product (Formula 16.2) and its value (scalar) will be the distance from the origin to the projection in the direction of the vector (Figure 16.7).

- Use `v2d_dotf` to calculate the dot product of two vectors.

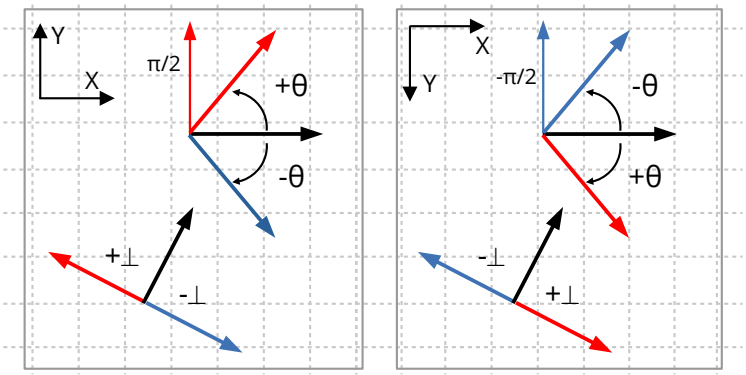


Figure 16.6: Rotation of a vector in Cartesian and screen systems.

$$
\begin{aligned}
proj_{\vec{v}}(p) &= \frac{v.x \cdot p.x + v.y \cdot p.y}{|\vec{v}|} \\
proj_{\vec{4,3}}(1,2) &= \frac{4 \cdot 1 + 3 \cdot 2}{5} = 2 \\
proj_{\vec{4,3}}(2,-2) &= 0.4 \\
proj_{\vec{4,3}}(5,1) &= 4.6 \\
proj_{\vec{4,3}}(-3,1) &= -1.8
\end{aligned}
$$

Formula 16.2: Projection of several points in a vector.

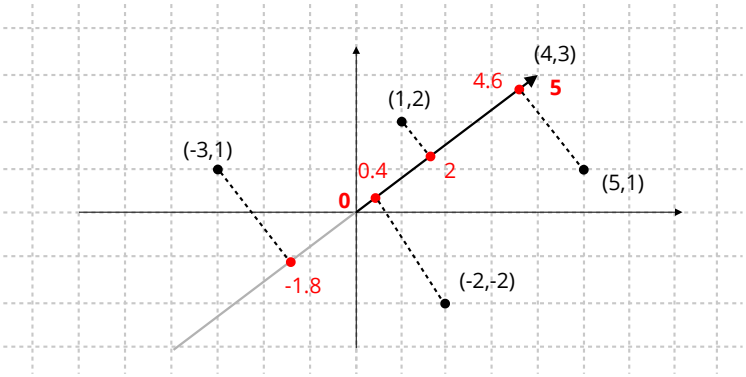


Figure 16.7: Geometric interpretation of projections.

If we are interested in the relative position between different projections, we can avoid dividing by the vector's modulus, which is more computationally efficient by not calculating square roots.

16.3. 2D Size

The `S2Df`, `S2Dd` structure stores information about a measure or size in two dimensions using its fields `width` and `height`.

- Use `s2df` to compose a measure through its elementary fields.

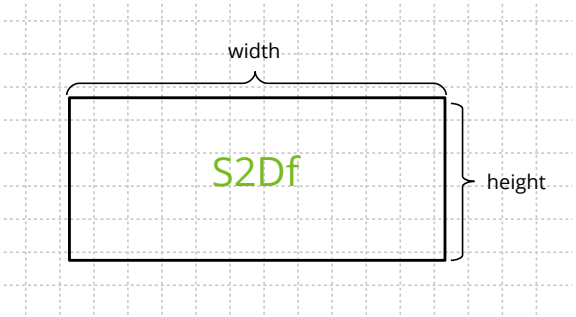


Figure 16.8: Size2D.

16.4. 2D Rectangles

A rectangle (or *frame*) (`R2Df`, `R2Dd`) (Figure 16.9) is used to locate elements in user interfaces or other 2D systems through a point of origin `V2Df` and a size `S2Df`. They can also be used in clipping operations, when optimizing drawing tasks.

- Use `r2d_collidef` to determine if two rectangles collide.
- Use `r2d_clipf` to determine if a rectangle is visible within an area.
- Use `r2d_joinf` to join the two rectangles.

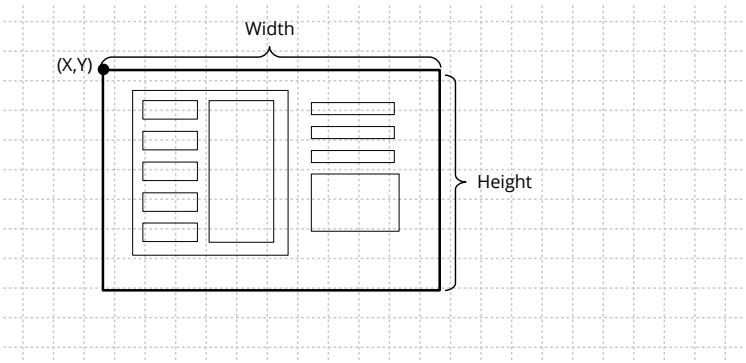


Figure 16.9: Positioning GUI elements using rectangles.

16.5. 2D Transformations

Affine transformations are a type of mathematical operation that allows coordinate changes between different reference systems. For example in (Figure 16.10) **(a)** we construct a polygon expressing the coordinates of its vertices in a Cartesian system: [(4,1), (2,5), (-3,5), (-4,2), (0, -3)]. Now let's imagine that we want to draw several instances of our model on a plane, each with a different position, orientation and size (Figure 16.10) **(b)**. We would need to calculate the coordinates of the points of the polygon in the new locations, in order to correctly draw the lines that delimit them.

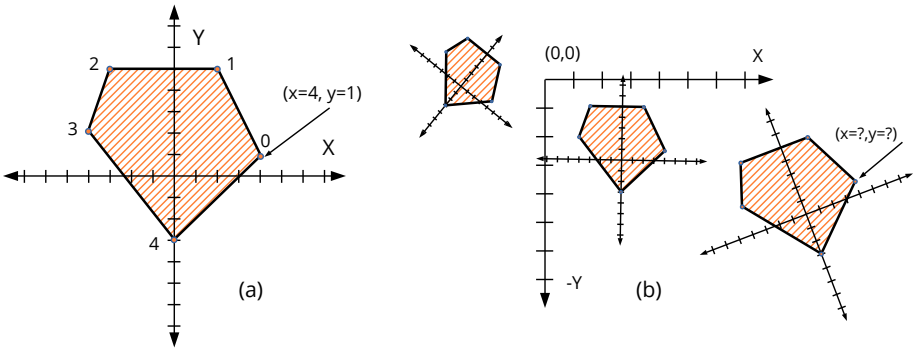


Figure 16.10: Geometric model (a) Expressed in a Cartesian system. (b) After applying transformations.

Vector Algebra gives us a powerful tool with which the relationship between two systems can be expressed using six real numbers (Figure 16.11). The first four values correspond to a 2x2 matrix with the coordinates of the vectors X=[1,0] and Y=[0,1] in the new reference system. This matrix integrates a possible rotation and scaling of the axes. The last two values indicate a displacement in the origin of coordinates. In (Formula 16.3) we have the mathematical development to transform the point [4.1] to a new base rotated 25° with respect to the origin and displaced 11 units on the X axis and -5 on the Y axis. Applying the same operation to all points, we would transform the object.

16.5.1. Elementary transformations

In principle, any combination of values [i.x, i.y, j.x, j.y, p.x, p.y] would provide a valid transformation, although if we do not choose them with certain criteria we will obtain aberrations that are not very useful in practice. The most used transformations in graphic and engineering applications are (Figure 16.12) (Figure 16.13) (Formula 16.4):

- Translation **(a)**: Moves the origin of the object to another point.
- Rotation **(b)**: Rotates the object on the origin of its local system.
- Scaling **(c)**: Change the size. If $sx < 1$, reduce. $sx > 1$, increase. $sx = 1$, does not

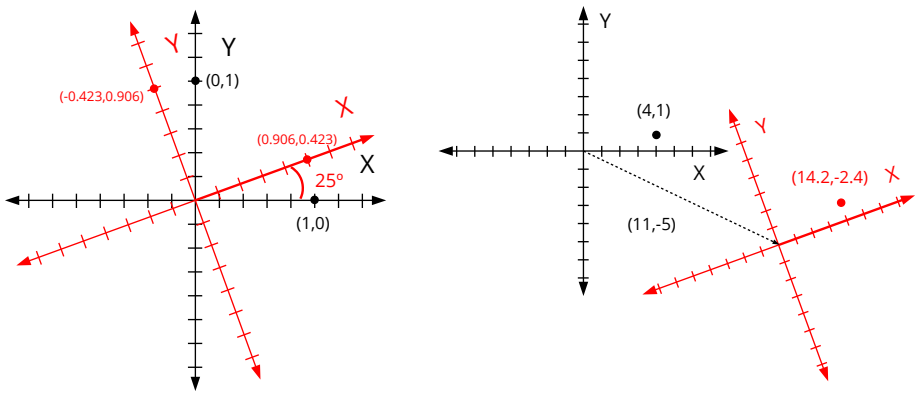


Figure 16.11: Change of base. Relationship of a point in two different reference systems.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} i.x & j.x \\ i.y & j.y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p.x \\ p.y \end{bmatrix} \\
&= \begin{bmatrix} 0.906 & -0.423 \\ 0.423 & 0.906 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \end{bmatrix} + \begin{bmatrix} 11 \\ -5 \end{bmatrix} \\
&= \begin{bmatrix} 3.2 \\ 2.6 \end{bmatrix} + \begin{bmatrix} 11 \\ -5 \end{bmatrix} \\
&= \begin{bmatrix} 14.2 \\ -2.4 \end{bmatrix}
\end{aligned}
$$

Formula 16.3: Point [4,1] transformation.

vary. In non-uniform scales, *sx* and *sy* have different values, which will produce a distortion in the aspect ratio.

- Identity **(d)**: It is the null transformation. When applied, the vectors remain unchanged.

16.5.2. Composition of transformations

It is possible to compose or accumulate transformations by matrix multiplication (Formula 16.5). The usual thing in 2d models will be to obtain the final location of an object from the elementary transformations translation, rotation and scaling. The accumulation is also useful for positioning elements in hierarchical structures, where the location of each object depends directly on that of its upper node (parent).

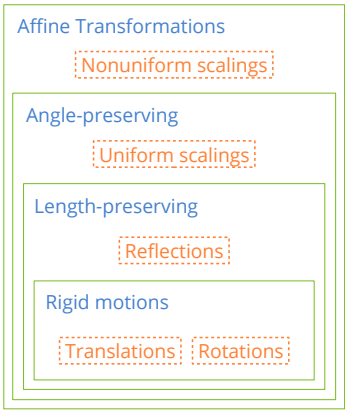


Figure 16.12: Classification of affine transformations.

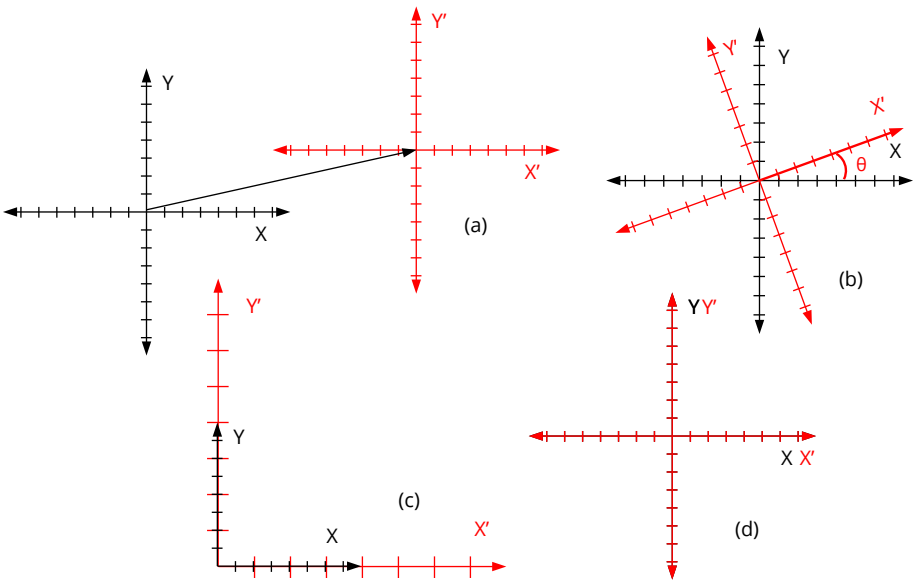


Figure 16.13: Geometric representation of elementary transformations. (a) Translation, (b) Rotation, (c) Scaling, (d) Identity.

- Use `t2d_movef` to add a displacement to an existing transformation.
- Use `t2d_rotatef` to add a rotation.
- Use `t2d_scalef` to add a scaling.
- Use `t2d_multf` to add a transformation.
- Use `t2d_vmultf` to apply a transformation to a vector.
- Use `t2d_vmultnf` to apply a transformation to several vectors.
- Use `kT2D_IDENTf` to reference the identity transformation.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p.x \\ p.y \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

Formula 16.4: Translation, Rotation, Scaling and Identity.

$$
\begin{aligned}
\begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} i_d.x & j_d.x \\ i_d.y & i_d.y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p_d.x \\ p_d.y \end{bmatrix} \\
i_d.x &= i_1.x \cdot i_2.x + j_1.x \cdot i_2.y \\
i_d.y &= i_1.y \cdot i_2.x + j_1.y \cdot i_2.y \\
j_d.x &= i_1.x \cdot j_2.x + j_1.x \cdot j_2.y \\
j_d.y &= i_1.y \cdot j_2.x + j_1.y \cdot j_2.y \\
p_d.x &= i_1.x \cdot p_2.x + j_1.x \cdot p_2.y + p_1.x \\
p_d.y &= i_1.y \cdot p_2.x + j_1.y \cdot p_2.y + p_1.y
\end{aligned}
$$

Formula 16.5: Composition of two arbitrary transformations.

Matrix multiplication is not commutative, but the order in which the operations are applied will affect the final result. For example in (Figure 16.14) **(a)**, the origin has been moved and then applied a rotation. In (Figure 16.14) **(b)** it has been done on the contrary, first rotate and then move.

Listing 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);
```

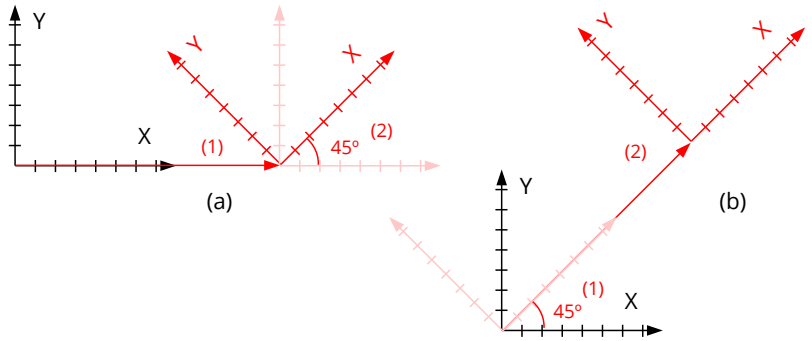


Figure 16.14: Effect of the order of application of transformations.

16.5.3. Decomposition and inverse

Any chain of translations, rotations, and scales defines an affine reference frame that can be expressed in terms of a single traslation, rotation, and scale (Figure 16.15). We can "undo" this transformation and return to the origin through the inverse transformation (Listing 16.2).

- Use `t2d_decomposef` to get the components of a transformation.
- Use `t2d_inversef` to get the inverse transformation.

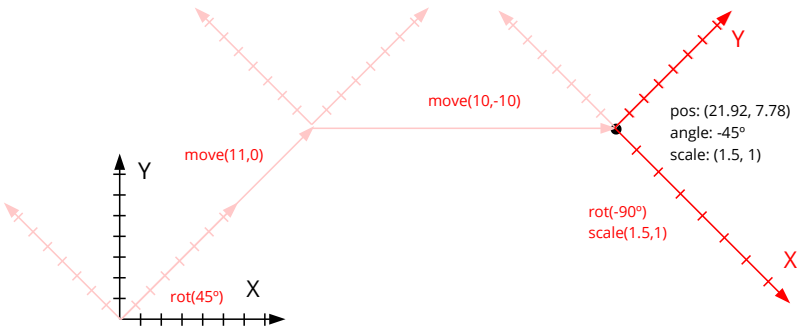


Figure 16.15: Transformation chain and final system.

Listing 16.2: Components of a reference and inverse system.

```
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);
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. 2D Segments

Segments are fragments of a line between two points **p0** and **p1** (Figure 16.16). They are the simplest geometric primitives, after vectors. We define the **t** parameter as the normalized position within the segment. Values between `0` and `1` will correspond to internal points of the segment, with the limits `t=0` (`p0`) and `t=1` (`p1`). Out of this range we will have the points outside the segment, but within the line that contains it. For example `t=2` would be the point after `p1` located at a distance equal to the length of the segment.

- Use `seg2d_lengthf` to get the length of the segment.
- Use `seg2d_close_paramf` to get the value of the parameter closest to a certain point.
- Use `seg2d_evalf` to get the point from the parameter.
- Use `seg2d_sqdistf` to get the distance (squared) between two segments.

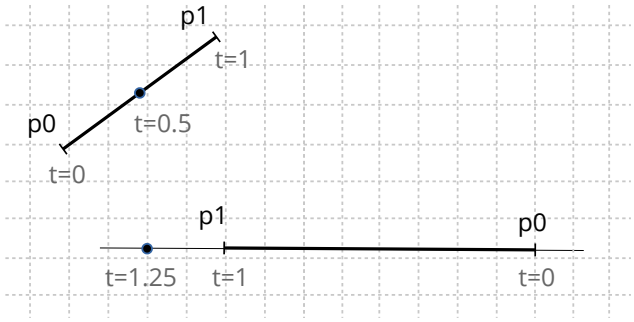


Figure 16.16: Segments in the plane.

16.7. 2D Circles

Circles allow us to group a set of points within the same container volume. Collision detection will be performed optimally since it is the geometric test that requires the fewest operations. Given a set of points, we can calculate the container circle in various ways (Figure 16.17) depending on the precision and speed needed.

- Use `cir2d_from_boxf` to get the circle from a 2D box.
- Use `cir2d_minimumf` to obtain the circle of minimum radius from a set of points.
- Use `cir2d_from_pointsf` to obtain the circle from the the set average. More balanced option in terms of precision/performance.

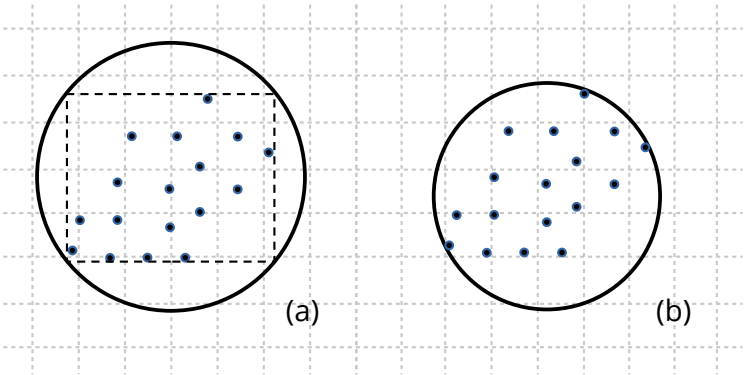


Figure 16.17: Container circle: From BBox **(a)**. Minimum radius **(b)**.

16.8. 2D Boxes

2D containers or *(Bounding boxes)* delimit the area of the plane occupied by different geometric elements (Figure 16.18). They are useful in the collision detection or *clipping operations*, which prevent non-visible figures from being drawn, improving overall performance.

- Use `box2d_from_pointsf` to create a 2D box from a set of points.
- Use `box2d_addnf` to change dimensions based on new points.
- Use `box2d_segmentsf` to get the four segments that delimit the box.

16.9. 2D Oriented Boxes

Oriented Bounding Boxes are 2D boxes that can rotate about their center (Figure 16.19), so they will no longer be aligned with axes. Here the collision detection is somewhat

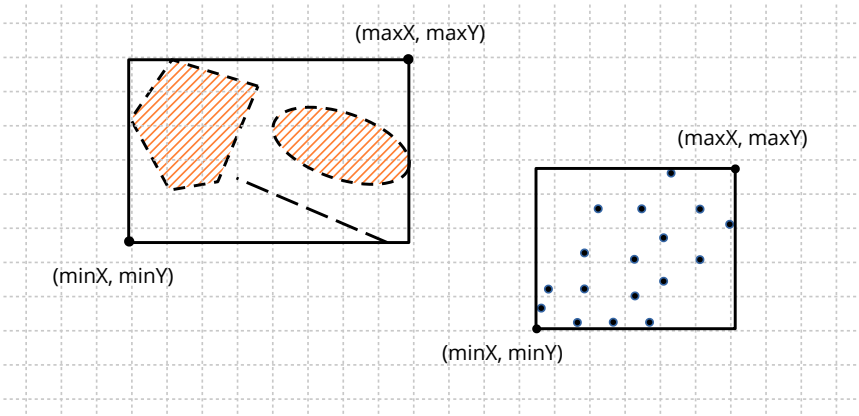


Figure 16.18: 2D boxes as a container for other objects.

complicated compared to 2D Axis-Aligned boxes, in exchange for providing a better fit against elongated objects that can rotate in the plane.

- Use `obb2d_from_pointsf` to create an oriented box from a set of points.
- Use `obb2d_from_linef` to create an oriented box from a segment.
- Use `obb2d_transformf` to apply a 2D transformation to the box.
- Use `obb2d_boxf` to get the aligned box containing the oriented box.

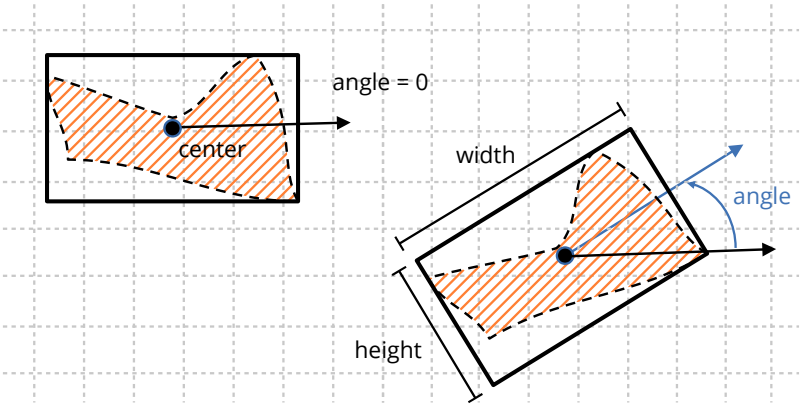


Figure 16.19: 2D oriented boxes.

We can obtain relevant parameters of an arbitrary set of points from the covariance matrix (Formula 16.6), which is geometrically represented by an ellipse rotated in the plane and centered on the mean of the distribution (Figure 16.20). This analysis allows `obb2d_from_pointsf` to calculate the 2D box associated with the distribution in a quite acceptable way, without becoming the optimal solution that is much more expensive in

computational terms.

$$
\begin{aligned}
\Sigma &= \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \\
\sigma_{xx} &= \frac{1}{N}\left[\sum_{i=1}^{N} {x_i}^2\right] - \mu_x^2 \\
\sigma_{yy} &= \frac{1}{N}\left[\sum_{i=1}^{N} {y_i}^2\right] - \mu_y^2 \\
\sigma_{xy} &= \frac{1}{N}\left[\sum_{i=1}^{N} x_i y_i\right] - \mu_x \mu_y \\
\sigma_{yx} &= \sigma_{xy} \\
\mu_x &= \frac{1}{N}\sum_{i=1}^{N} x_i \\
\mu_y &= \frac{1}{N}\sum_{i=1}^{N} y_i
\end{aligned}
$$

Formula 16.6: Calculation of the covariance matrix.

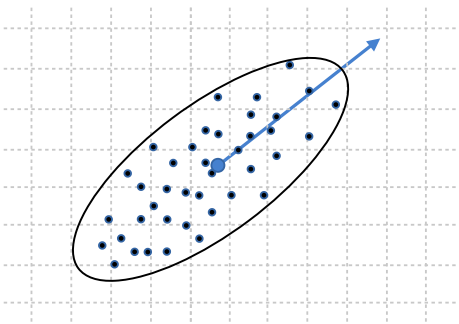


Figure 16.20: The covariance matrix represents an ellipse rotated in the plane.

Use oriented boxes (`OBB2Df`) for "elongated" point distributions. In rounded or square cases the aligned box (`Box2Df`) can provide a volume with a smaller area.

16.10. 2D Triangles

Triangles are widely used in computational geometry, especially when performing certain calculations on polygons or surfaces. They are also the basis of most graphical APIs, so on many occasions we will need to approximate objects using triangles. The **centroid**

is the equilibrium point found at the intersection of the medians (Figure 16.21).

- Use `tri2df` to compose a triangle.
- Use `tri2d_transformf` to apply a transformation.
- Use `tri2d_centroidf` to get the center of mass.
- Use `tri2d_areaf` to calculate the area.

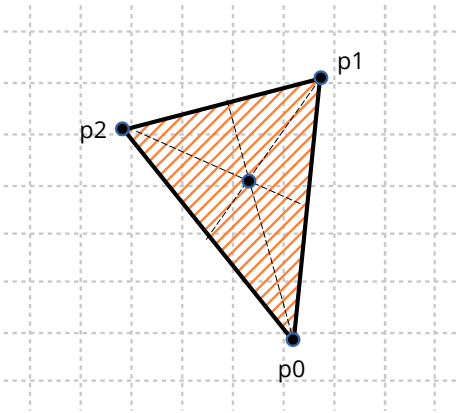


Figure 16.21: 2D triangles.

16.11. 2D Polygons

Polygons are widely versatile figures, since they allow us to define arbitrary regions delimited by rectilinear segments. Geom2D supports so-called **simple polygons**, which are those whose sides cannot intersect each other.

- Use `pol2d_createf` to create a polygon from the path formed by its vertices.
- Use `pol2d_ccwf` to get the direction of path rotation. See *"CW and CCW anglesCW and CCW angles"* (page 238).
- Use `pol2d_transformf` to apply a transformation to the polygon.
- Use `pol2d_areaf` to get the area.
- Use `pol2d_boxf` to get the polygon boundaries.

We can classify the polygons into three large groups (Figure 16.22):

- **Convex:** The most "desired" from the point of view of calculation simplicity. They are those where any segment that joins two interior points, is totally within the polygon.
- **Concave:** Or not convex. The opposite of the above. It is one that has an interior angle of more than 180 degrees.

- **Weakly:** It is one that presents holes through "cut" segments where two vertices are duplicated to allow access and return of each hole. It is an easy way to empty the interior of regions without requiring multiple cycles. The calculation of areas and collisions will take into account these cavities.

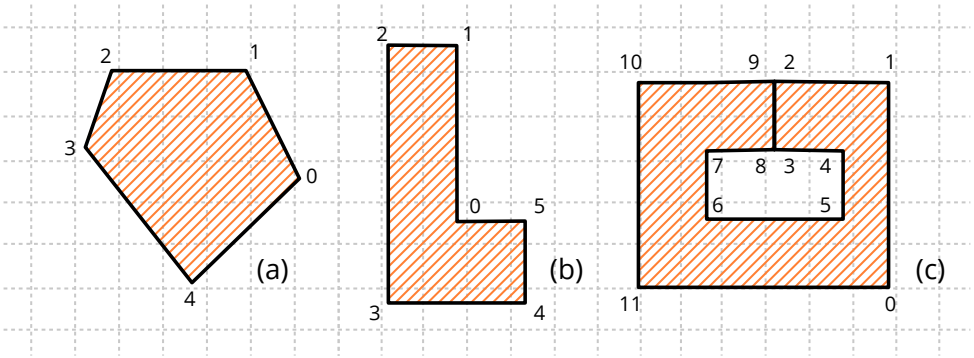


Figure 16.22: 2D polygons. (a) Convex, (b) Concave, (c) Weak. All of them defined counter-clockwise.

16.11.1. Polygon center

It is difficult to define a central point in a figure as irregular as a polygon can be. Normally we will interpret as such the centroid or **center of mass** but, in non-convex cases, this point can be located outside the polygon. In labeling tasks, it is necessary to have a representative point that is within the figure. We consider the **visual center** to be that point within the polygon located at a maximum distance from any edge (Figure 16.23). In convex polygons it will coincide with the centroid.

- Use `pol2d_centroidf` to get the centroid of the polygon.
- Use `pol2d_visual_centerf` to get the visual center of the polygon. It implements an adaptation of the **polylabel** algorithm of the MapBox[1] project.

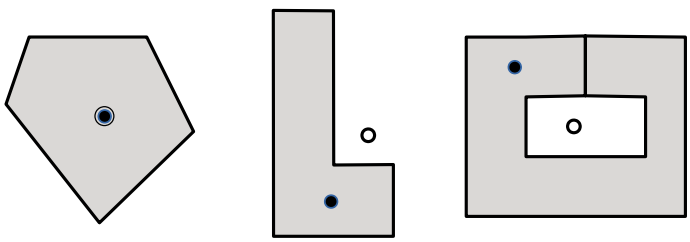


Figure 16.23: "Central" point of a polygon. Line: Centroid, Fill: Visual or Label Center.

[1] `https://github.com/mapbox/polylabel`

16.11.2. Polygon decomposition

Certain calculations or rendering tasks can be considerably optimized if we reduce the complexity of the geometry to be treated. Decomposing a polygon is nothing more than obtaining a list of simpler polygons whose union is equivalent to the original figure (Figure 16.24). As an inverse operation, we would have the calculation of the **convex hull**, which is obtaining the convex polygon that encloses a set of arbitrary points (Figure 16.25).

- Use `pol2d_trianglesf` to get a list of the triangles that make up the polygon.
- Use `pol2d_convex_partitionf` to get a list of convex polygons equivalent to the polygon.
- Use `pol2d_convex_hullf` to create a convex polygon that "wraps" a set of points.

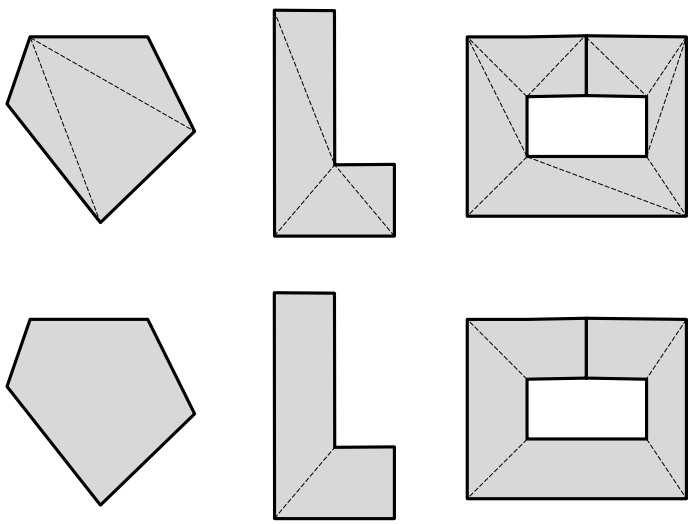


Figure 16.24: Decomposition of a polygon by triangulation or convex components.

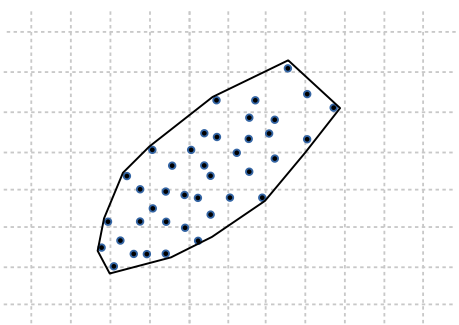


Figure 16.25: Convex hull of a set of points.

16.12. 2D Collisions

Collision detection is responsible for studying and developing algorithms that check if two geometric objects intersect at some point. As the general case would be quite complex to implement and inefficient to evaluate, a series of **collision volumes** (Figure 16.26) are defined that will enclose the original sets and where the tests can be significantly simplified. The use of these most elementary forms is usually known as *broad phase collision detection* (Figure 16.27), since it seeks to detect "non-collision" as quickly as possible. In *"Hello 2D Collisions!"* (page 569) you have an example application.

- Use `col2d_poly_obbf` to detect the collision between an oriented box and a polygon.
- Use `col2d_tri_trif` to detect the collision between two triangles.
- Use `col2d_circle_segmentf` to detect the collision between a circle and a segment.

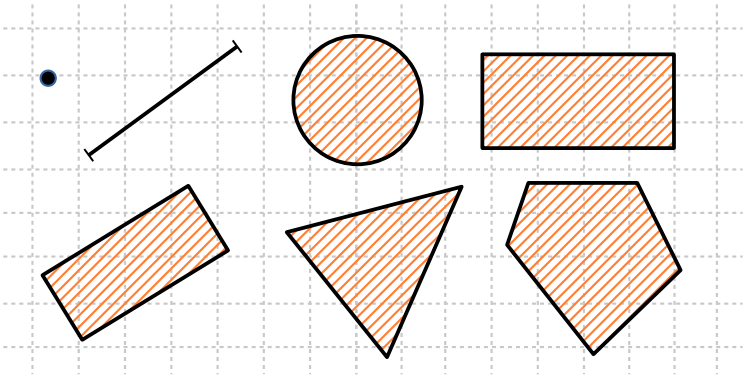


Figure 16.26: 2D Collision Volumes: Point, Segment, Circle, Box, Oriented Box, Triangle, and Polygon.

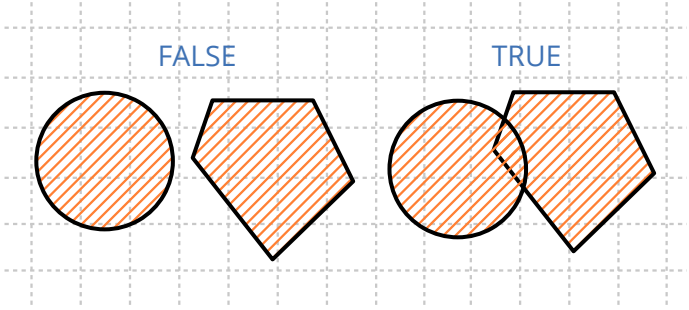


Figure 16.27: Broad phase collision detection.

Col2D provides functions to check each pair of previously presented collision volumes. Most of these methods use the **Separation Axis Theorem** (Figure 16.28). This theorem indicates, in essence, that if it is possible to find a line where the projections of the vertices do not intersect, then the figures do not intersect. In the specific case of convex polygons, it is only necessary to evaluate **n lines**, where n is the number of sides of the polygon.

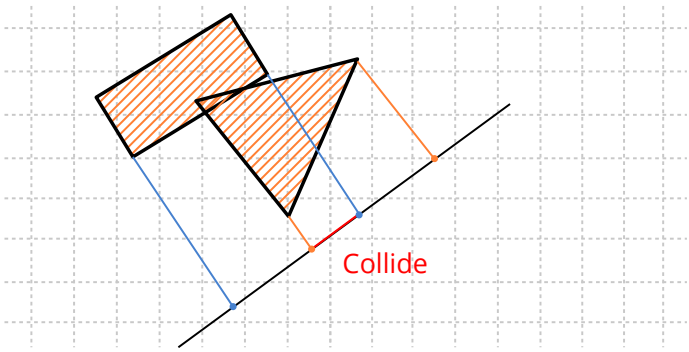


Figure 16.28: Separation axis theorem detecting a collision.

Chapter 17

Draw2D library

17.1. Draw2D

The *Draw2D* library integrates all the functionality necessary to create two dimensions vector graphics. It depends directly on *Geom2D* (Figure 17.1) and, as we will see later, drawing does not imply having a graphical user interface in the program. It is possible to generate images using an internal memory buffer, without displaying the result in a window.

- *"2D Contexts"* (page 257).
- *"Drawing primitives"* (page 265).
- *"Colors"* (page 277) and *"Palettes"* (page 279).
- *"Pixel Buffer"* (page 280) and *"Images"* (page 283).
- *"Typography fonts"* (page 288).

This library connects directly to the native technologies of each operating system: **GDI+** on Windows systems, **Quartz2D** on macOS and **Cairo** on Linux. In essence, *draw2d* offers a common and light interface so that the code is portable, delegating the final work in each of them. With this we guarantee three things:

- Efficiency: These APIs have been tested for years and are maintained by system manufacturers.
- Presence: They are integrated as stardard in all computers, so it is not necessary to install additional software.
- Performance: The programs are smaller since they do not require linking with special routines for handling graphics, typography or images.

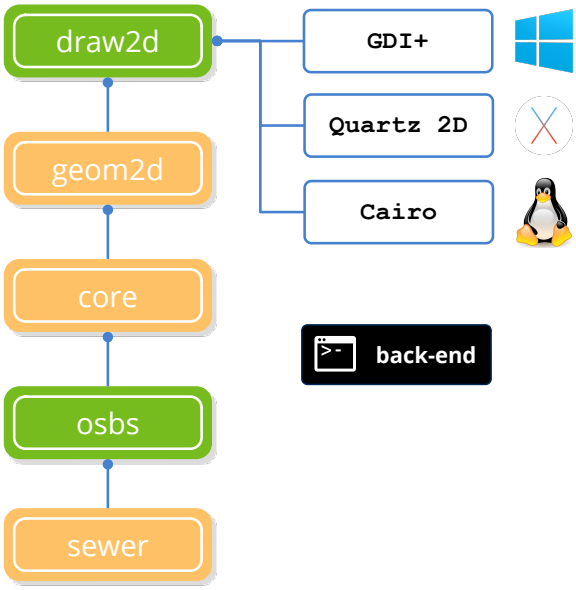


Figure 17.1: Dependencies of *draw2d*. See *"NAppGUI API"* (page 145).

17.2. 2D Contexts

Vector graphics are composed of basic primitives such as lines, circles, text, etc, using the painter's algorithm (Figure 17.2): Incoming operations overlap existing ones. The result is stored in an intermediate buffer known as *canvas* or *surface*. This drawing surface is part of an object called **context** that also maintains certain parameters related to the appearance of primitives: Colors, line attributes, reference system, gradients, etc..

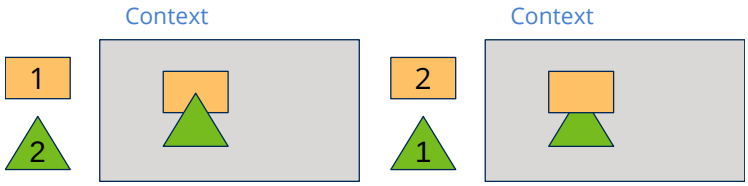


Figure 17.2: Painter's algorithm. New objects will overlap existing ones.

One of the advantages of working with parametric shapes is that image scaling can be done without loss of quality (Figure 17.3). This is because the conversion to pixels, a process called rasterization (Figure 17.4), is done in real time and constantly adjusts to the change of vectors. In bitmap images, an increase in size has associated a loss of quality.

Draw2D allows working with two types of 2D contexts (Figure 17.5).

- Window context. The destination will be an area within a user interface window managed by a `View` control. This control maintains its own drawing context and

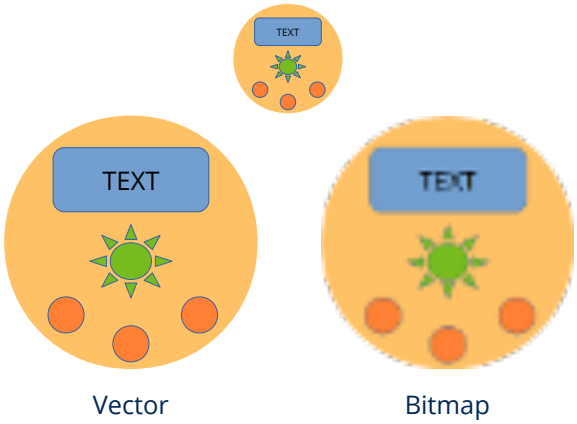


Figure 17.3: Vector scaling and bitmap scaling.

Figure 17.4: Rasterization of a circle.

sends it "ready to use" through the `EvDraw` event (Listing 17.1).

Listing 17.1: Drawing in a window.

```
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));
```

- Image context. Here the drawing commands will be directly dumped into memory to subsequently obtain an image with the final result (Listing 17.2).

Listing 17.2: Draw on an image.

```
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);
}
```

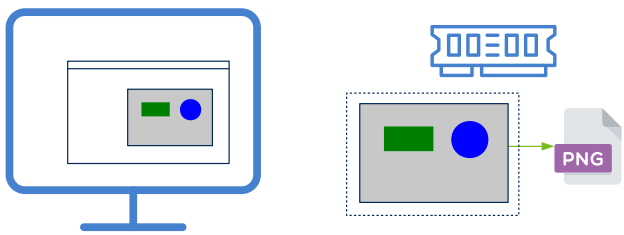


Figure 17.5: Window and image contexts.

As we can see, the drawing itself is done in the same way, the only thing that changes is how we obtained the context (`DCtx`). This allows us to write generic graphic routines without worrying about the destination of the final result. In the example *DrawImg*[1] you have a practical step-by-step development of the use of contexts. The images that accompany the rest of the chapter have been obtained from this application.

Because it is not necessary to have a window to draw, Draw2d can be used in console applications to compose or edit images in an automated way.

17.2.1. Reference systems

The drawing origin of coordinates is located in the upper left corner (Figure 17.6). The positive **X** move to the left and the positive **Y** down. Units are measured in pixels (or points in *"Retina displaysRetina displays"* (page 264)). For example, the command:

```
draw_circle(ctx, ekSKFILL, 300, 200, 100);
```

will draw a circle of 100 pixel radius whose center is 300 pixels to the left and 200 pixels down from the origin. This initial system is called **identity** since it has not yet been manipulated, as we will see below.

[1] `https://nappgui.com/en/howto/drawimg.html`

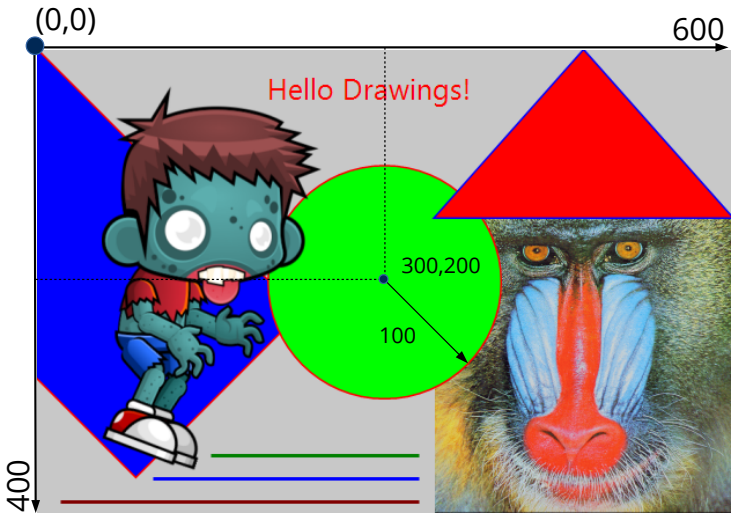


Figure 17.6: **Identity** reference system in 2D contexts.

Although the initial scale is in pixels, we must banish the idea that we are directly manipulating pixels when drawing. Drawing contexts use floating point coordinates. For example, drawing a line between the points (0.23, 1.432) and (-45.29, 12.6756) is perfectly valid. Transformations and antialiasing may slightly alter the position or thickness of certain lines. Nor should we expect "identical" pixel-level results when migrating applications to different platforms, since each system uses its own rasterization algorithms. We must think that we are drawing on the real plane. To directly manipulate the pixels of an image, see `image_pixels` *and* `image_from_pixels`*.*

This initial reference system can be manipulated by *"2D Transformations"* (page 241). The most common transformations in graphics are: Translations (Figure 17.7), Rotations (Figure 17.8) and Scaling (Figure 17.9).

- `draw_matrixf` will change the context reference system.

Listing 17.3: Coordinate origin translation 100 units in both directions.

```
T2Df t2d;
t2d_movef(&t2d, kT2D_IDENTf, 100, 100);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

Listing 17.4: Coordinate origin rotation 15 degrees.

```
T2Df t2d;
t2d_rotatef(&t2d, kT2D_IDENTf, 15 * kBMATH_DEG2RADf);
draw_matrixf(ctx, &t2d);
```

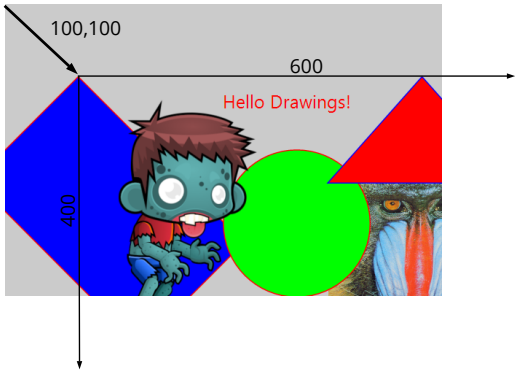


Figure 17.7: Translation (Listing 17.3).

```
i_draw(...);
```

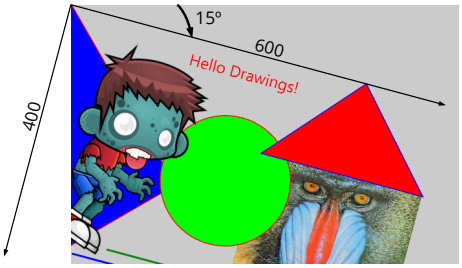


Figure 17.8: Rotation (Listing 17.4).

Listing 17.5: Scaling, size halving.

```
T2Df t2d;
t2d_scalef(&t2d, kT2D_IDENTf, .5f, .5f);
draw_matrixf(ctx, &t2d);
i_draw(...);
```

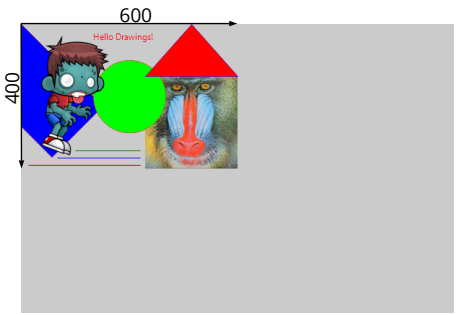


Figure 17.9: Scaling (Listing 17.5).

The transformations can be accumulated, but we must bear in mind that they are not commutative operations, but that the order in which they are applied will influence the final result. For example in (Figure 17.10) we observe that the drawing has moved `(100, 50)` pixels, instead of `(200, 100)`, because the translation is affected by previous

scaling. More details at *"Composition of transformationsComposition of transformations"* (page 242).

Listing 17.6: Composition of transformations.

```
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(...);
```

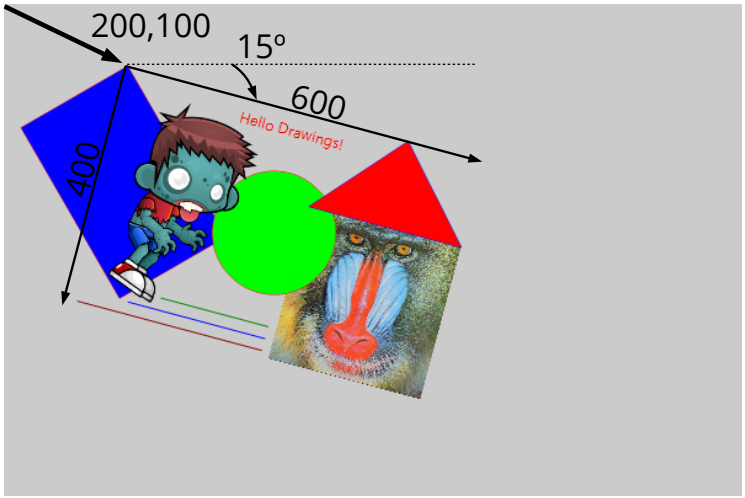


Figure 17.10: Composition of transformations (Listing 17.6).

17.2.2. Cartesian systems

There is a dichotomy when drawing in 2D: On the one hand, traditionally desktop systems and digital images place the origin of coordinates in the upper left corner with the Y axis growing down (Figure 17.11). On the other hand, the Cartesian systems used in geometry place it in the lower left corner, with Y growing up. This creates a dilemma about whether one system is better than another.

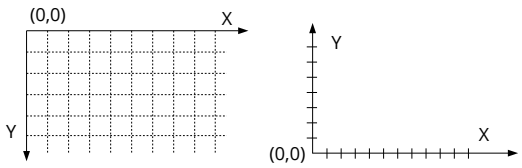


Figure 17.11: 2D system on monitors (left) and Cartesian (right).

The answer is clearly no. Even in the same drawing, we may need to combine both depending on the element we are treating. For texts and images, the screen system is more

intuitive since it reproduces the paper or canvas of the physical world. For mathematical functions, bar graphs, plans and other aspects related to the technical world, the Cartesian is much more comfortable and natural.

- `draw_matrix_cartesianf` set the context reference system in Cartesian coordinates. In (Figure 17.12) we have used a 6x4 unit Cartesian system mapped onto a 600x400 pixel window.

Listing 17.7: Drawing in Cartesian coordinates.

```
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);
```

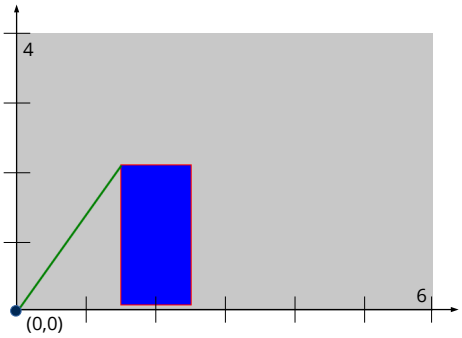


Figure 17.12: Cartesian coordinates (Listing 17.7).

17.2.3. Antialiasing

Given the discrete nature of monitors and digital images, a staggered effect (sawtooth) is produced by transforming vector primitives to pixels (Figure 17.13). This effect becomes less noticeable as the resolution of the image increases, but still the "pixelated" remains patent. The **antialiasing**, is a technique that reduces this step effect by slightly varying the colors of the pixels in the environment near the lines and contours (Figure 17.14). With this, the human eye can be deceived by blurring the edges and generating images of greater visual quality. In return we have the cost in the performance of applying it, although for years that the calculations related to antialiasing are made directly in hardware (Figure 17.15), so the impact will be minimal.

- `draw_antialias` allows to activate or deactivate the antialiasing calculations.

Figure 17.13: Antialiasing off.

Figure 17.14: Antialiasing on.

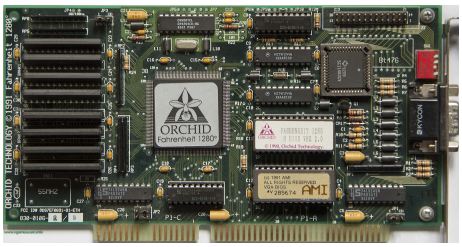


Figure 17.15: Orchid Fahrenheit 1280 (1992). One of the first cards that incorporated 2d graphic acceleration.

17.2.4. Retina displays

At the end of 2014 Apple introduced its news iMac with high resolution *Retina Display* (5120x2880). Normally, these monitors work in **scaled** mode (2560x1440) allowing double density pixels (Figure 17.16). Apple differentiates between **points** on the screen, which are what really manipulates the application and physical pixels. Therefore, our 600x400 window will really have 1200x800 pixels on Retina computers, although the application will still "see" only 600x400 points. The operating system converts transparently. In fact, we don't have to do anything to adapt our code, since it will work in the same way on both normal iMac and those equipped with Retina monitors.

This double density will be used by the rasterizer to generate higher quality images by having more pixels in the same screen area. In (Figure 17.17) and (Figure 17.18) we see the extra quality that these models provide.

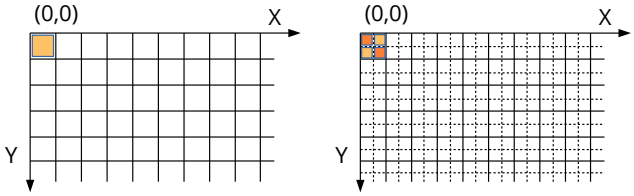


Figure 17.16: Double density pixels on *Retina Display* (right).

Figure 17.17: Normal screen (with antialiasing).

Figure 17.18: *Retina Display* (with antialiasing).

17.3. Drawing primitives

When drawing in 2D contexts we have a series of elementary shapes such as lines, figures, text and images. In *DrawHello*[2] you have the source code of the application that will accompany us throughout this section.

17.3.1. Line drawing

The most elementary operation is to draw a line between two points. In 2d contexts the lines are solid objects and not a mere row of pixels. Let's think we are using thick tip pens, where the theoretical line will always remain in the center of the stroke (Figure 17.19). We can change the shape of the endings (linecap), the joints (linejoin) and establish a pattern for dashed lines.

[2] `https://nappgui.com/en/howto/drawhello.html`

- `draw_line` will draw a line.
- `draw_polyline` will draw several connected lines.
- `draw_arc` will draw an arc.
- `draw_bezier` will draw a Bézier curve of degree 3 (cubic).
- `draw_line_color` will set the line color.
- `draw_line_width` set the line width.
- `draw_line_cap` set the style of the ends.
- `draw_line_join` set the style of the unions.
- `draw_line_dash` set a dot pattern for dashed lines.

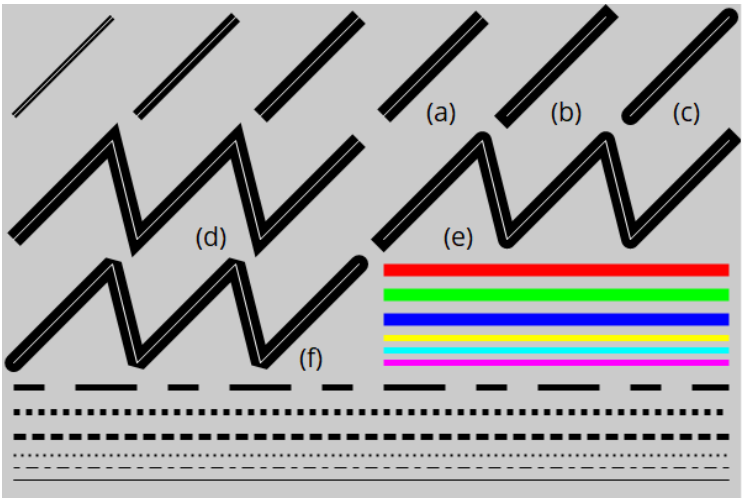


Figure 17.19: Different line styles. **(a)** `ekLCFLAT`. **(b)** `ekLCSQUARE`. **(c)** `ekLCROUND`. **(d)** `ekLJMITER`. **(e)** `ekLJROUND`. **(f)** `ekLJBEVEL`. The pattern: [5, 5, 10, 5], [1, 1], [2, 1], [1, 2], [5, 5, 10, 5], NULL.

17.3.2. Figures and borders

To draw figures or closed areas we have several commands. As we see in (Figure 17.20) we can draw the outline of the figure, its interior or both. For the contour, the established line style will be taken into account as we have seen in the previous section.

- `draw_rect` for rectangles.
- `draw_rndrect` for rectangles with rounded edges.

- `draw_circle` for circles.
- `draw_ellipse` for ellipses.
- `draw_polygon` for polygons.
- `draw_fill_color` set the area fill color.

Listing 17.8: Drawing of figures (outlines and/or fills).

```
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);
```

As we saw in *"2D Contexts"* (page 257), the order in which the operations are performed matters. It is not the same to fill and then draw the outline as vice versa. The center of the stroke will coincide with the theoretical contour of the figure.

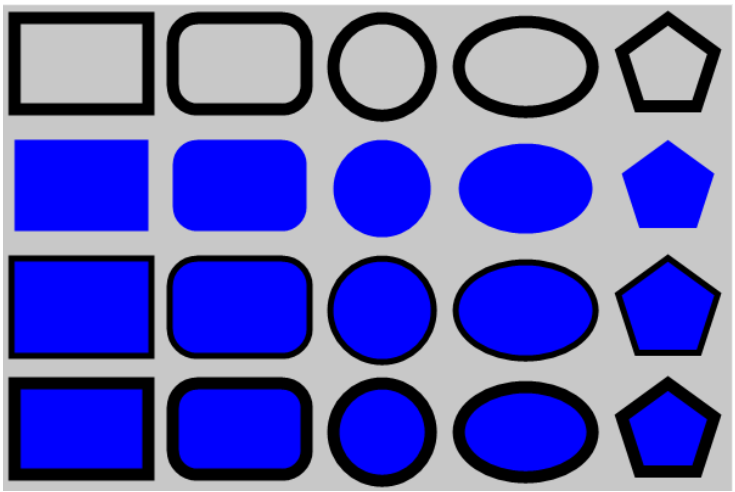


Figure 17.20: Stroke only `ekSTROKE`. Fill only `ekFILL`. First stroke, then fill `ekSKFILL`. First fill, then stroke `ekFILLSK`.

17.3.3. Gradients

Gradients allow regions to be filled using a gradient instead of a solid color (Figure 17.21). Several base colors and their relative position along a vector are defined (Listing 17.9). The positions [0, 1] correspond to the extremes and the values within this range to the possible intermediate stops. Each line perpendicular to the vector defines a uniform color that will extend indefinitely until reaching the limits of the figure to be filled.

- Use `draw_fill_linear` to activate the fill with gradients.

- Use `draw_fill_color` to return to solid color fill.

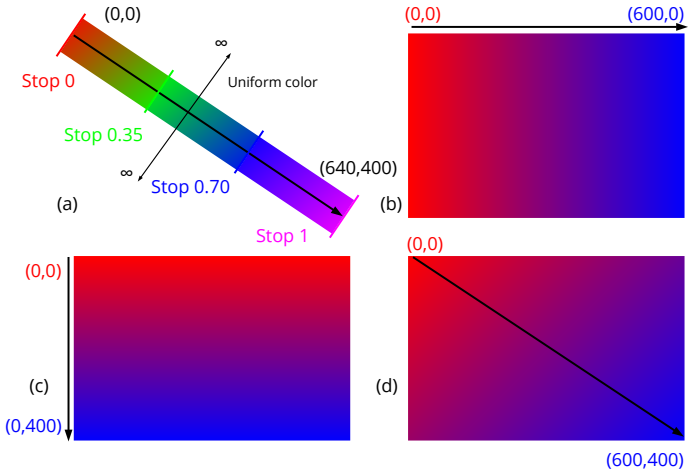


Figure 17.21: Linear gradients. The color is interpolated along a vector.

Listing 17.9: Definition of (Figure 17.21) gradients.

```
// (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. Gradient transformation

Since the gradient is defined by a vector, it is possible to set a transformation that changes the way it is applied. This matrix is totally independent from the one applied to drawing primitives `draw_matrixf`, as we saw in *"Reference systemsReference systems"* (page 259).

- Use `draw_fill_matrix` to set the gradient transformation. With this we can get several effects:
- **Global gradient:** The gradient will be applied globally to the background, and the figures will be cutouts of the same pattern (Figure 17.22). To do this we will set the identity matrix as a gradient transformation (Listing 17.10). It is defined by default.

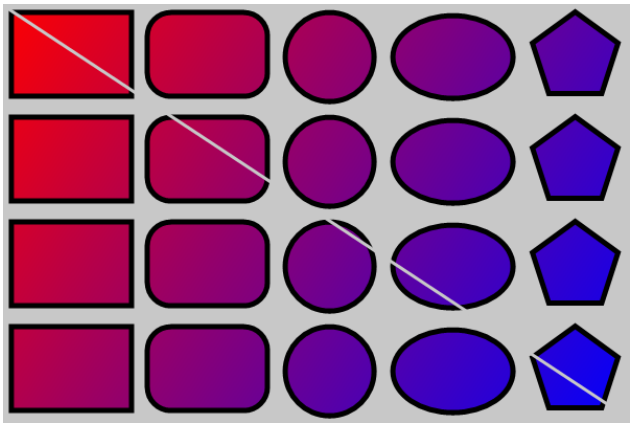


Figure 17.22: Global gradient. The continuity between figures is not lost.

Listing 17.10: Gradient matrix for the whole drawing.

```
draw_fill_linear(ctx, c, stop, 2, 0, 0, 600, 400);
draw_fill_matrix(ctx, kT2D_IDENTf);
i_draw_shapes(ctx);
```

- **Local gradient:** The vector is transferred to the origin of the figure or to a point in its near surroundings (Figure 17.23). With this, we will be able to apply the gradient locally and that only affects a specific figure. In (Listing 17.11) we have slightly varied the transformation to fix the origin in a corner and not in the center of the ellipse. This may vary depending on the desired effect.

Listing 17.11: Gradient matrix for a figure.

```
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);
```

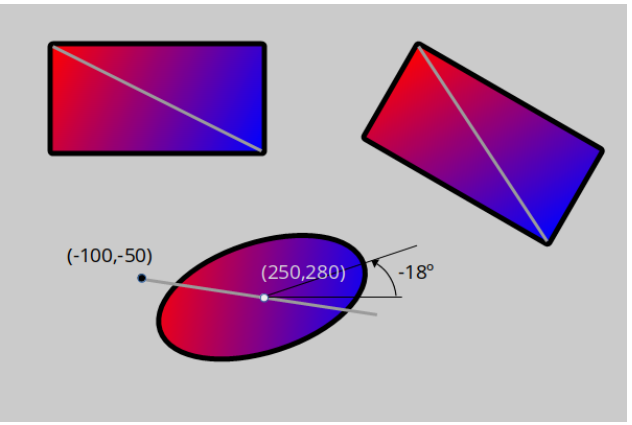


Figure 17.23: Local gradient. The origin is placed in the figure.

```
draw_fill_matrix(ctx, &t2d); // Gradient matrix
draw_ellipse(ctx, ekSKFILL, 0, 0, 100, 50);
```

17.3.5. Gradients in lines

In addition to region fill, gradients can also be applied to lines and contours (Figure 17.24) (Listing 17.12).

- Use `draw_line_fill` to draw the lines with the current fill pattern.
- Use `draw_line_color` to return to solid color.

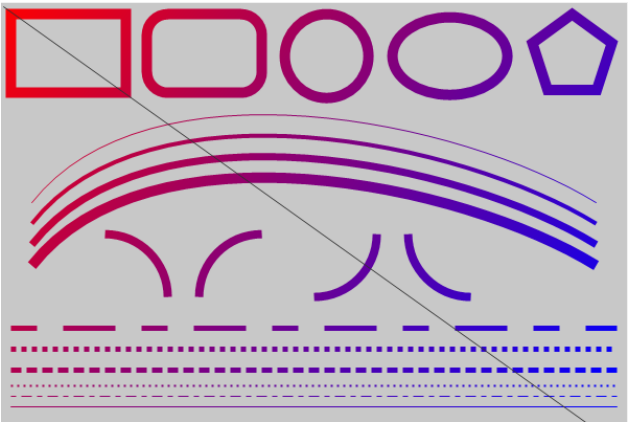


Figure 17.24: Drawing lines using gradients.

Listing 17.12: Gradients in lines.

```
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);
```

17.3.6. Gradient Limits

As we have said, the color fill will spread evenly and indefinitely along all the lines perpendicular to the vector, but... What happens outside its limits? In (Listing 17.13) (Figure 17.25) the gradient has been defined in x=[200, 400], this measure being lower than the figure to be filled:

- Use `draw_fill_wrap` to define the behavior of the gradient out of bounds.
- `ekFCLAMP` the end value is used as a constant in the outer area.
- `ekFTILE` the color pattern is repeated.
- `ekFFLIP` the pattern is repeated, but reversing the order which prevents the loss of continuity in color.

Listing 17.13: Uniform color outside the limits of the gradient (Figure 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);
```

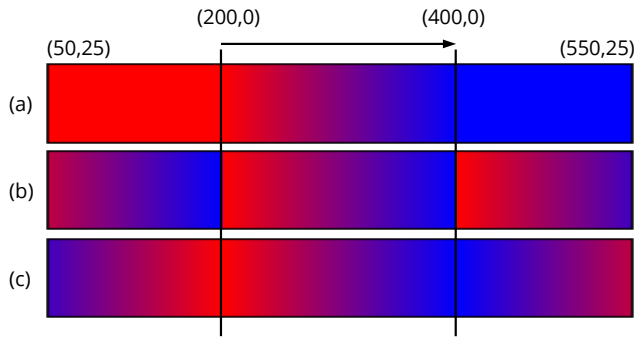


Figure 17.25: Limit Behavior: **(a)** `ekFCLAMP`, **(b)** `ekFTILE`, **(c)** `ekFFLIP`.

17.3.7. Drawing text

Text rendering is the most important part of the user interface. In the old days, small *bitmaps* were used with the image of each character, but in the early 90's vector fonts based on Bezier curves came into play. The large number of fonts, the immense set of *"Unicode"* (page 155) characters and the possibility of scaling, rotating, or layout the text in paragraphs was a great technical challenge in those years. Fortunately, all this casuistry is largely solved by the native APIs of each operating system, which allows us to provide a simplified interface to add text to our drawings..

- Use `draw_text` to draw texts in 2D contexts.
- Use `draw_text_color` to set the color of the text.

- Use `draw_font` to set the font.
- Use `draw_text_width` to set the maximum width of a block of text.
- Use `draw_text_trim` to indicate how the text will be cut.
- Use `draw_text_align` to set the alignment of a text block.
- Use `draw_text_halign` to set the internal alignment of the text.
- Use `draw_text_extents` to get the size of a block of text.

To draw single-line texts, we just have to call the function, passing a UTF8 string (Listing 17.14) (Figure 17.26). Previously, we can assign the font, color and alignment.

Listing 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);
```

Figure 17.26: Single-line texts, with alignment and transformations.

If the string to be displayed has new lines (character '\n') they will be taken into account and the text will be shown in several lines (Listing 17.15) (Figure 17.27). We can also obtain the measure in pixels of a block, useful to integrate the text with other primitives.

Listing 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);
```

Figure 17.27: Texts with a '\n' character.

If the text does not contain new lines, it will be drawn continuously expanding horizontally. This may not be the most appropriate in long paragraphs, so we can set a maximum width, forcing its drawing in several lines (Listing 17.16) (Figure 17.28).

Listing 17.16: Maximum width and internal alignment in text blocks.

```
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);
```

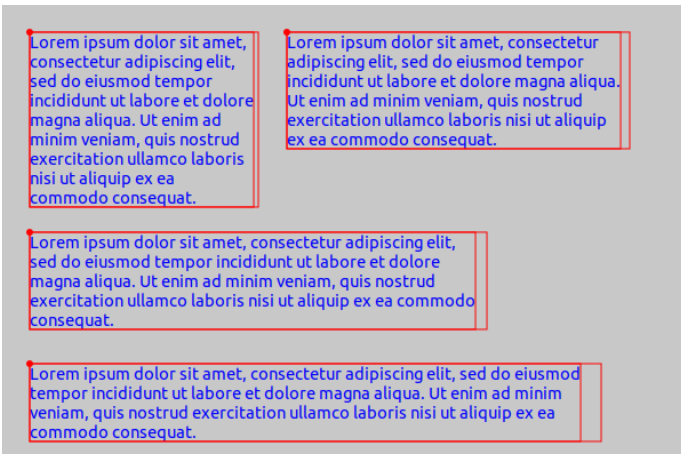


Figure 17.28: Text paragraphs with width limit. The maximum and real width obtained with `draw_text_extents` are shown.

Finally, we can use `draw_text_path` to treat the text like any other geometric region, highlighting the border or filling with gradients. In this case `draw_text_color` will have no effect and the values of `draw_fill_color`, `draw_fill_linear` and `draw_line_color` will be used (Listing 17.17) (Figure 17.29).

Listing 17.17: Text with dotted stroke and gradient fill.

```
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);
```

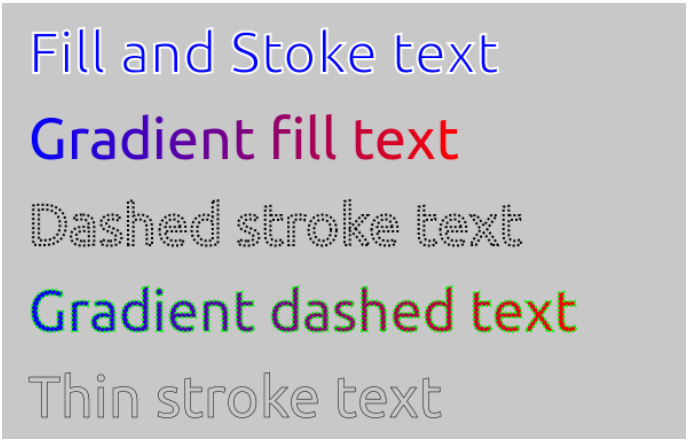


Figure 17.29: Combining fill and stroke.

`draw_text` is much faster than `draw_text_path`, so we must limit the use of the latter to what is strictly necessary.

17.3.8. Drawing images

Images generated procedurally or read from disk can be used as a drawing primitive more (Listing 17.18) (Figure 17.30). As with text or other figures, the transformation of the context will affect the geometry of the image.

- Use `draw_image` to draw an image.
- Use `draw_image_frame` to draw a sequence of an animation.
- Use `draw_image_align` to set the alignment of the image with respect to the insertion point.

Listing 17.18: Translated and rotated image drawing.

```
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);
```

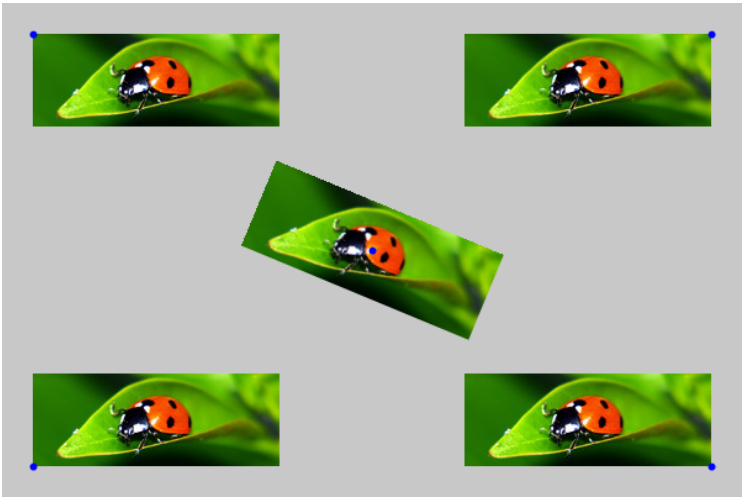


Figure 17.30: Drawing images with alignment.

17.3.9. Default parameters

Each context maintains certain state parameters. At the beginning of the drawing, either by the method `OnDraw` or after creating the context with `dctx_bitmap` the default values are those shown in (Table 17.1):

Parameter	Value	Change with
Matrix	Identity (0,0) Sup-Left corner, pixels.	`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`

Parameter	Value	Change with
FillColor	kCOLOR_BLACK	draw_fill_color
FillMatrix	Identity (0,0) Sup-Left corner, pixels.	draw_fill_matrix
Font	System default, regular size.	draw_font
Text max width	-1	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

Table 17.1: Default values in 2D contexts.

17.4. Geom2D Entities Drawing

In the previous section we have seen the basic primitives for drawing in 2D. However, *Draw2D* has specialized functions for *"Geom2D"* (page 235) objects. These new functions would be totally dispensable, since you could get the same result using draw_rect, draw_polygon, etc. They are included as a mere shortcut, in addition to offering a version of them based on *"Math templatesMath templates"* (page 53), very useful when developing generic algorithms in C++. The line and fill properties will be those that are in effect at any given time within the context, due to: draw_line_color, draw_line_width, draw_fill_color, etc..

- Use draw_v2df to draw a point.
- Use draw_seg2df to draw a segment.
- Use draw_cir2df to draw a circle.
- Use draw_box2df to draw an aligned box.
- Use draw_obb2df to draw an oriented box.
- Use draw_tri2df to draw a triangle.
- Use draw_pol2df to draw a polygon.

You can find a complete example of the use of 2D entities in *Col2DHello*[3] (Figure 17.31). In addition to drawing, this application shows other concepts related to graphics and

[3] https://nappgui.com/en/howto/col2dhello.html

geometric calculation such as:

- Create 2D objects on demand.
- *Click+Drag* interactivity.
- Collision detection.
- Calculation of areas.
- Triangulation of polygons and decomposition into convex components.
- Calculation of the optimal circle that surrounds a set of points.
- Calculation of the oriented box (`OBB2Df`) that best represents a set of points.
- Calculation of the Convex Hull.

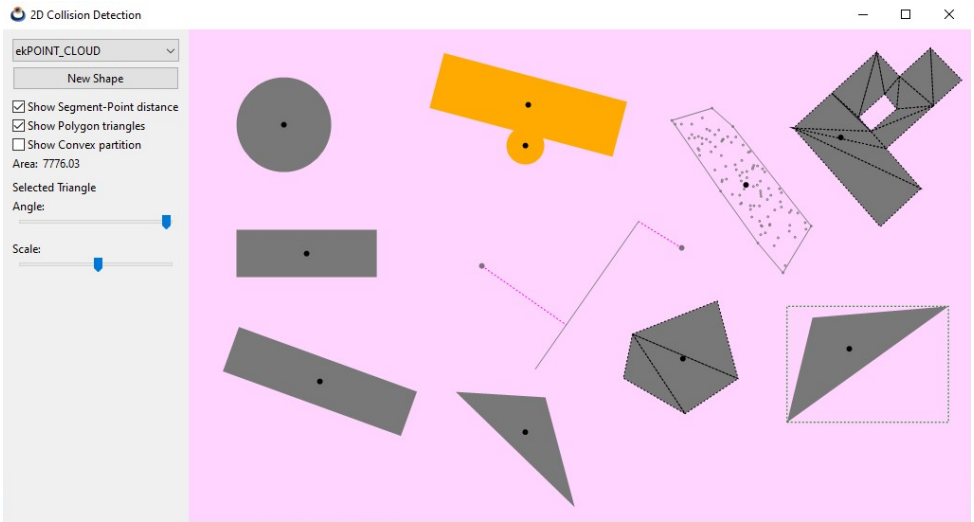


Figure 17.31: *Col2dHello* application, which illustrates how to work with 2D geometry.

17.5. Colors

The colors in Draw2D are encoded using a 32-bit integer with the four RGBA channels in Little-Endian: Red in byte 0, green in 1, blue in 2 and alpha (or transparency) in 3 (Figure 17.32). The alias `color_t` is used as an equivalent to `uint32_t`. In the particular case that byte 3 is equal to 0 (fully transparent), the first three bytes will not contain RGB information, but an index with a special color.

- Use `color_rgba` to create a color using its RGBA components.
- Use `color_get_rgba` to get the RGBA components.

- Use `color_html` to translate an string into HTML format (`"#RRGGBB"`).
- Use `kCOLOR_BLACK` and others to access predefined basic colors.

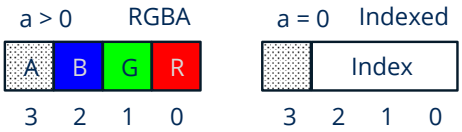


Figure 17.32: 32-bit RGBA value representation.

17.5.1. HSV space

RGB representation is based on the addition of the three primary light colors. It is the most widespread within the generation of computer images, especially when calculating shading and reflections. It is also used in TV, monitors or projectors where each pixel is obtained by combining the light of three emitters. However, it is very unintuitive for human color editing. For example, given a color in RGB, it is very difficult to increase the brightness or vary the tone (between red and orange, for example) by manipulating the triplet (r, g, b). The HSV space (*Hue*, *Saturation*, *Value*) also called HSB (*Brightness*) solves this problem, since the effect of altering this group of values will be highly predictable (Figure 17.33).

- Use `color_hsbf` to create an RGB color from its components **H**, **S**, **B**.
- Use `color_to_hsbf` to get the **H**, **S**, **B** components.

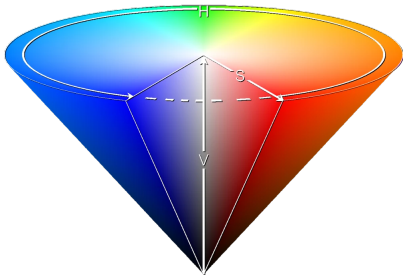


Figure 17.33: HSV space represented by an inverted cone. As V decreases, so will the number of colors available.

- **Hue:** Continuous cyclical value between 0 and 1. Where `0`=Red, `1/3`=Green, `2/3`=Blue, `1`=Red (Table 17.2).
- **Saturation:** It is equivalent to adding white paint to the base tone. When `S=1` no white is added (maximum saturation, pure color). But if `S=0` we will have a pure white, regardless of the tone.
- **Brightness:** It is equivalent to adding black paint to the `HS` combination. If `B=1` no black is added (maximum brightness). If `B=0` we will have a pure black, regardless of the hue and saturation.

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)

Table 17.2: Equivalence RGB/HSV.

Unlike RGB, HSVs are not totally independent. As we reduce the brightness, the number of colors of the same tone will decrease until we reach `B=0` where we will have pure black regardless of H and S. On the other hand, if `S=0` H will be overridden and we will have the different shades of gray as B changes from 0 (black) to 1 (white).

17.6. Palettes

A palette is nothing more than an indexed list of colors (Figure 17.34), usually related to *"Pixel Buffer"* (page 280). Its main utility is to save space in the images representation, since each pixel is encoded by an index of 1, 2, 4 or 8 bits instead of the real color where 24 or 32 bits are necessary. For this reason, it is usual to have palettes of 2, 4, 16 or 256 colors.

- Use `palette_create` to create a palette.
- Use `palette_colors` to access the elements.

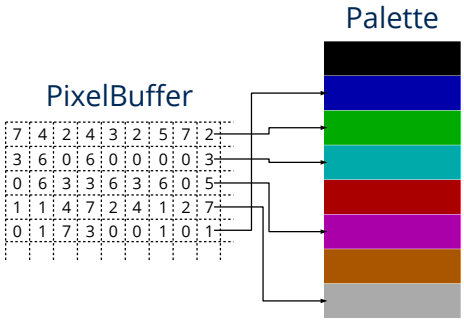


Figure 17.34: Palette associated with an indexed pixel buffer.

17.6.1. Predefined palette

We have several predefined palettes both in color (Figure 17.35) and in grays (Figure 17.36). The RGB8 palette has been created by combining 8 tones of red (3bits), 8 tones of green (3bits) and 4 tones of blue (2bits). This is so because the human eye distinguishes much less the variation of blue than the other two colors.

- Use `palette_ega4` to create a predefined palette of 16 colors.
- Use `palette_rgb8` to create a 256 color palette.
- Use `palette_gray4` and similars to create a palette in grays.
- Use `palette_binary` for a two-color palette.

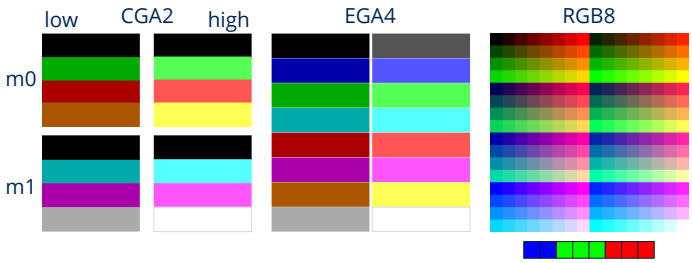


Figure 17.35: Predefined color palettes.

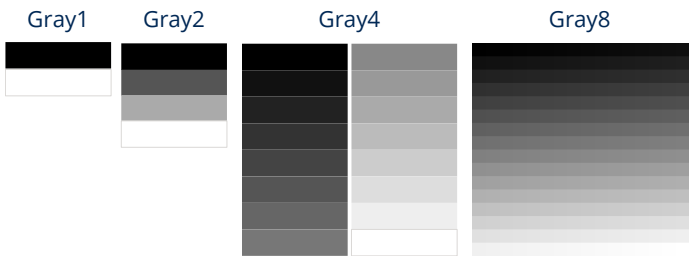


Figure 17.36: Predefined gray palettes.

17.7. Pixel Buffer

A **pixel buffer** (`Pixbuf`) is a memory area that represents a grid of color dots or pixels. They allow direct access to information but are not optimized for drawing on the screen, so we must create an `Image` object to view them. They are very efficient for procedural generation or the application of filters, since reading or writing a value does not require more than accessing its position within the buffer.

- Use `pixbuf_create` to create a new pixel buffer.
- Use `image_pixels` to get the pixels of an image.
- Use `pixbuf_width` to get the width of the grid.

- Use `pixbuf_height` to get the height of the grid.

All operations on pixel buffers are performed on the CPU. They are efficient to the extent that we directly access memory, but they cannot be compared with alternatives that use the GPU for digital image processing.

17.7.1. Pixel formats

The format refers to how the value of each pixel is encoded within the buffer (Table 17.3) (Figure 17.37).

- Use `pixbuf_format` to get the pixel format.
- Use `pixbuf_format_bpp` to get the number of bits wanted for each pixel.

Value	Description
`ekRGB24`	*True color* +16 million simultaneous, 24 bits per pixel.
`ekRGBA32`	*True color* with alpha channel (transparencies), 32 bits per pixel.
`ekGRAY8`	256 shades of gray, 8 bits per pixel.
`ekINDEX1`	Indexed, 1 bit per pixel.
`ekINDEX2`	Indexed, 2 bits per pixel.
`ekINDEX4`	Indexed, 4 bits per pixel.
`ekINDEX8`	Indexed, 8 bits per pixel.

Table 17.3: Pixel formats.

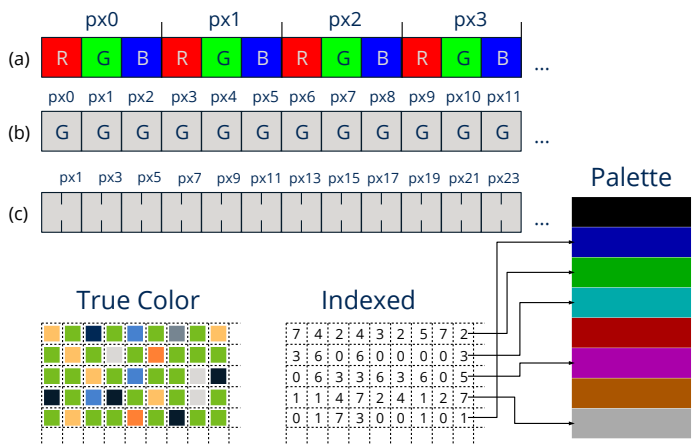


Figure 17.37: **(a)** True color, **(b)** shades of gray, **(c)** indexed.

17.7.2. Procedural images

One way to "fill" buffers is through algorithms that calculate the value of each pixel. A clear example is found in the representation of fractal sets (Figure 17.38), an area of mathematics dedicated to the study of certain dynamic systems. In *"Fractals"* (page 415) you have the complete application.

- Use `pixbuf_data` to get a pointer to the contents of the buffer.
- Use `pixbuf_set` to write the value of a pixel.
- Use `pixbuf_get` to read the value of a pixel.

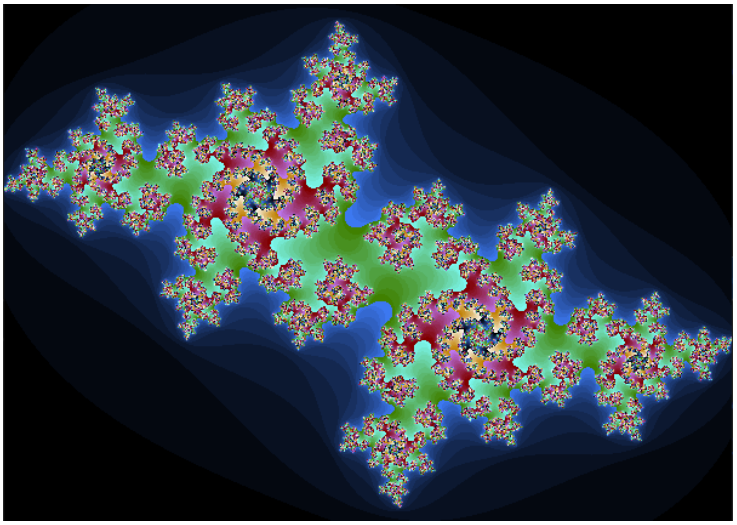


Figure 17.38: Julia set. Pixel-pixel generated image using fractal algorithms.

While `pixbuf_set` and `pixbuf_get` allow safe pixel manipulation, it may sometimes be necessary to get a little extra in terms of performance. In (Listing 17.19) we have some macros for direct access to the memory area returned by `pixbuf_data`. Use them with great care and knowing what you are doing, since they do not have error control methods, so segmentation failures are likely if they are not used correctly.

Listing 17.19: Quick macros for manipulating a buffer type `ekINDEX1` (1 bit per 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. Copy and conversion

During the digital processing of an image, we may have to chain several operations, so it will be useful to be able to make copies of the buffers or format conversions.

- Use `pixbuf_copy` to make a copy.
- Use `pixbuf_convert` to convert to another format (Table 17.4).

Source	Destiny	Observations
RGB24	RGB32	Alpha channel is added with the value 255
RGB32	RGB24	Alpha channel is removed with possible loss of information.
RGB(A)	Gray	RGB channels are weighted at a ratio of 77/255, 148/255, 30/255. Alpha channel is lost.
Gray	RGB(A)	RGB channels (gray, gray, gray) are duplicated. Alpha channel to 255.
RGB(A)	Indexed	The smallest distance between each pixel and the palette is calculated. Possible loss of information.
Indexed	RGB(A)	The palette will be used to obtain each RGBA value.
Indexed	Indexed	If the destination has a lower number of bits, $out = in \ \% \ bpp$ will be applied with possible loss of information.
Gray	Indexed	The Gray8 format will be considered indexed for all purposes.
Indexed	Gray	The Gray8 format will be considered indexed for all purposes.

Table 17.4: Conversion between formats.

17.8. Images

There is a close relationship between pixel buffers and images. Although the firsts contain "raw" color information, the latter are objects directly linked to the graphical API of each system, which allows them to be drawn in 2d contexts or viewed in a window (Figure 17.39).

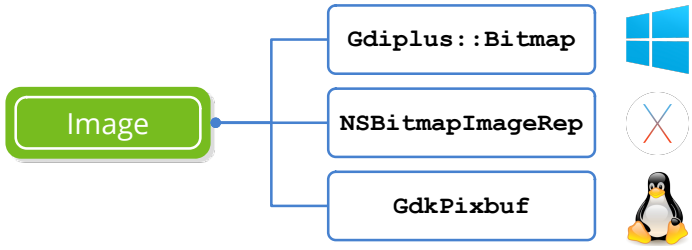


Figure 17.39: `Image` objects have a direct link to graphics APIs, while `Pixbuf` do not.

The structure of a digital image, also called *bitmap* or *raster graphics*, is the same as that of a buffer pixel. We have a discrete grid of color dots characterized by its resolution (width, height) and depth, which is the amount of bits needed to encode each pixel (Figure 17.40). *bitmap* images work best for taking snapshots of the real world, where it is practically impossible to describe the scene using geometric primitives, as we saw in *"Drawing primitives"* (page 265). On the other hand, as it is composed of discrete points, it does not behave well in the face of changes in size where it will suffer a loss of quality.

Figure 17.40: On the left an image of 64x64 pixels and 16 colors. Right 256x256 pixels and 16 million colors.

17.8.1. Load and view images

In most cases, the only thing we will need to know about images will be how to read them from disk or other data source and then display them on the screen as part of the user interface (Listing 17.20) (Figure 17.41). We consider that the images are stored in one of the standard formats: JPG, PNG, BMP or GIF.

Listing 17.20: Loading and viewing images.

```
Image *img = image_from_file("lenna.jpg", NULL);
Image *icon = image_from_resource(pack, ekCANCEL);
...
imageview_image(view, img);
button_image(button, icon);
```

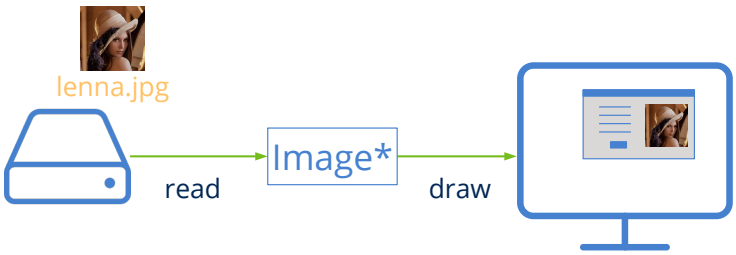


Figure 17.41: Integration of images in the user interface.

- Use `image_from_file` to load an image from disk.
- Use `image_from_data` to create an image from a memory buffer.

- Use `image_from_resource` to get a picture of a resource package.
- Use `image_read` to create an image from *"Streams"* (page 193).
- In the demo *UrlImg*[4] you have an example of how to download them from a Web server.

Once the image object is loaded in memory, we have several ways to view it:

- Use `draw_image` to draw an image in a 2d context.
- Use `imageview_image` to assign an image to a view.
- Use `button_image` to assign an image to a button.
- Use `popup_add_elem` to assign a text and icon to a drop-down list.

17.8.2. Generate images

As we saw in *"2D Contexts"* (page 257), if necessary we can create our own images from drawing commands to later display them in the interface (Figure 17.42) or save them to disk.

- Use `dctx_image` to create an image from a 2d context.

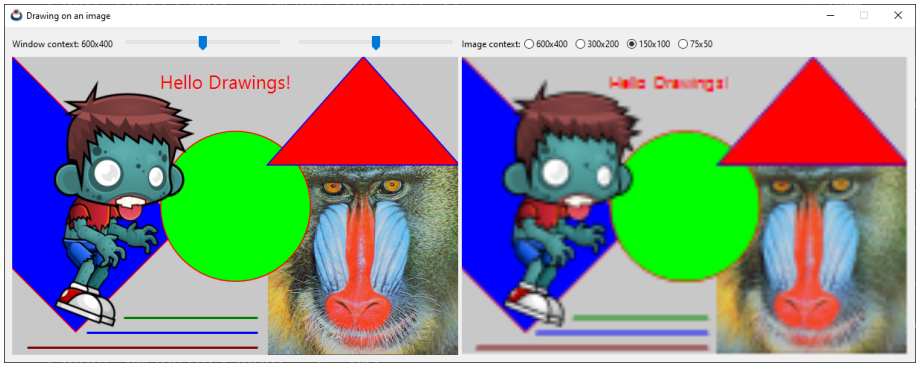


Figure 17.42: Image generated by drawing commands.

17.8.3. Pixel access

Images are **immutable objects** optimized for recurring on-screen drawing, so certain licenses are allowed, both in the internal organization of color information and in the management of possible copies. For this reason it is not possible to directly manipulate the pixels, but we must access them using a *"Pixel Buffer"* (page 280).

[4] `https://nappgui.com/en/howto/urlimg.html`

- Use `image_from_pixels` to create an image from the color information.
- Use `image_from_pixbuf` to create an image from a pixel buffer.
- Use `image_pixels` to get a buffer with the pixels of the image.
- Use `image_width` to get the width.
- Use `image_height` to get the height.
- Use `image_format` to get the pixel format.

Apple technical documentation: "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."

The **pixel buffers** allow us to optimally manipulate the content of the image. To view the result or store it in any of the supported formats, we must create a new image (Figure 17.43).

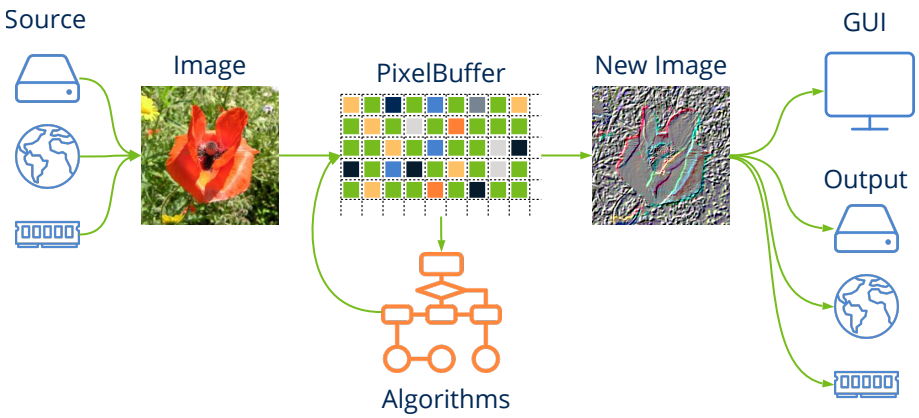


Figure 17.43: Image editing process.

17.8.4. Save images: Codecs

One of the biggest problems of digital images is the large amount of memory they need. An image of only 1024x768 pixels and 32 bits of color needs 3 megabytes of memory. It may not seem like much, but at the end of the 80s this was a great handicap since memory was very expensive and transmissions were very slow. This is why several coding (compression) systems were devised that reduced the amount of memory needed and that were consolidated with the rise of the Internet (Figure 17.44).

- Use `image_get_codec` to get the *codec* associated with the image.
- Use `image_codec` to change the *codec* associated with the image.
- Use `image_to_file` to save it to disk.
- Use `image_write` to write it in a `Stream`.

Figure 17.44: Image formats supported by NAppGUI.

Draw2D does not natively support other formats than those mentioned. If necessary, you will have to find a way to create a `Pixbuf` from the specific data of your format, in order to integrate these images into the user interface.

- **JPEG:** *Joint Photographic Experts Group* is a format with a very good compression rate based on the Fourier Transform. Ideal for capturing real-world snapshots, although it will detract some quality from the original capture (lossy compression).
- **PNG:** *Portable Network Graphics* emerged in response to legal problems with the GIF format. Supports lossless LZ77/Deflate compression and indexed pixel formats. Ideal for computer generated diagrams, graphics or images.
- **GIF:** *Graphics Interchange Format* uses the proprietary compression algorithm LZW, although the patent expired in 2003. It has survived PNG because it can include animations in a single file, something that neither of the two previous formats supports.
- **BMP:** *BitMaP*. Windows native format widely surpassed by the other three. Although it supports a special type of compression called *Run-Length encoding*, the truth is that most files are saved uncompressed. BMP files take up much more space, for this reason very little is used on the Internet and almost nothing on non-Windows machines. It is supported by almost all programs and systems because it is very simple an fast to interpret.

To be able to display on the screen, the image must be decompressed (de-encoded), a process that is performed automatically when reading the image. When saving it to disk or sending it over the network, the opposite process is performed, compressed or encoded using the algorithm associated with it (Table 17.5), but it can be changed.

Constructor	Codec
`image_from_file`	The original codec.

Constructor	Codec
image_from_data	The original codec.
image_from_resource	The original codec.
image_from_pixels	Transparencies? Yes:ekPNG No:ekJPG.
dctx_image	ekPNG.

Table 17.5: Default image codecs.

Generally, GDI+, NSImage or GdkPixbuf support for codec settings is quite limited. For example, it is not possible to generate indexed PNG files, which is very useful when reducing the size of images for the web. If the application requires more control over the export, we will have no choice but to use libpng, libjpeg or any other third-party solution.

17.9. Typography fonts

Typography fonts are graphic objects (files) that contain the characters and symbols we see on a monitor. We remember that a *"Unicode"* (page 155) string only stores the character codes *(codepoints)* without any information on how they should be drawn. The graph associated with a character is known as **glyph** and, in a font file, there are as many glyphs as *codepoints* can represent the typography. The matching between *codepoints* and their corresponding glyphs is carried out by the operating system graphic sub-system (Listing 17.21) (Figure 17.45).

Listing 17.21: Drawing a text string.

```
Font *font = font_create("Comic Sans MS" 28, 0);
draw_font(ctx, font);
draw_text(ctx, "Hello World", 200, 250);
font_destroy(&font);
```

17.9.1. Create fonts

When displaying texts on graphic interfaces it is necessary to establish a typography, otherwise the system would not know how to render it. There will always be some fount defined by default, but we can change it when customizing the appearance of our texts.

- Use font_create to create a new font.
- Use font_family to get the typeface.
- Use draw_font to set the font in 2d contexts.

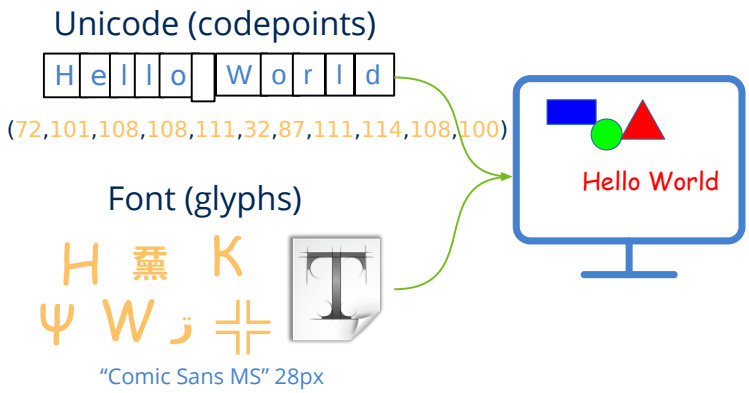


Figure 17.45: Text representation: *codepoints* + glyphs.

- Use `label_font` to change the font associated with a `Label` control.

The most representative feature of a typeface design is the family to which it belongs *(font family)* (Figure 17.46). Each computer has a series of families installed that do not have to coincide with those incorporated in another machine. This is an important fact to keep in mind since, for portability, we should not assume that a certain typeface family will be present on all machines that run the program. Sentences like:

```
Font *font = font_create("Comic Sans MS", 28, 0);
```

they will not be completely portable, since we are not sure that the *Comic Sans MS* typeface is installed in all computers. We have two alternatives to guarantee the existence of a certain font:

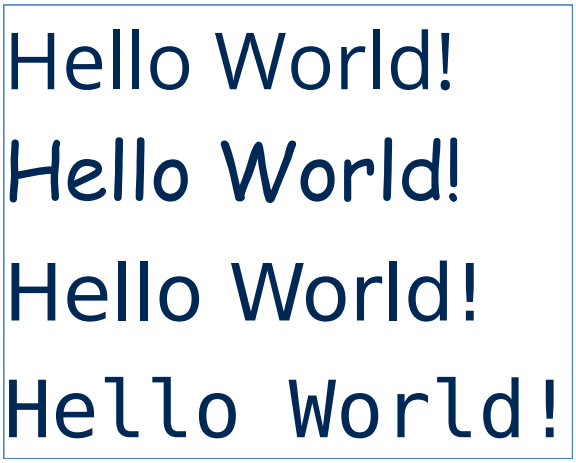


Figure 17.46: Different typographic families.

- Use `font_system` to get the default font of the operating system. It will always be available but its appearance will be different according to operating system.

- Use `font_regular_size` to get the default size for buttons and other controls.
- Use `font_installed_families` to obtain the list of families installed in the machine and choose the one that best suits our purposes.

17.9.2. System font

As we just mentioned, there is always a default font associated with the window environment and that, in a way, gives part of its personality. Using this font guarantees us the correct integration of our program in all the systems where it runs, making our code totally portable (Figure 17.47). Interface controls like `Button` or `Label` have the system font of regular size associated by default. The correspondence of *font_system* in the different systems is:

- **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.

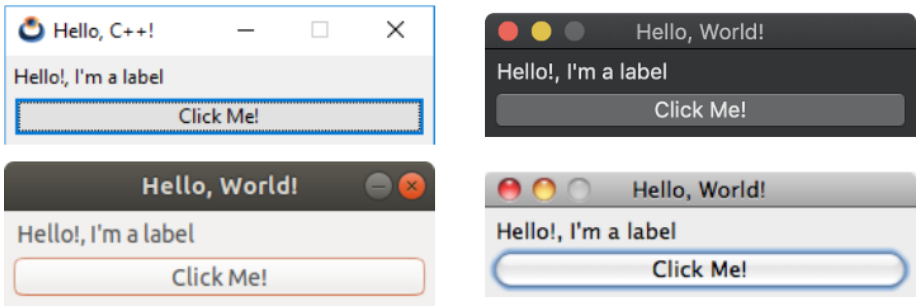


Figure 17.47: Use of the system font.

In addition to the system font we have another default **monospace** font available (Figure 17.48). These typefaces mimic old typewriters, where all characters occupy the same space. Usually used for technical documents or source code files.

- Use `font_monospace` to create a generic monospace typography.

Figure 17.48: Proportional font (variable width) and monospace (fixed width).

17.9.3. Font characteristics

In addition to the family, we can adjust the size and style of the font. The size refers to the average height (in pixels) of the characters that make up the typeface, where margins and displacements in relation to the *baseline* are not taken into account (Figure 17.49). The **total height of a line of text** is known as *cell height* and, as a general rule, it will be somewhat larger than the *char height* font size.

Figure 17.49: Character height *(char height = font size).*

We can also change the style of the text, setting its attributes through the parameter `style` combining the values of `fstyle_t` (Figure 17.50).

- `ekFBOLD`. Bold.
- `ekFITALIC`. Italic.
- `ekFUNDERLINE`. Underlined.
- `ekFSTRIKEOUT`. Strikethrough.

17.9.4. Size in points

By default, the font size is expressed in pixels, but can be changed by adding `ekFPOINTS` to the `style` parameter. This unit is related to paper fonts. Here is the DPI (*dots per inch*) concept that indicates the amount of isolated ink drops that a printing device can emit per metric inch. In typography the criterion of 72 DPI's is established, therefore, the size of a point is approximately 0.35mm. In this way it is easy to calculate the font size from the points: 12pt=4.2mm, 36pt=12.7mm or 72pt=25.4mm (1 inch). This is the unit used in **word processors**, which already work based on a print page size. The problem

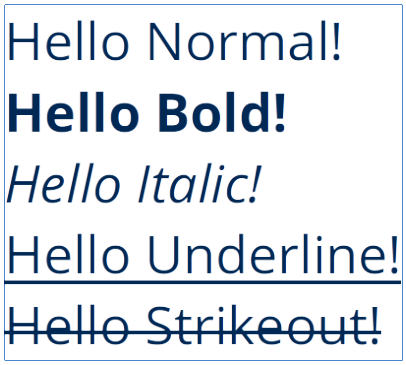


Figure 17.50: Text style.

comes when we want to represent sources expressed in points on a screen, since there is no exact correspondence between pixels and millimeters. The final pixel size depends on the resolution and physical size of the monitor. A conversion agreement between pixels and inches is required, which results in the term PPI (*pixels per inch*). Traditionally, in Windows systems 96 PPI is established while in Apple iMac it is 72 PPI. This causes the fonts expressed in points to be 33% larger in Windows (Figure 17.51). Also in the Microsoft system it is possible to configure the PPI by the user, which adds more uncertainty about the final size of the texts on the screen.

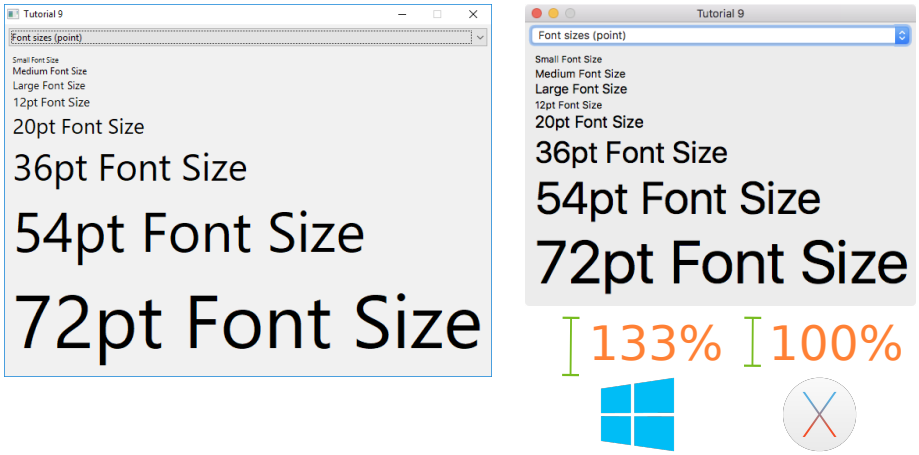


Figure 17.51: Unit `ekFPOINTS` is not advisable for screens.

17.9.5. Bitmap and Outline fonts

In the first computers typographies were created as raster graphics *Bitmap Fonts* (Figure 17.52). Each character fitted a fixed-sized cell where those pixels that made it were marked. The biggest problem is that they don't scale well. As the text on the screen grows larger, the jagged effect of the pixels becomes apparent.

Figure 17.52: Bitmap fonts.

In 1982 Adobe launched the PostScript format that included those known as *Outline Fonts* (Figure 17.53). This format contains a geometric description of each symbol based on Bezier lines and curves. In this way the pixelated effect of the bitmap is avoided, since when the character is scaled, the pixels that compose it are re-computed in a process known as **rasterization**. In the late 80's Apple launches the *TrueType* format and sells a license to Microsoft that incorporates it in Windows 3.1, opening the door of the mass market to vector sources. Today all systems work with scalable fonts, having the clearest representatives in *TrueType* and *OpenType*.

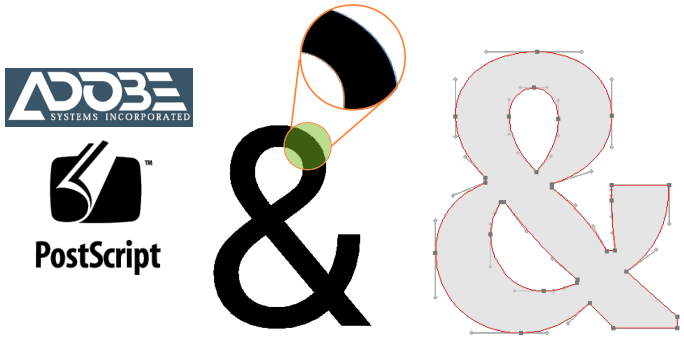


Figure 17.53: Outline fonts, on which *TrueType* and *OpenType* formats are based.

17.9.6. Unicode and glyphs

Unicode is a very large table. In version 11 (June 2018) there are 137,374 *codepoints* registered and this number grows with each new revision of the standard. If the application needs special symbols (above the BMP-*Basic Multilingual Plane)* we must make sure that the selected fonts contain glyphs for them. To see the relationship between codepoints and glyphs we can use the BabelMap application (Figure 17.54), and within it the Font Analysis option. From a Unicode block, it will show those installed sources that include glyphs for that range. In macOS we have a similar application called *Character Viewer* and in Ubuntu another one called *Character Map*.

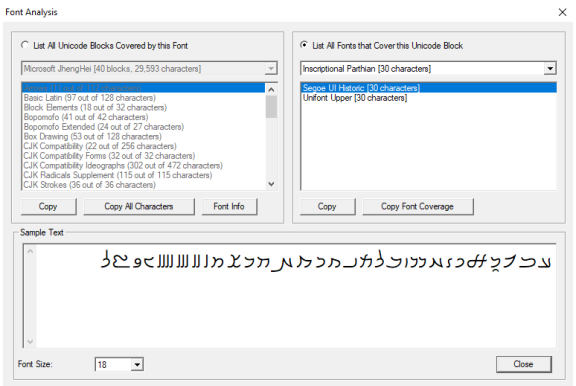


Figure 17.54: BabelMap Font Analysis gives us information about the glyphs included in each typeface.

Chapter 18

Gui library

18.1. Gui

The *Gui* library allows you to create graphical user interfaces in a simple and intuitive way. Only available for desktop applications for obvious reasons (Figure 18.1), unlike the rest of libraries that can also be used in command line applications.

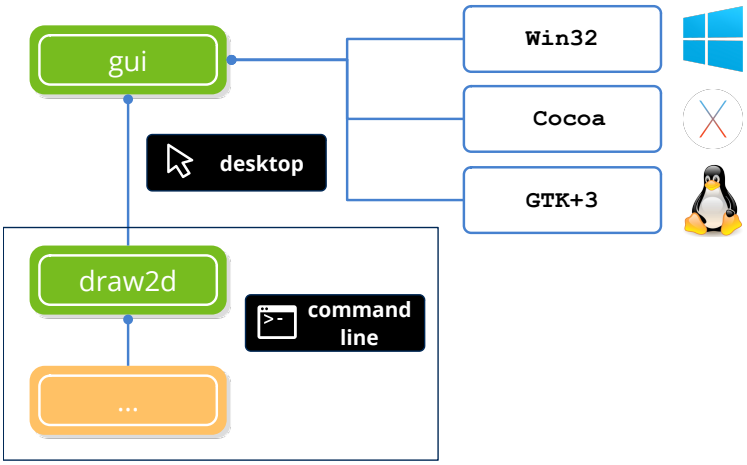


Figure 18.1: Dependencies of *Gui*. See *"NAppGUI API"* (page 145).

Like *"Draw2D"* (page 256) and *"Osbs"* (page 166) *Gui* relies on the APIs of each operating system. In addition to the advantages already mentioned in these two cases, native access to interface elements will cause our programs to be fully integrated in the desktop and according to the visual theme present in each machine (Figure 18.2).

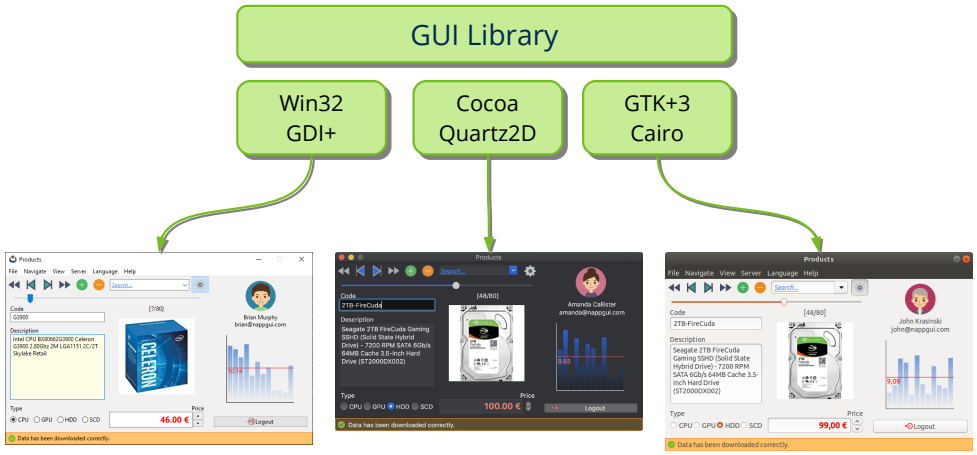


Figure 18.2: The interfaces created with *Gui* will adapt to the style of each window environment.

18.1.1. Declarative composition

The *Gui* library moves away from the concept of treating windows (or dialog boxes) as an external resource of the program. On the contrary, these are created directly from the source code avoiding layout by visual editors (Figure 18.3). We must bear in mind that window managers use different fonts and templates, so specifying specific positions and sizes for the elements will not be portable between platforms (Figure 18.4). On the contrary, in *Gui* the controls are located in a virtual grid called `Layout`, which will calculate its location and final size at runtime and depending on the platform (Figure 18.5).

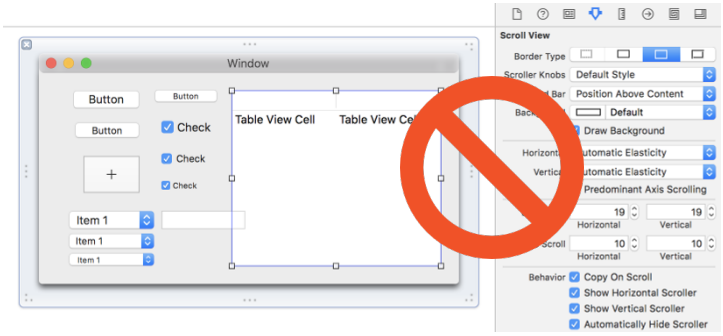


Figure 18.3: Resource editors are not good allies to create complex dynamic interfaces. Even less if we want to carry them between platforms.

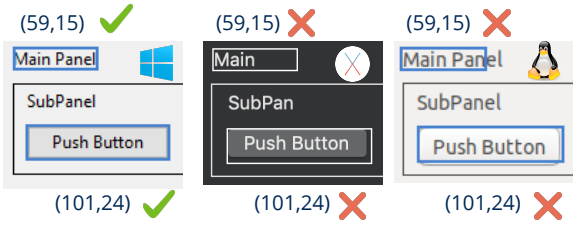


Figure 18.4: Using fixed dimensions for controls will not adapt well when migrating the program.

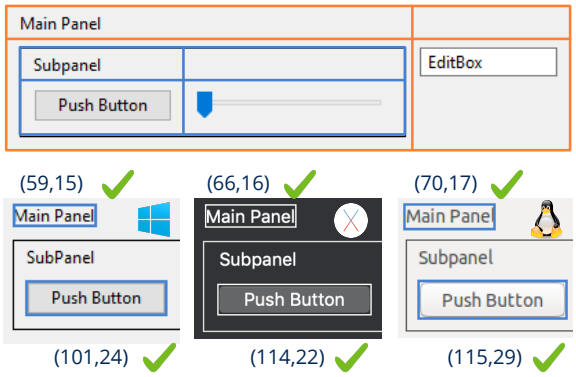


Figure 18.5: The Layout calculates the position and size of the components at runtime.

In addition, another relevant fact is that interfaces are living objects subject to constant changes. A clear example is the translations, which alter the location of the elements due to the new dimension of the text (Figure 18.6). *Gui* will adapt to these events automatically, recalculating positions to maintain a consistent layout.

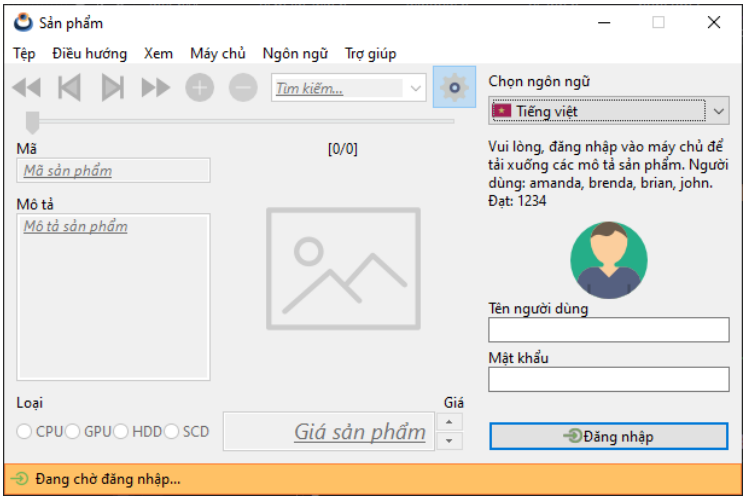


Figure 18.6: The windows automatically adapt to runtime changes.

18.1.2. Anatomy of a window.

In (Figure 18.7) we have the main parts of a window. **Controls** are the final elements with which the user interacts to enter data or launch actions. The **views** are rectangular regions of relatively large size where information is represented by text and graphics, being able to respond to keyboard or mouse events. Finally, all these elements will be grouped into **panels** and will be layout by **layouts**.

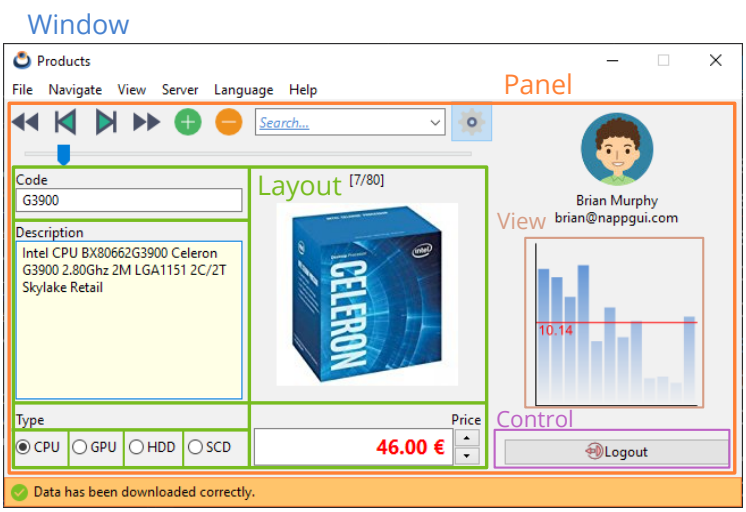


Figure 18.7: Notable parts in an interface window.

- *"GuiControl"* (page 303). Different types of controls and views.

- *"Layout"* (page 332). Virtual and invisible grid where the controls will be located.
- *"Window"* (page 347). Main window with title bar and frame.
- *"Menu"* (page 363). Drop-down list with options.
- *"MenuItem"* (page 363). Each of the menu items.

18.1.3. GUI Events

Desktop applications are event driven, which means that they are continually waiting for the user to perform some action on the interface: Press a button, drag a *slider*, write a text, etc. When this occurs, the window manager detects the event and notifies the application (Figure 18.8), which must provide an **event handler** with the code to execute. For example in (Listing 18.1) we define a handler to respond to the press of a button. Obviously, if there is no associated handler, the application will ignore the event.

- Use `event_params` to obtain the parameters associated with the event. Each type of event has its own parameters. See (Table 18.1).
- Use `event_result` to write the response to the event. Very few events require sending a response.

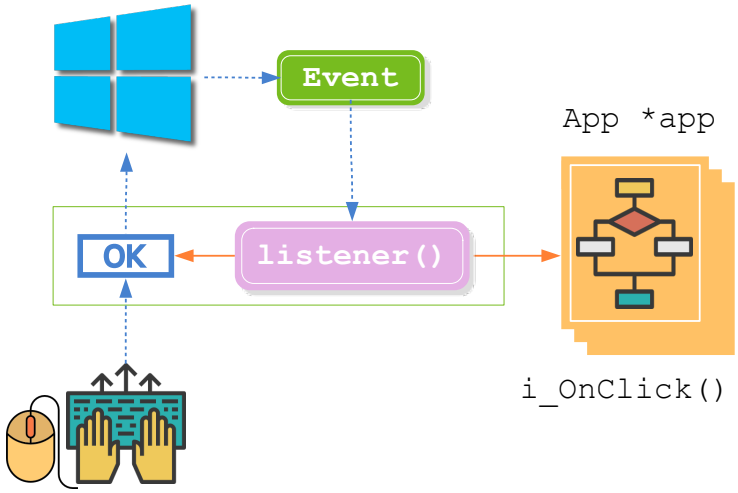


Figure 18.8: Notification of an event through the handler.

Listing 18.1: Assign a handler for the push of a button.

```
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));
```

Event	Handler	Parameters	Response
Click in label	`label_OnClick`	`EvText`	-
Click on button	`button_OnClick`	`EvButton`	-
Selection in PopUp	`popup_OnSelect`	`EvButton`	-
Selection in ListBox	`listbox_OnSelect`	`EvButton`	-
Press key on Edit	`edit_OnFilter`	`EvText`	`EvTextFilter`
End of edit in Edit	`edit_OnChange`	`EvText`	`bool_t`
Edit has received or lost keyboard focus	`edit_OnFocus`	`bool_t`	-
Key press on Combo	`combo_OnFilter`	`EvText`	`EvTextFilter`
End of editing in Combo	`combo_OnChange`	`EvText`	`bool_t`
Slider movement	`slider_OnMoved`	`EvSlider`	-
Click on UpDown	`updown_OnClick`	`EvButton`	-
Draw the contents of a view	`view_OnDraw`	`EvDraw`	-
The size of a view has changed	`view_OnSize`	`EvSize`	-
The mouse enters the area of a view	`view_OnEnter`	`EvMouse`	-
The mouse leaves the area of a view	`view_OnExit`	-	-
The mouse moves over a view	`view_OnMove`	`EvMouse`	-
A mouse button was pressed	`view_OnDown`	`EvMouse`	-
A mouse button has been released	`view_OnUp`	`EvMouse`	-
Click on a view	`view_OnClick`	`EvMouse`	-
Dragging on a view	`view_OnDrag`	`EvMouse`	-
Mouse wheel on a view	`view_OnWheel`	`EvWheel`	-
Press key on a view	`view_OnKeyDown`	`EvKey`	-
Release key on a view	`view_OnKeyUp`	`EvKey`	-
View has received or lost keyboard focus	`view_OnFocus`	`bool_t`	-

Event	Handler	Parameters	Response
View resing keyboard focus	view_OnResignFocus	GuiControl	bool_t
View accepts keyboard focus	view_OnAcceptFocus	-	bool_t
The scroll bars are being manipulated.	view_OnScroll	EvScroll	real32_t
Keystroke in TextView	textview_OnFilter	EvText	EvTextFilter
TextView has received or lost keyboard focus	textview_OnFocus	bool_t	-
Close a window	window_OnClose	EvWinClose	bool_t
Window moving around the desk	window_OnMoved	EvPos	-
Window is re-dimensioning	window_OnResize	EvSize	-
Click on an item menu	menuitem_OnClick	EvMenu	-
Color change	comwin_color	color_t	-

Table 18.1: List of all interface events.

18.2. GuiControl

GuiControl is the virtual base class where common functionality is defined for all the controls and views that we use to create the user interfaces (Figure 18.9).

- Use guicontrol_button and others to do dynamic casting between types.
- Use GuiControlPtr to convert derived types.
- Use guicontrol_enable to enable or disable a control.
- Use guicontrol_visible to show or hide a control.
- *"Label"* (page 304). Small blocks of descriptive text.
- *"Button"* (page 306). Push buttons, check boxes or radio.
- *"PopUp"* (page 308). Button with drop-down list.
- *"Edit"* (page 309). Text edit box.
- *"Combo"* (page 312). Edit box with drop-down list.
- *"ListBox"* (page 312). List box.
- *"UpDown"* (page 314). Increment and decrement buttons.

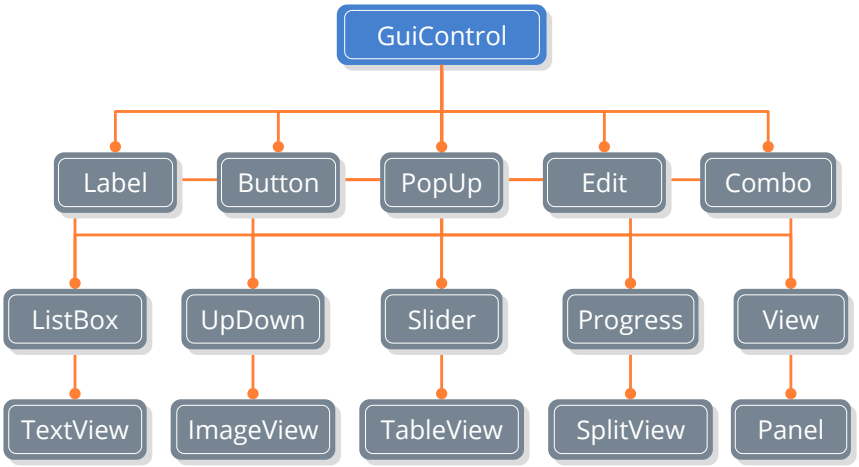


Figure 18.9: Controls and views.

- *"Slider"* (page 314). Sliding bar.
- *"Progress"* (page 315). Progress bar.
- *"View"* (page 315). Generic view where you can freely draw.
- *"TextView"* (page 321). View to show and edit texts in multiple formats.
- *"ImageView"* (page 322). View to display images.
- *"TableView"* (page 323). Table view to display information in rows and columns.
- *"SplitView"* (page 329). View divided into two resizable parts.
- *"Panel"* (page 341). Sub-window within the main one with its own controls.

18.3. Label

Label controls are used to insert small blocks of text into windows and forms. They are of uniform format, that is, the font and color attributes will be applied to the entire text. In most cases the content will be limited to a single line, although it is possible to show blocks that extend in several lines. The control size will be adjusted to the text it contains (Figure 18.10). In *"Hello Label!Hello Label!"* (page 493) you have an example of use.

- Use `label_create` to create a text control.
- Use `label_multiline` to create a multi-line control.
- Use `label_align` to set the internal alignment of the text.

- Use `label_font` to set the font.

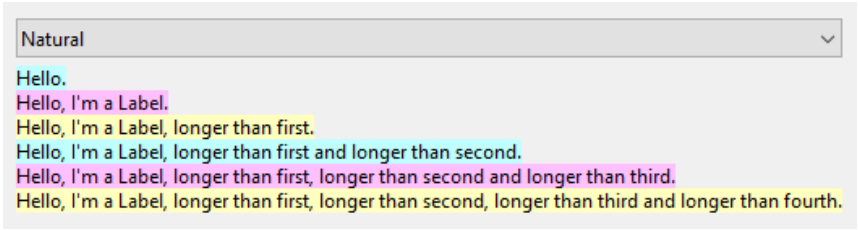


Figure 18.10: Label controls.

In the case that the column of `Layout` has a width smaller than the text, some dots (ellipse) will be displayed at the clipping point (Figure 18.11), except in multi-line labels, which will expand vertically to accommodate all text (Figure 18.12).

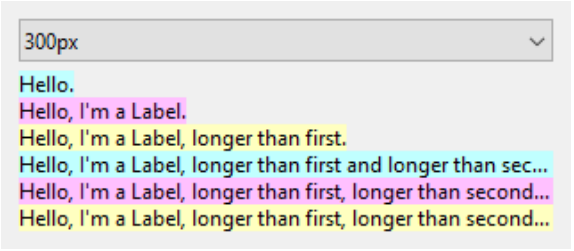


Figure 18.11: Text adjustment by reducing the width of the control.

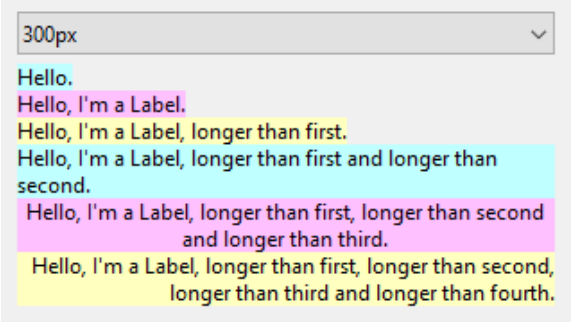


Figure 18.12: Multi-line labels will expand vertically to accommodate all text.

In (Figure 18.13) we have an example of the use of *Label* in forms. If necessary, we can make the texts sensitive to the mouse by varying their style and colors (Figure 18.14).

- Use `label_style_over` to change the font style.
- Use `label_color_over` to change text color.
- Use `label_bgcolor_over` to change background color.
- Use `label_OnClick` to respond to a click on the text.

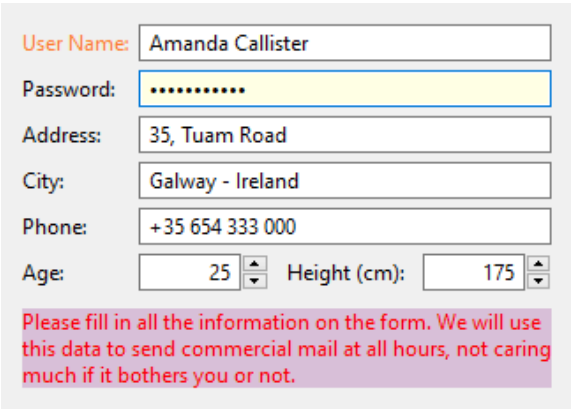


Figure 18.13: Using simple and multiline *Label* in forms.

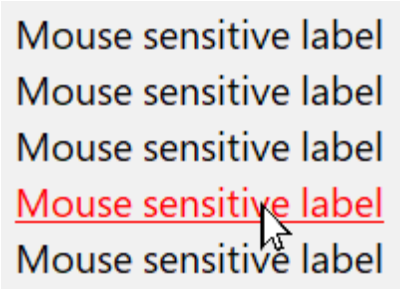


Figure 18.14: *Label* controls sensitive to the mouse.

18.4. Button

The buttons are another classic element in graphic interfaces, where we distinguish four types: the push button, checkbox, radiobutton and flat button typical of toolbars (Figure 18.15) . In *"Hello Button!Hello Button!"* (page 498) you have an example of use.

- Use `button_push`to create a push button.
- Use `button_check`to create a check box.
- Use `button_check3` to create a box with three states.
- Use `button_radio` to create a radio button.
- Use `button_flat`to create a flat button.
- Use `button_flatgle`to create a flat button with status.
- Use `button_text` to assign text.
- Use `button_OnClick` to respond to keystrokes.

In addition to capturing the event and notifying the application, the *checkbox* and *flatgle* maintain a state (pressed/check or released/uncheck).

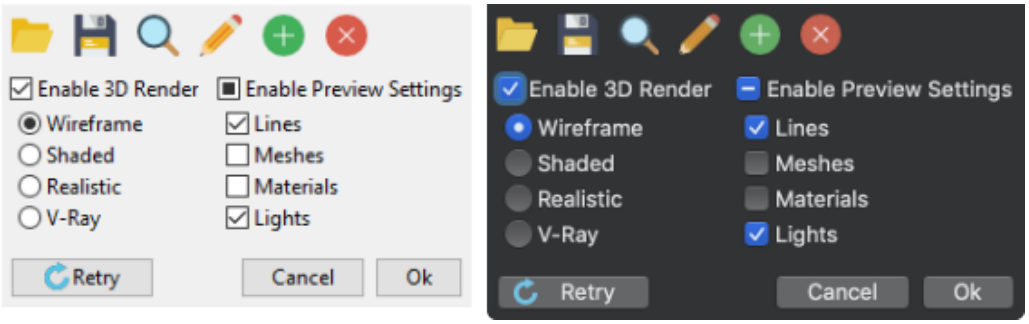


Figure 18.15: Buttons on different platforms.

- Use `button_state`to set the button status.
- Use `button_get_state` to get the status of the button.

18.4.1. RadioGroup

Special mention is required of the radio buttons, which only make sense when they appear in a group, since they are used to select a single option within a set. Groups are formed at the `Layout` level, that is, all *radiobuttons* of the same layout will be considered from the same group, where only one of them can be selected. If we need several sub-groups, we must create several sub-layout, as shown (Figure 18.16) (Listing 18.2). When capturing the event, the field `index`from `EvButton` will indicate the index of the button that has been pressed.

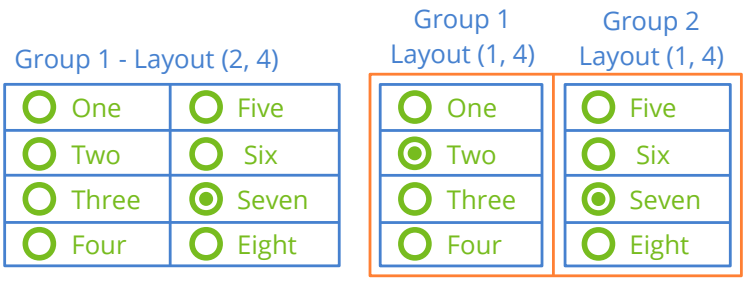


Figure 18.16: Radio groups linked to different layouts.

Listing 18.2: Radio button groups.

```
Button *button1 = button_radio();
Button *button2 = button_radio();
Button *button3 = button_radio();
Button *button4 = button_radio();
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);
```

18.5. PopUp

PopUps are buttons that have a drop-down menu associated with them (Figure 18.17). Apparently they look like *pushbuttons* that when pressed show a list of options. In *"Hello PopUp and Combo!Hello PopUp and Combo!"* (page 501) you have an example of use.

- Use `popup_create` to create a popup.
- Use `popup_add_elem` to add an item to the list.
- Use `popup_OnSelect` to respond to the selection.

Figure 18.17: PopUps on Windows, macOS and Linux.

18.6. Edit

EditBox are small text boxes with editing capabilities. Like the `Label` they are of uniform format: The typeface and colors will affect the entire text (Figure 18.18). They are usually used to edit fields in forms, normally restricted to a single line, although they can also be extended to several of them. To edit texts with multiple attributes use `TextView`. In Hello Edit and UpDown![1] you have an example of use.

- Use `edit_create`to create an edit box.
- Use `edit_multiline` to create a multi-line editing box.
- Use `edit_passmode` to hide the text of the control.
- Use `edit_phtext` to set a *placeholder*.
- Use `edit_autoselect` to automatically select all text.

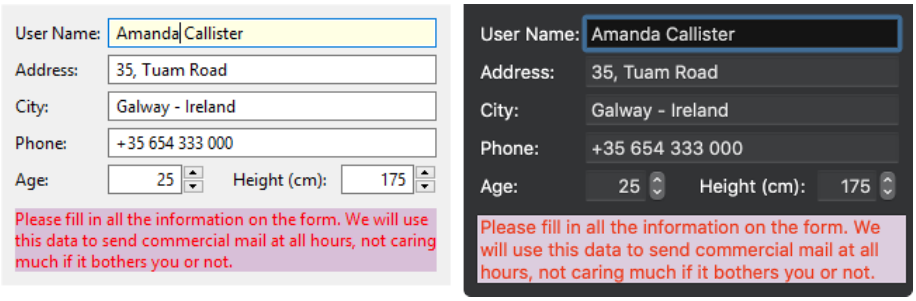


Figure 18.18: Edition boxes on different platforms.

18.6.1. Filter texts

- Use `edit_OnChange` to validate the final text.

[1] `HelloEditandUpDown!`

- Use `edit_OnFilter` to detect and correct each user keystroke.

Depending on the value we are editing, it may be necessary to validate the entered text. We can do this when finishing editing or while we are writing. For the first case we will use the `edit_OnChange` (Listing 18.6.1) event that will call the handler just before the control loses keyboard focus (Figure 18.19). If the text is invalid, the handler must return `FALSE`, thus preventing the focus from changing to the next control, remaining in the editbox and forcing the user to correct it.

```
static void i_OnChange(UserData *data, Event *e)
{
    const EvText *p = event_params(e, EvText);
    if (is_valid_text(data, p->text) == FALSE)
    {
        // Force the focus remain in editbox
        bool_t *r = event_result(e, bool_t);
        *r = FALSE;
    }
}
...
edit_OnChange(edit, listener(NULL, i_OnChange, void));
```

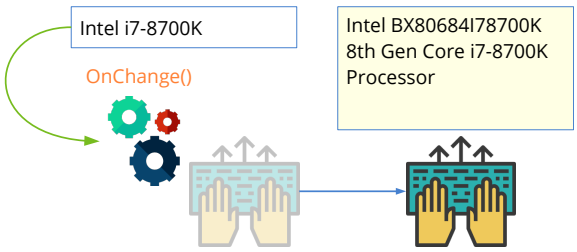


Figure 18.19: The *OnChange* event is called just before the control loses focus.

It will not be possible to move keyboard focus to another control while the text is invalid.

In case we want to implement more elaborate filters that correct the text while it is being written, we will use the `edit_OnFilter` event. We will receive, through the `EvText` structure, a copy of the current text, the position of the cursor (caret) and the number of characters added or deleted. From here, the filter will be able to validate the text, setting the `apply` field of `EvTextFilter` to `FALSE`. If the new characters are not appropriate, we will return the new text and cursor position in the `text` and `cpos` fields of `EvTextFilter`, putting `apply` to `TRUE`. For example, in (Listing 18.3) we have a simple filter that only allows numeric characters (Figure 18.20).

Listing 18.3: Filter that only allows numeric characters.

```
static void OnFilter(void *noused, Event *e)
{
    const EvText *params = event_params(e, EvText);
```

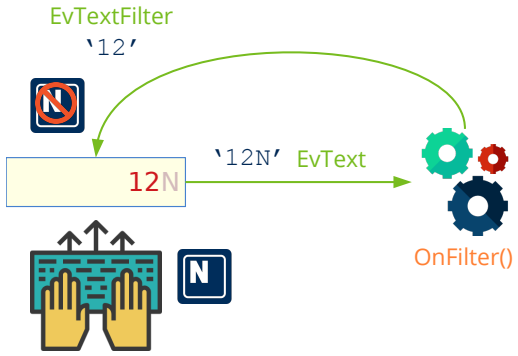


Figure 18.20: The *OnFilter* event is called after each key press.

```
    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(edit, listener(NULL, i_OnFilter, void));
```

18.6.2. Clipboard operations

Being native components, the `Edit` controls support typical clipboard operations: Copy, Paste, Cut, etc, as well as their keyboard shortcuts. However, it can be useful to access these operations from the program code, allowing, for example, the text selected in the control to be copied to the clipboard.

- Use `edit_copy` to copy the selected text to the clipboard.
- Use `edit_cut` to cut the selected text, copying it to the clipboard.
- Use `edit_paste` to paste the clipboard text at the caret position.

18.7. Combo

ComboBox are text editing boxes with drop-down list (Figure 18.21). Therefore, they will work in the same way as `Edit` controls on which methods for the management of the list are added. In *"Hello PopUp and Combo!Hello PopUp and Combo!"* (page 501) you have an example of use.

- Use `combo_create` to create a combo.
- Use `combo_text` to set edit text.
- Use `combo_color` to set the text color.
- Use `combo_bgcolor` to set the background color.
- Use `combo_add_elem` to add an item to the list.

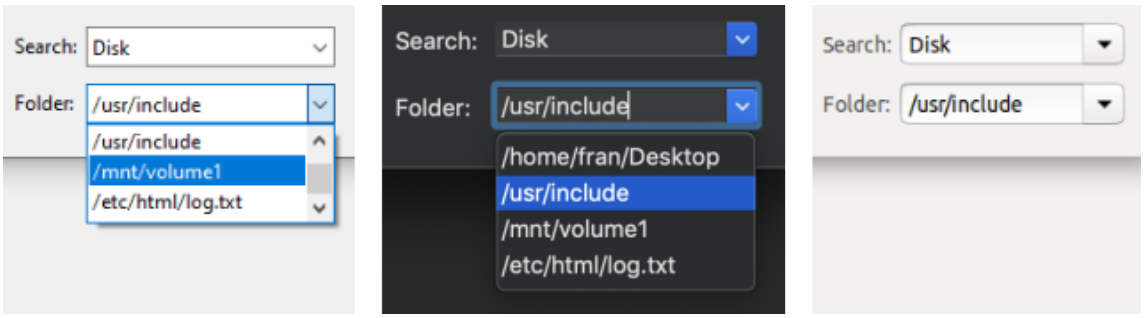


Figure 18.21: Combos on Windows, macOS and Linux.

18.8. ListBox

The **ListBox** are controls that display a series of elements as a list (Figure 18.22), (Figure 18.23), (Figure 18.24). Depending on how it is configured, we can select one or more elements or view *checkboxes* to check them. The control enables scroll bars when necessary and allows keyboard navigation. In *"Hello ListBox!Hello ListBox!"* (page 507) you have an example of use.

- Use `listbox_create` to create a list control.
- Use `listbox_add_elem` to add an element.
- Use `listbox_multisel` to enable the multiple selection.
- Use `listbox_checkbox` to enable the checkboxes.
- Use `listbox_OnSelect` to respond to the selection.

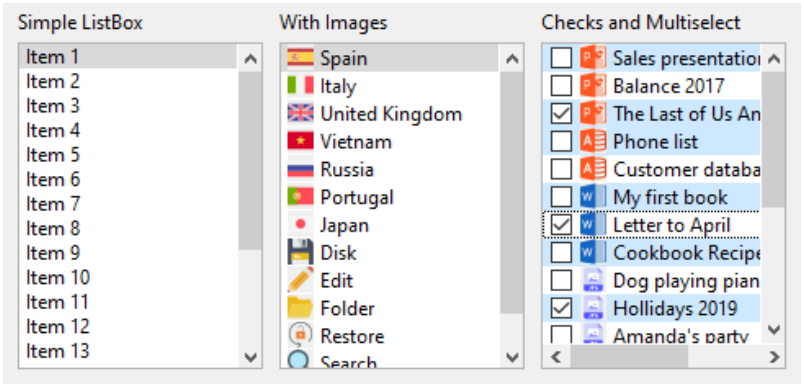


Figure 18.22: ListBox controls in Windows.

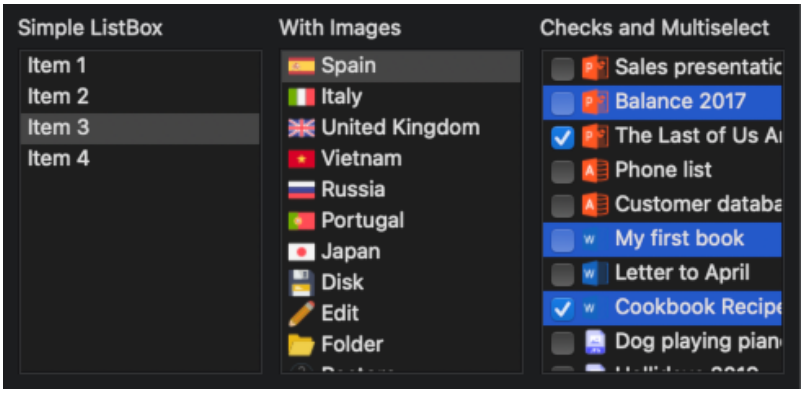


Figure 18.23: ListBox controls in macOS.

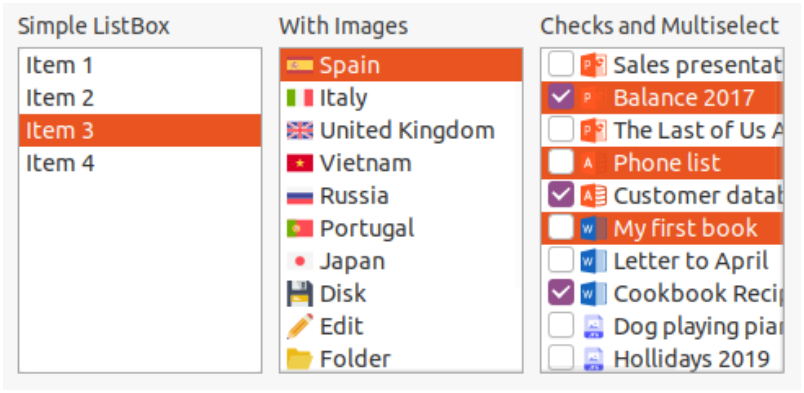


Figure 18.24: ListBox controls in Linux.

18.9. UpDown

UpDown are two-part horizontally divided button controls (Figure 18.25). Each part has a small arrow printed and is normally used to make discrete increases in numerical values associated with controls `Edit`.

- Use `updown_create` to create an updown button.
- Use `updown_OnClick` to respond to keystrokes.

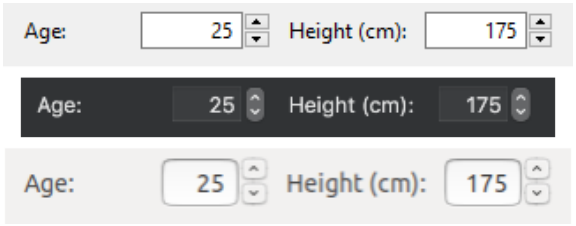


Figure 18.25: UpDown on Windows, macOS and Linux.

18.10. Slider

Sliders are normally used to edit continuous and bounded numerical values (Figure 18.26). As the control moves, *OnMoved* events occur that return a value between 0 and 1. In *"Hello Slider and Progress!Hello Slider and Progress!"* (page 509) you have an example of use.

- Use `slider_create` to create a horizontal slider.
- Use `slider_vertical` to create a vertical slider.
- Use `slider_OnMoved` to respond to scrolling.

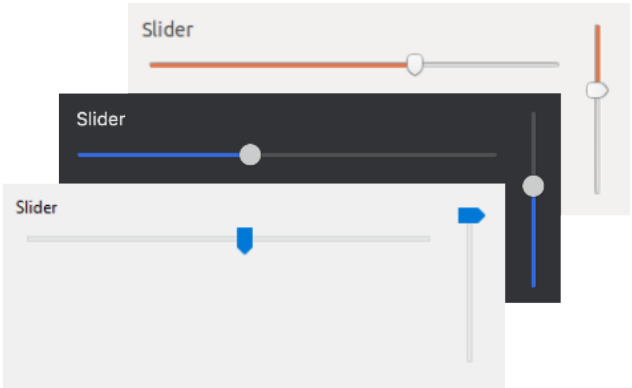


Figure 18.26: Sliders on Windows, macOS and Linux.

18.11. Progress

Progress bars are passive controls that show the remaining time to complete a certain task (Figure 18.27). As time passes we must update the control. The undefined state will show an animation without indicating status, which will be useful when we cannot determine the required time.

- Use `progress_create` to create a progress bar.
- Use `progress_undefined` to set the bar as undefined.
- Use `progress_value` to update the progress of the task.

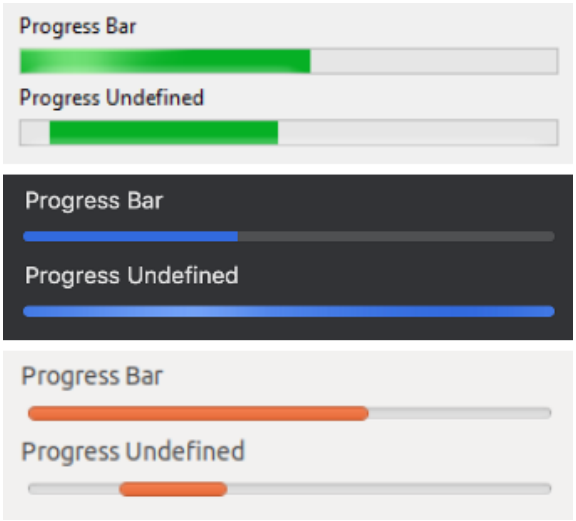


Figure 18.27: ProgressBar on Windows, macOS and Linux.

18.12. View

The **View** controls or custom views (Figure 18.28) are blank areas within the window that allow us to implement our own components. We will have total freedom to draw and capture the mouse or keyboard events that allow us to interact with it.

- Use `view_create` to create a view.
- Use `view_data` to set a data object.
- Use `view_get_data` to get this object.
- Use `view_size` to set the default size. See *"Natural sizingNatural sizing"* (page 334).

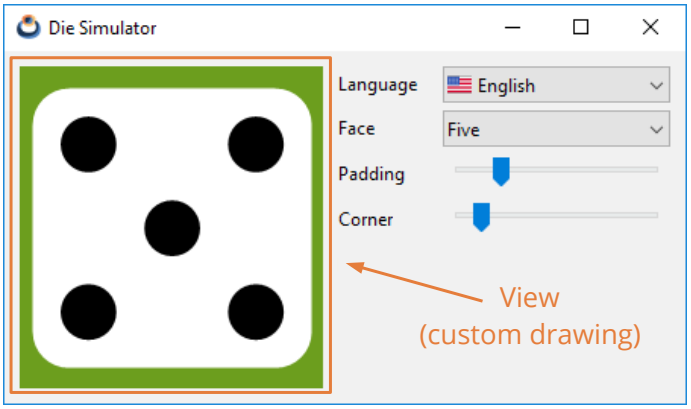


Figure 18.28: Custom view control.

18.12.1. Draw in views

The contents of the drawing area will need to be refreshed on certain occasions. Either because the operating system must update a previously overlapping part, or because the drawing itself has changed (animations, user actions, etc.). When the time comes, the window manager will launch an `OnDraw` event that the application must capture to implement the drawing commands that allow the content to be recreated.

- Use `view_OnDraw` to set the drawing handler.
- Use `view_update` to force an area update.

The `OnDraw` event handler will receive a drawing context, on which the different *"Drawing primitives"* (page 265) (Listing 18.4) can be applied.

Listing 18.4: Basic drawing in custom views.

```
static void i_OnDraw(UserData *data, Event *e)
{
    const EvDraw *p = event_params(e, EvDraw);
    draw_clear(p->ctx, kCOLOR_RED);
    draw_line_width(p->ctx, 10.f);,
    draw_line_color(p->ctx, kCOLOR_GREEN);
    draw_rect(p->ctx, ekSTROKE, 0, 0, p->width, p->height);
}
...
view_OnDraw(view, listener(data, i_OnDraw, UserData));
```

In *"Die"* (page 391) you have a simple example application that implements drawing custom views. It represents the figure of a die, allowing us to edit certain parameters of the drawing. This interaction will launch a series of events that will require the redrawing of our figure. The entire cycle can be summarized in these steps (Figure 18.29):

- Some event occurs that requires updating the content of the view.

- The application calls the `view_update` method to notify that the view must be updated.
- At the appropriate moment, the system will send an `OnDraw` event with a `DCtx` context ready to draw.

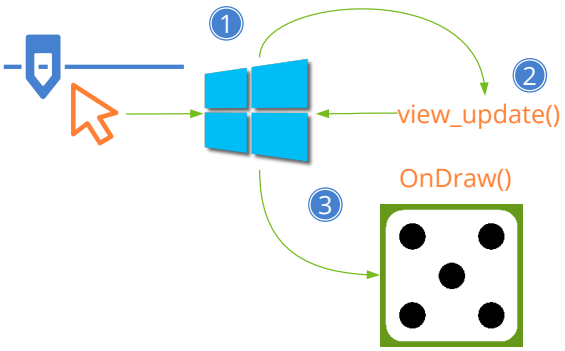


Figure 18.29: Refresh cycle of a custom view.

The operating system can launch `OnDraw` events at any time without previously calling `view_update`.

18.12.2. Scrolling views

It is possible that the "scene" to be rendered is much larger than the control itself, so it will show only a small fragment of it (Figure 18.30). In these cases we will say that the view is a *viewport* of the scene. We can manage it in two ways:

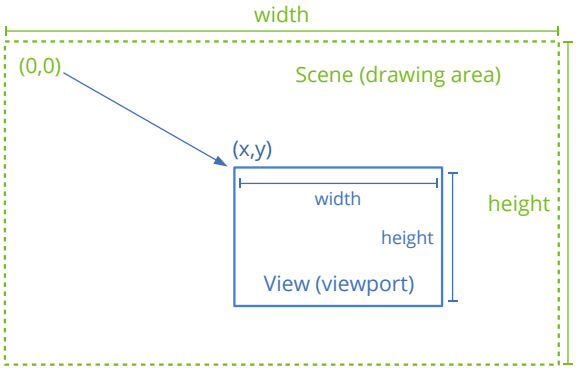


Figure 18.30: Scene and view (viewport).

- Use `draw_matrixf` at the beginning of `OnDraw` to indicate the transformation that integrates the displacement, zoom and possible rotation of the viewport with respect to the scene. All of this must be managed by the application and we do not have to do anything special, except call `view_update` every time it is necessary to refresh.
- Use scroll bars that allow the user to move freely through the content. In this case,

managing the view is a bit more complicated. This is what we must take into account:

- Use `view_scroll` or o `view_custom` to create the view.
- Use `view_content_size` to indicate the measurements of the scene, so that the bars are sized correctly.
- Use `view_scroll_x`, `view_scroll_y` if we want to move the scroll bars from the code.
- Use `view_viewport` to get the position and dimensions of the visible area.
- Use `view_OnScroll` to detect when the user manipulates the scroll bars.

Something important to keep in mind is to avoid drawing non-visible elements, especially in very large scenes or with a multitude of objects. The operating system will send successive `OnDraw()` events as the user manipulates the scrollbars, indicating the *viewport* parameters in the `EvDraw` structure. In *"Scroll drawings"* (page 623) you have an example application that shows how to correctly manage this type of cases.

It is possible that the dimensions of the viewport received in `OnDraw` are somewhat larger than the size of the control. This is because certain window managers force you to draw in certain non-visible areas close to the edges, in order to avoid flickering when scrolling very quickly.

18.12.3. Using the mouse

In order to interact with the control, it is necessary to define handlers for the different mouse events (Listing 18.5), (Figure 18.31). The operating system will notify the user's actions so that the application can launch the relevant actions. It is not necessary to use all of them, only the essential ones in each case.

Listing 18.5: Response to mouse events.

```
static void i_OnMove(UserData *data, Event *e)
{
    const EvMouse *p = event_params(e, Event);
    do_something_onmouse_moved(p->x, p->y);
}
...
view_OnMove(view, listener(data, i_OnMove, UserData));
```

- Use `view_OnEnter` to know when the cursor enters the view.
- Use `view_OnExit` to know when the cursor leaves the view.
- Use `view_OnMove` to know when the cursor is moving through the view.

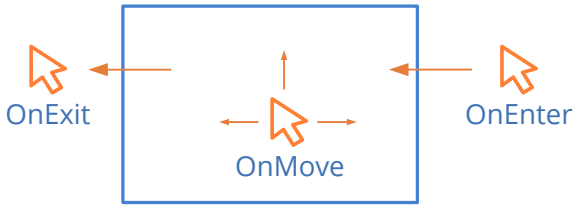


Figure 18.31: View position events.

- Use `view_OnDown` to know when a button is pressed within the view.
- Use `view_OnUp` to know when a button is released inside the view.
- Use `view_OnClick` to identify a click (Fast Up + Down).
- Use `view_OnDrag` to move the cursor with a pressed button.
- Use `view_OnWheel` to use the mouse wheel.

If the view uses scroll bars, the cursor (x,y) position passed to `EvMouse` in each event, refers to the global coordinates of the scene, taking into account the displacement. In views without scroll bars, they are the control local coordinates. The local coordinates of the viewport are in (lx,ly).

18.12.4. Using the keyboard

When a view receives *"Keyboard focusKeyboard focus"* (page 351), all keystrokes will be directed to it, so we must implement the appropriate handlers.

- Use `view_OnKeyDown` to detect when a key is pressed.
- Uses `view_OnKeyUp` to detect when a key is released.
- Use `view_OnFocus` to notify the application whenever the view receives (or loses) keyboard focus. In (Figure 18.32), the view changes the color of the active cell when it has focus.
- Use `view_OnResignFocus` to prevent the view from losing keyboard focus.
- Use `view_OnAcceptFocus` to prevent the view from getting keyboard focus.

If a view does not need to use the keyboard, make sure it cannot receive focus when you press `[TAB]` "TabstopsTabstops" (page 338). It also implements `view_OnAcceptFocus` to prevent it from getting focus when you click on it.

In the `KeyDown` and `KeyUp` events a `vkey_t` will be received with the value of the pressed key. In (Figure 18.33) and (Figure 18.34) the correspondence of these codes is shown. In *"Synchronous applicationsSynchronous applications"* (page 373) we may need

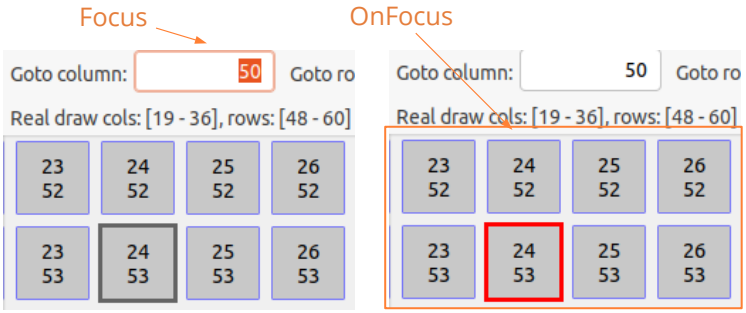


Figure 18.32: View without keyboard focus (left) and with it (right).

to know if a key is pressed or not during the update cycle (synchronous) where we do not have access to the `OnKeyDown` and `OnKeyUp` events (asynchronous). This can be done by assigning the view a keyboard buffer using `view_keybuf`, which will capture the events associated with each key and allow us to consult its status at any time in a comfortable way.

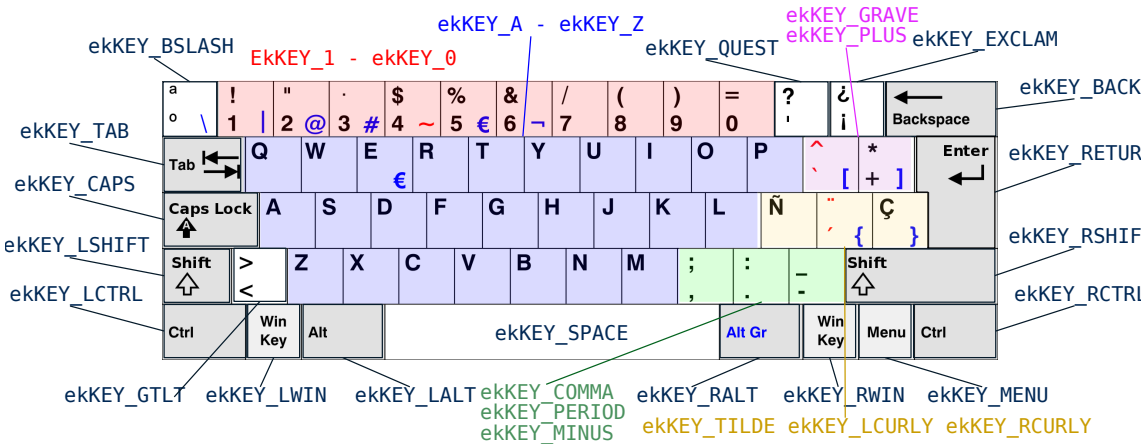


Figure 18.33: Keyboard codes.

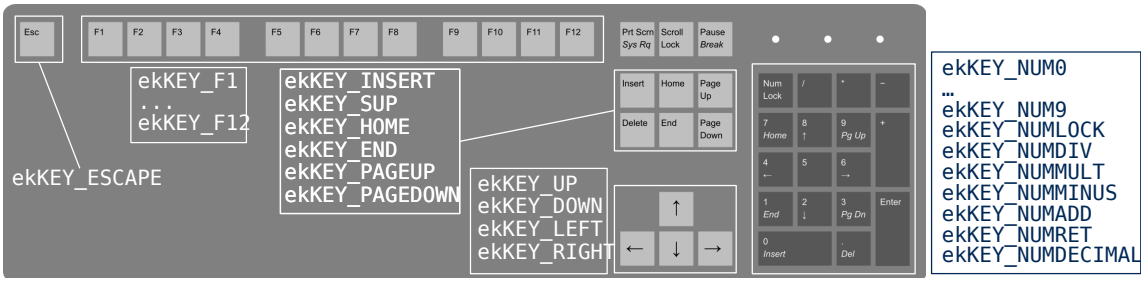


Figure 18.34: Keyboard Extended Codes.

18.13. TextView

TextView are views designed to work with rich text blocks (Figure 18.35), where fonts, sizes and colors can be combined. We can consider them as the basis of a text editor. In *"Hello TextView!Hello TextView!"* (page 511) you have an example of use.

- Use `textview_create` to create a text view.
- Use `textview_writef` to add text to the view.
- Use `textview_printf` to add text in the format of `printf`.
- Use `textview_rtf` to add content in Microsoft **RTF** format.
- Use `textview_clear` to erase all text.

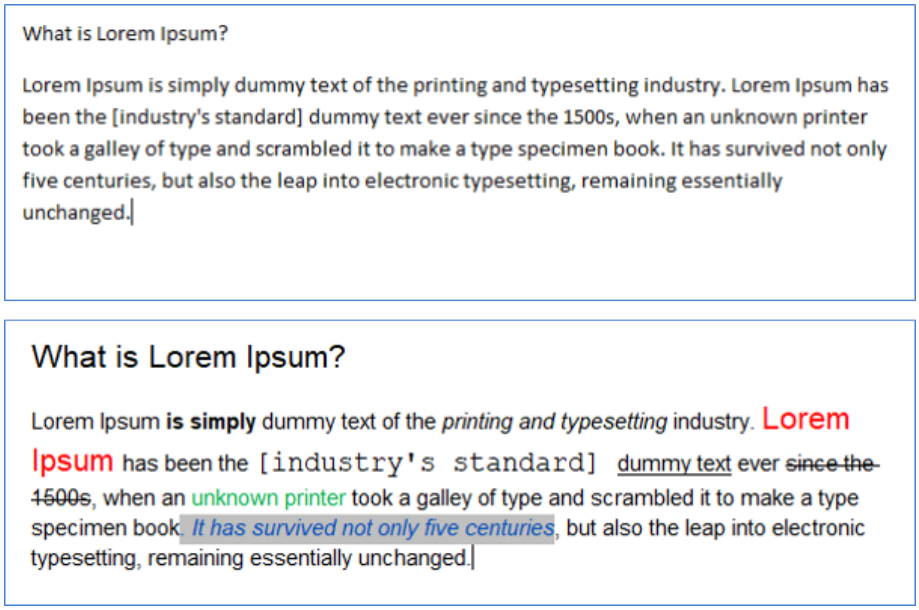


Figure 18.35: Plain text and rich text.

18.13.1. Character format

One of the advantages of rich text over plain text is the ability to combine different character formats within the same paragraph (Figure 18.36). Changes will be applied to new text added to the control.

Use `textview_family` to change the font.

Use `textview_fsize` to change the character size.

Use `textview_fstyle` to change the style.

Use `textview_color` to change the color of the text.

Use `textview_bgcolor` to change the background color of the text.

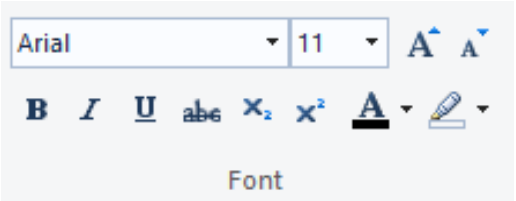


Figure 18.36: Typical Character Format Controls.

18.13.2. Paragraph format

You can also set attributes per paragraph (Figure 18.37). The new line character `'\n'` is considered the closing or end of the paragraph.

Use `textview_halign` to set to paragraph alignment.

Use `textview_lspacing` to set line spacing (line spacing).

Use `textview_bfspace` to indicate the vertical space before the paragraph.

Use `textview_afspace` to indicate the vertical space after the paragraph.

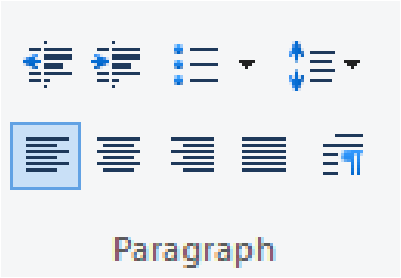


Figure 18.37: Typical controls for paragraph formatting.

18.13.3. Document format

Finally we have several attributes that affect the entire document or control.

Use `textview_units` to set the text units.

Use `textview_pgcolor` to set the background color of the control (page).

18.14. ImageView

ImageView are specialized views in visualizing images and **GIF** animations.

- Use `imageview_create` to create an image control.

Figure 18.38: ImageView in a panel.

- Use `imageview_image` to set the image that the control will display.
- Use `imageview_scale` to set the image adjustment mode.

18.15. TableView

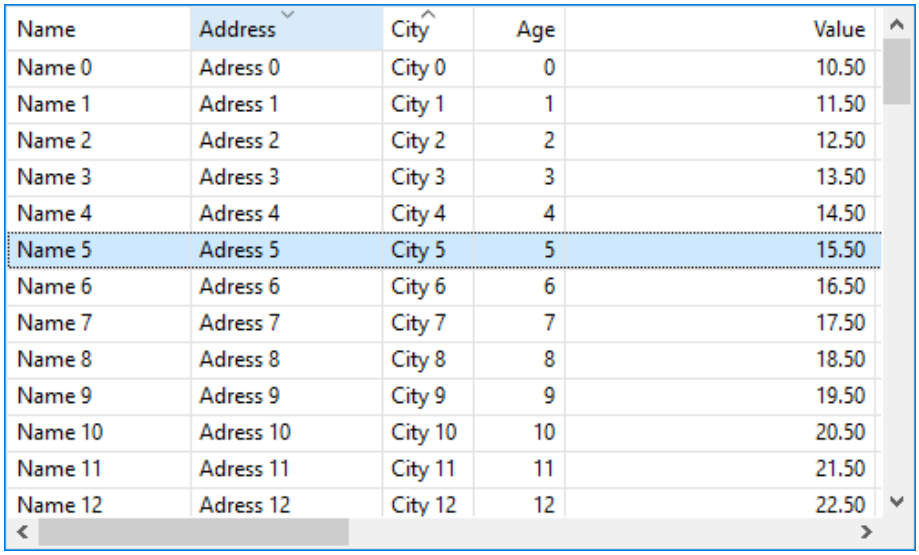
TableViews are data views that display tabulated information arranged in rows and columns (Figure 18.39), (Figure 18.40), (Figure 18.41). The control enables scroll bars and allows keyboard navigation. In *"Hello TableView!Hello TableView!"* (page 514) you have an example of use.

- Use `tableview_create` to create a table view.
- Use `tableview_new_column_text` to add a column.
- Use `tableview_size` to set the default size.

18.15.1. Data connection

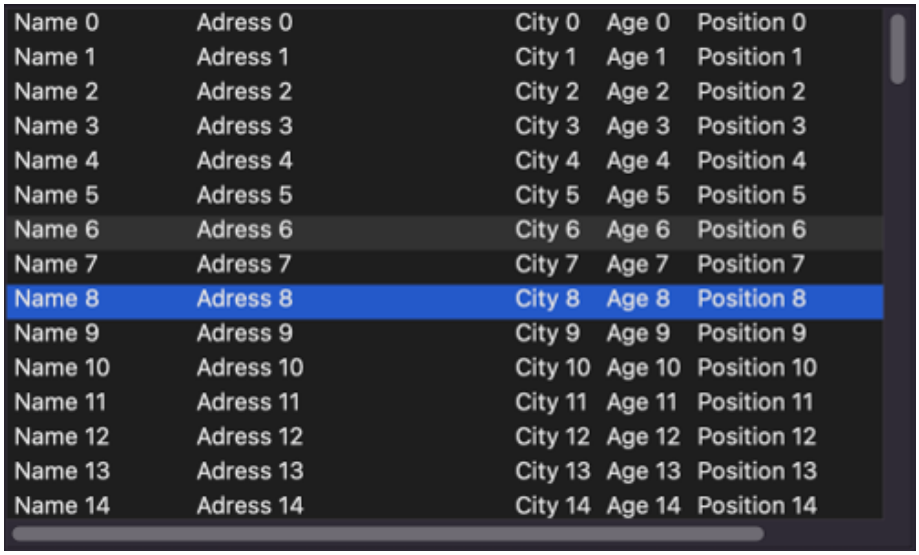
Let's think that a table can contain thousands of records and these can change at any time from different data sources (disk, network, DBMS, etc). For this reason, the TableView **will not maintain any internal cache**. It has been designed with the aim of making a quick visualization of the data, but without going into their management. Ultimately, it is the application that must provide this information in a fluid manner.

- Use `tableview_OnData` to bind the table to the data source.
- Use `tableview_update` to force an update of the table data.



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

Figure 18.39: TableView control in 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

Figure 18.40: TableView control in macOS.

When a table needs to draw its contents, in response to an `OnDraw` event, it will first ask the application for the total number of records via a `ekGUI_EVENT_TBL_NROWS` notification. With this it can calculate the size of the document and configure the scroll bars (Figure 18.42). Subsequently, it will launch successive `ekGUI_EVENT_TBL_CELL` events, where it will ask the application for the content of each cell (Figure 18.43). All these requests will be made through the *callback* function set in `tableview_OnData` (Listing 18.6).

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

Figure 18.41: TableView control in Linux.

TableView will only ask for the content of the visible part at any time.

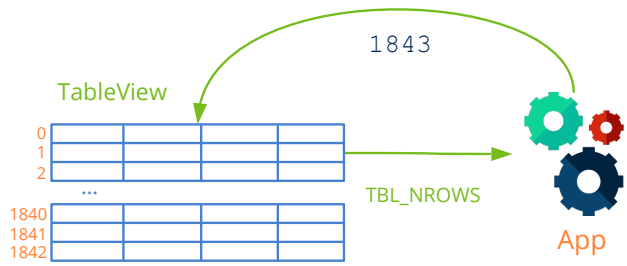


Figure 18.42: Ask for the number of rows in the data set.

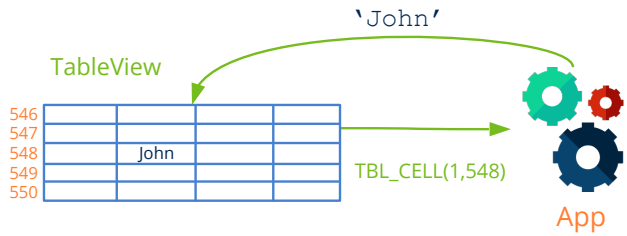


Figure 18.43: Request for the data of a cell.

Listing 18.6: Data connection example.

```
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);
```

18.15.2. Data cache

As we have already commented, at each instant the table will only show a small portion of the data set. In order to supply this data in the fastest possible way, the application can keep a cache with those that will be displayed next. To do this, before starting to draw the view, the table will send an `ekGUI_EVENT_TBL_BEGIN` type event where it will indicate the range of rows and columns that need updating (Figure 18.44). This event will precede any `ekGUI_EVENT_TBL_CELL` seen in the previous section. In the same way, once all the visible cells have been updated, the `ekGUI_EVENT_TBL_END` event will be sent, where the application will be able to free the resources in the cache (Listing 18.7).

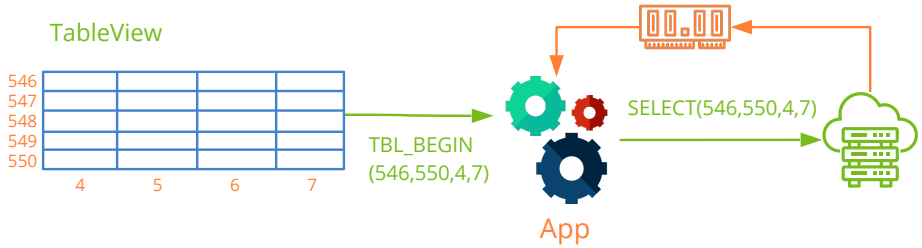


Figure 18.44: Use of data cache.

Listing 18.7: Example of using data cache.

```
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_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);
```

18.15.3. Multiple selection

When we navigate through a `TableView` we can activate the multiple selection, which will allow us to mark more than one row of the table (Figure 18.45).

- Use `tableview_multisel` to turn multiselect on or off.
- Use `tableview_selected` to get the selected rows.
- Use `tableview_select` to select a set of rows.
- Use `tableview_deselect` to deselect.
- Use `tableview_deselect_all` to uncheck all rows.
- Use `tableview_OnSelect` to receive an event when the selection changes.

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

Figure 18.45: TableView with multiple selection.

Navigation through a `TableView` works the same as other similar controls, such as the file explorer.

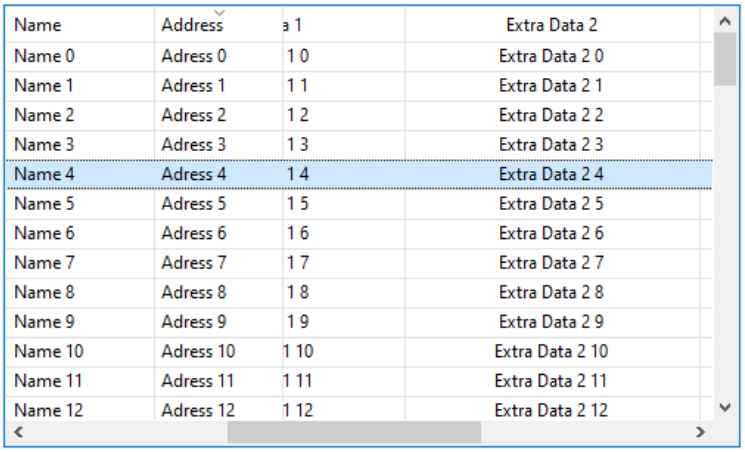
- `[UP]/[DOWN]` to scroll vertically.
- `[LEFT]/[RIGHT]` to scroll horizontally.
- `[PAGEUP]/[PAGEDOWN]` advance or reverse a page.
- `[HOME]` goes to the beginning of the table.
- `[END]` goes to the end of the table.
- `[CTRL]+click` multiple selection with the mouse.

- `[SHIFT]+[UP]/[DOWN]` multiple selection with the keyboard.

In multiple selection, an **automatic de-selection of the rows** will occur whenever we click releasing `[CTRL]` or press any navigation key releasing `[SHIFT]`. If we want to navigate without losing the previous selection, we must activate the `preserve` flag in `tableview_multisel`.

18.15.4. Configure columns

- Use `tableview_column_width` to set the width of a column.
- Use `tableview_column_limits` to set limits on the width.
- Use `tableview_column_resizable` to allow the column to be stretched or collapsed.
- Use `tableview_header_visible` to show or hide the header.
- Use `tableview_OnHeaderClick` to notify the header click.
- Use `tableview_column_freeze` to set one or more columns (Figure 18.46).



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

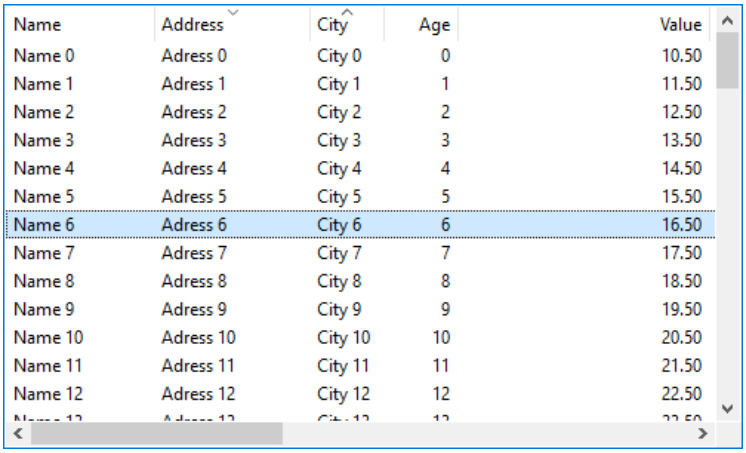
Figure 18.46: Columns 0 and 1 frozen.

18.15.5. Grid drawing

- Use `tableview_grid` to show or hide the inner lines (Figure 18.47), (Figure 18.48).

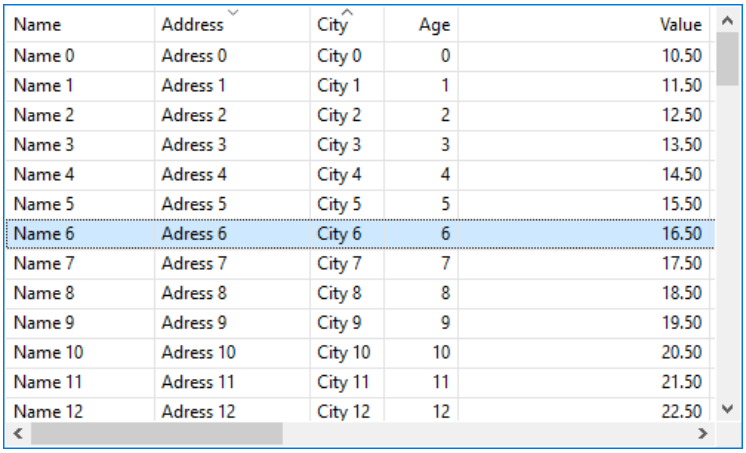
18.16. SplitView

The **SplitView** are views divided into two parts, where in each of them we place another view or a panel. The dividing line is scrollable, which allows resizing both halves,



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

Figure 18.47: TableView with no interior lines.



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

Figure 18.48: TableView with interior lines.

dividing the total size of the control between the children (Figure 18.49), (Figure 18.50), (Figure 18.51). In *"Hello SplitView!Hello SplitView!"* (page 521) you have an example of use.

- Use `splitview_horizontal` to create a split view.
- Use `splitview_size` to set the initial size.

18.16.1. Add controls

There are several functions for adding "child" controls to the splitview. The first call to any of them will place the view or panel on the left or top side. The second call will be on the right or lower side. Successive calls will generate an error.

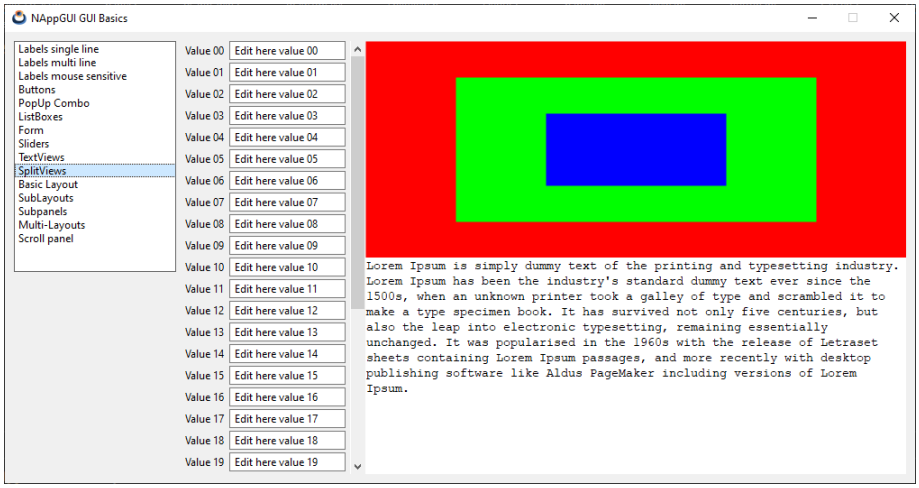


Figure 18.49: SplitView in Windows.

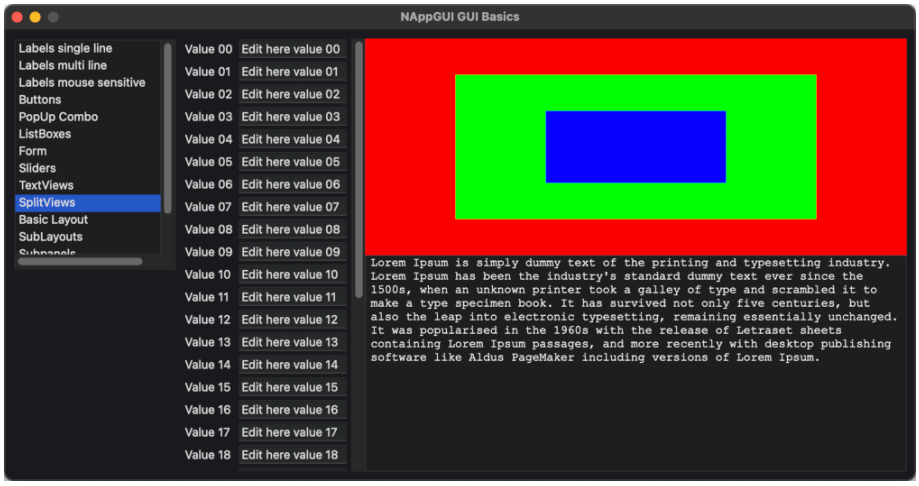


Figure 18.50: SplitView in macOS.

- Use `splitview_view` to add a custom view.
- Use `splitview_panel` to add a panel.

18.16.2. Split modes

We have two modes of behavior of the dividing bar and both are activated from this function:

- Use `splitview_pos` to set the mode of the splitter.

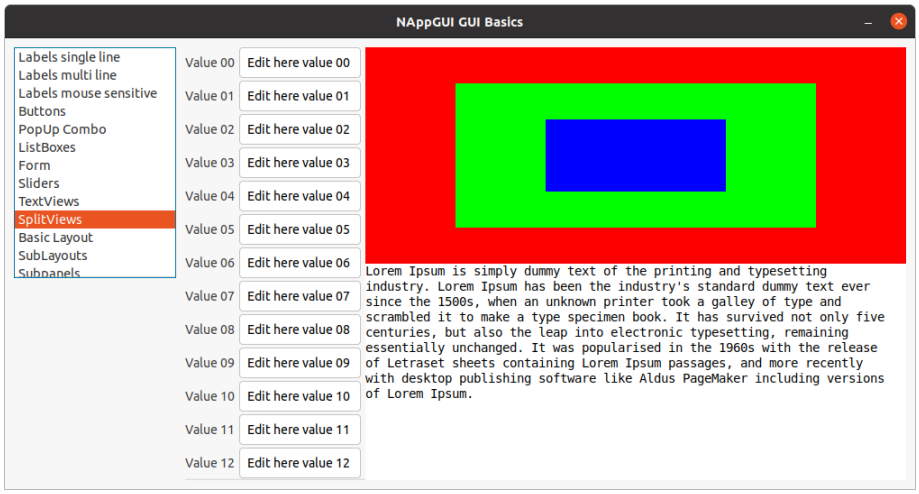


Figure 18.51: SplitView in Linux.

- Proportional mode: The position of the divider will always remain constant with respect to the size of the splitview. That is, a value of 0.3 means that the left view will always occupy 1/3 of the total size and the right view 2/3. To do this, indicate a value between 0 and 1 in the `pos` parameter.
- Fixed mode: Resizing the splitview will always leave one of the parts with a constant size. If `pos > 1` the left/top child will keep the indicated number of pixels. On the contrary, if `pos < 0` the same will happen with the right/bottom view.

The ratio or value will change if the user drags the dividing line, but the operating mode will not.

18.17. Layout

A **Layout** is a virtual and transparent grid always linked with a `Panel` which serves to locate the different interface elements (Figure 18.52). Its inner cells have the ability to be automatically sized according to their content, which results in great portability because it is not necessary to indicate specific coordinates or sizes for the controls. To illustrate the concept, we will slightly simplify the code of *"Hello Edit and UpDown!Hello Edit and UpDown!"* (page 503) (Listing 18.8), whose result we can see in (Figure 18.53).

- Use `layout_create` to create a new layout.
- Use `layout_label` and similars to place controls in the different cells.

Listing 18.8: Layout with two columns and five rows.

```
Layout *layout = layout_create(2, 5);
```

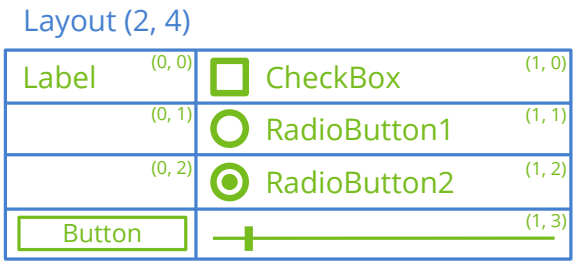


Figure 18.52: A layout is used to locate controls in the panel area.

```
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);
```

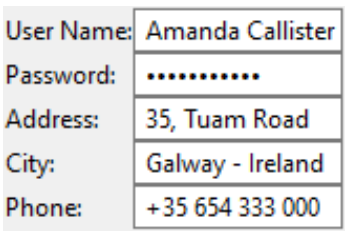


Figure 18.53: Result of (Listing 18.8).

18.17.1. Natural sizing

The result of (Figure 18.53), although it is not very aesthetic, it is what we call **natural sizing** which is the default layout applied depending on the content of the cells. In (Table 18.2) we have the default measurements of each control. The column width is fixed to that of the widest element and the height of the rows is calculated in the same way. The final size of the layout will be the sum of the measures of both columns and rows.

Control	Width	Height
`Label`	Adjusted to the text.	Adjusted to the text considering `'\n'`
`Button` (push)	Adjusted to text + margin.	According to the theme of the OS.
`Button` (check/radio)	Adjusted to text + icon.	Adjusted to the icon.
`Button` (flat)	Adjusted to the icon + margin.	Adjusted to the icon + margin.
`PopUp`	Adjusted to the longest text.	According to the theme of the OS.
`Edit`	100 Units (px).	Adjusted to text + margin.
`Combo`	100 Units (px).	According to the theme of the OS.
`ListBox`	128 px or `listbox_size`.	128 px or `listbox_size`.
`UpDown`	According to the theme of the OS.	According to the theme of the OS.
`Slider` (horizontal)	100 Units (px).	According to the theme of the OS.
`Slider` (vertical)	According to the theme of the OS.	100 Units (px).
`Progress`	100 Units (px).	According to the theme of the OS.
`View`	128 px or `view_size`.	128 px or `view_size`.
`TextView`	256 px or `textview_size`.	144 px or `textview_size`.
`ImageView`	64 px or `imageview_size`.	64 px or `imageview_size`.
`TableView`	256 px or `tableview_size`.	128 px or `tableview_size`.
`SplitView`	128 px or `splitview_size`.	128 px or `splitview_size`.
`Panel`	Natural size.	Natural size.
`Panel` (with scroll)	256 px or `panel_size`.	256 px or `panel_size`.

Table 18.2: Natural dimensioning of controls.

The margins and constants applied to the controls are those necessary to comply with the **human guidelines** of each window manager. This means that a *PushButton* with the

text "Hello" will not have the same dimensions in WindowsXP as in macOS Mavericks or Ubuntu 16.

Empty cells will be 0-sized and will not affect the composition.

18.17.2. Margins and format

The natural sizing we have just seen adjusts the panel to the minimum size necessary to correctly house all the controls, but it is not always aesthetic. We can shape it by adding margins or forcing a given size for rows and columns (Listing 18.9) (Figure 18.54).

- Use `layout_hsize` to force the width of a column.
- Use `layout_vsize` to force the height of a row.
- Use `layout_hmargin` to establish an inter-column margin.
- Use `layout_vmargin` to establish an inter-row margin.
- Use `layout_margin` to set a margin at the edge of the layout.

Listing 18.9: Applying format to (Listing 18.8).

```
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);
```

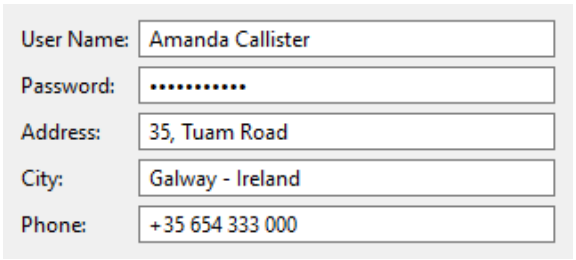


Figure 18.54: Result of (Listing 18.9).

18.17.3. Alignment

It is usual for the width of a control to be less than the width of the column that contains it, either because a fixed width has been forced or because there are wider elements in the same column. In these cases, we can indicate the horizontal or vertical alignment of the control with respect to the cell (Figure 18.55). In (Table 18.3) you have the default alignments.

- Use `layout_halign` to change the horizontal alignment of a cell.
- Use `layout_valign` to change the vertical alignment of a cell.

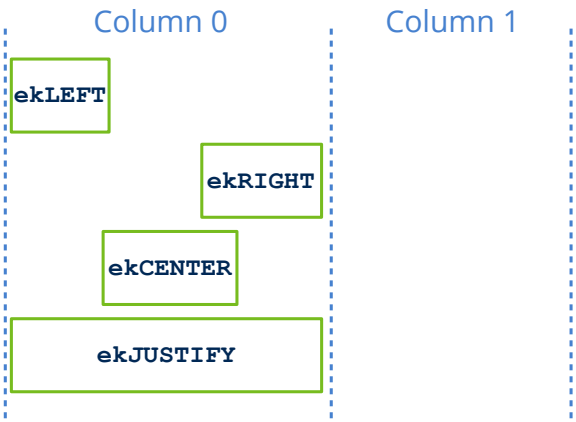


Figure 18.55: Horizontal alignment.

Control	Horizontal	Vertical
`Label`	`ekLEFT`	`ekCENTER`
`Button` (push)	`ekJUSTIFY`	`ekCENTER`
`Button` (others)	`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`
`Progress`	`ekJUSTIFY`	`ekCENTER`
`View`	`ekJUSTIFY`	`ekJUSTIFY`
`TextView`	`ekJUSTIFY`	`ekJUSTIFY`
`ImageView`	`ekJUSTIFY`	`ekJUSTIFY`
`TableView`	`ekJUSTIFY`	`ekJUSTIFY`

Control	Horizontal	Vertical
SplitView	ekJUSTIFY	ekJUSTIFY
Layout (sublayout)	ekJUSTIFY	ekJUSTIFY
Panel	ekJUSTIFY	ekJUSTIFY

Table 18.3: Default alignment of controls.

18.17.4. Sub-layouts

Consider now the panel of (Figure 18.56). It is not difficult to realize that this arrangement does not fit in any way in a rectangular grid, so it is time to use **sublayouts**. In addition to individual controls, a cell also supports another layout, so we can divide the original panel into as many parts as necessary until the desired layout is achieved. The main layout will size each sublayout recursively and integrate it into the final composition. In *"Hello Sublayout!Hello Sublayout!"* (page 539) you have the code that generates this example.

- Use `layout_layout` to assign a complete layout to a cell in another layout.

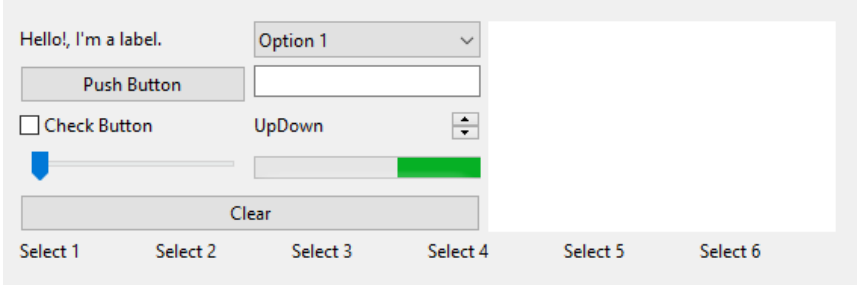


Figure 18.56: Complex panel composition.

In this case we have applied the philosophy of divide and conquer, to ensure that each part fits into an individual grid (Figure 18.57). Each sublayout has been coded in an independent function to give greater consistency to the code, applying margins and format individually within each of them (Listing 18.10).

Listing 18.10: Sublayout integration (partial).

```
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);
```

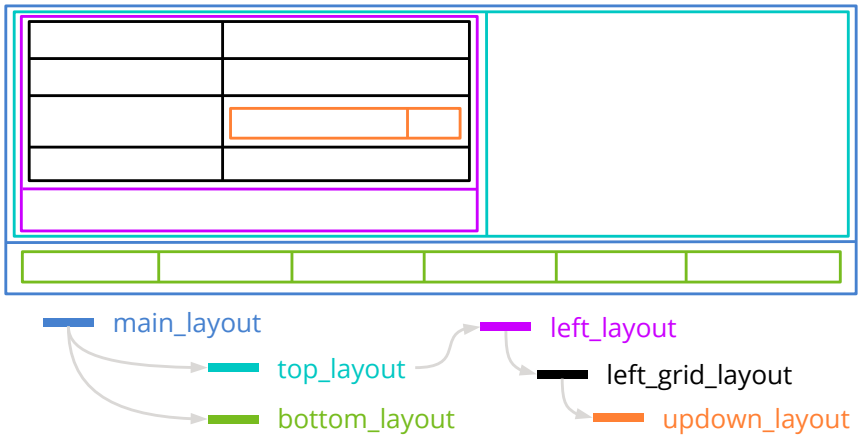


Figure 18.57: Sublayouts needed to compose the (Figure 18.56) panel.

```
    layout_vmargin(layout1, 0, 5);
    return layout1;
}
```

18.17.5. Cell expansion

On certain occasions, the size of a layout is forced by external conditions. This happens when we have a sublayout in a cell with `ekJUSTIFY` alignment (internal expansion) or when the user changes the size of a resizable window (external expansion). This will produce an "pixel excess" between the natural sizing and the actual cell size (Figure 18.58). This situation is resolved by distributing the pixel surplus equally among all the sublayout columns, which in turn, will be recursively expanding until they reach an empty cell or an individual control. We can change this equitable distribution through these functions:

- Use `layout_hexpand` to expand a single cell and leave the rest with its default size.
- Use `layout_hexpand2` to expand two cells indicating the growth rate of each.
- Use `layout_hexpand3` to expand three cells.

The vertical expansion works exactly the same, distributing the excess space between the rows of the layout.

18.17.6. Tabstops

Normally we will use the `[TAB]` key and the `[SHIFT]+[TAB]` combination to navigate through the different controls of a window or form. Terms like **taborder** or **tabstop** refer to both the navigation order and the membership (or not) of an element in said

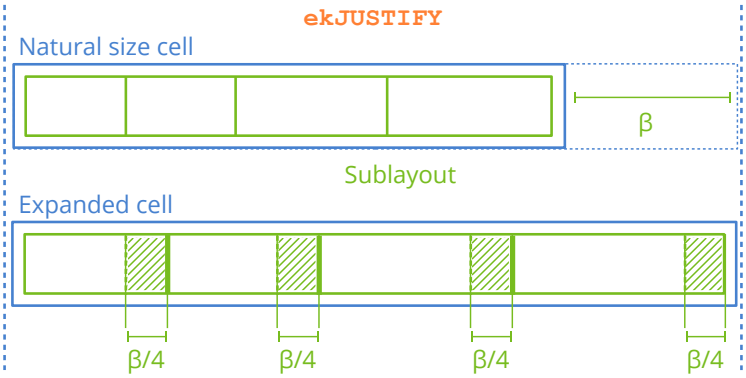


Figure 18.58: When the size of the sublayout is given by external conditions, the excess of pixels is equally distributed between the columns (horizontal expansion) and rows (vertical expansion).

list. While it is possible to arrange the elements of a *tab-list* randomly, layouts provide a coherent natural order based on the placement of controls. By default, each layout creates a *tab-list* going through all its cells by rows , but we can change it:

- Use `layout_taborder` to arrange the *tab-list* by rows or columns.
- Use `layout_tabstop` to add or remove controls from the *tab-list*.

Figure 18.59: Taborder by rows in layouts and sublayouts.

Not every cell in a layout has to be a *tabstop*, since it doesn't make sense for static controls like `Label` to receive keyboard focus. In (Table 18.4) you have which controls are included by default in that list. With `layout_tabstop` you can add or remove controls from the *tab-list*. Most of the time we will use it than allowing certain custom views `View` to receive keyboard focus.

Control	Included
`Label`	NO

Control	Included
`Button`	YES
`PopUp`	YES
`Edit`	YES
`Combo`	YES
`ListBox`	YES
`UpDown`	NO
`Slider`	YES
`Progress`	NO
`View`	NO
`TextView`	NO
`ImageView`	NO
`TableView`	YES
`SplitView`	YES
`Layout` (sublayout)	YES
`Panel`	YES

Table 18.4: Controls included in the *tab-list*.

When the taborder enters a sublayout, it will follow the local order of the latter. When exiting the sublayout it will continue with the main order.

18.18. Cell

Cells are the inner elements of a *"Layout"* (page 332) and will house a control or a sublayout (Figure 18.60).

- Use `layout_cell` to get the cell.
- Use `cell_control` to get the control inside.
- Use `cell_layout` to get the inner sublayout.
- Use `cell_enabled` to enable or disable the controls.
- Use `cell_visible` to show and hide the content.
- Use `cell_padding` to set the (Figure 18.61) padding.

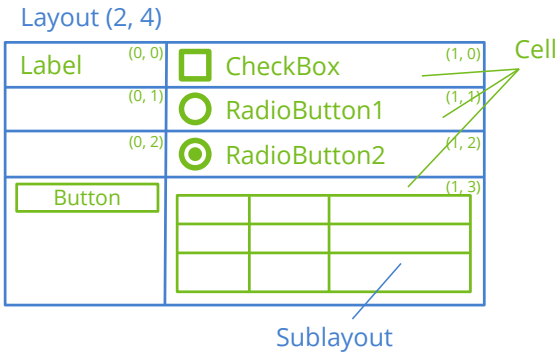


Figure 18.60: Cells inside a Layout

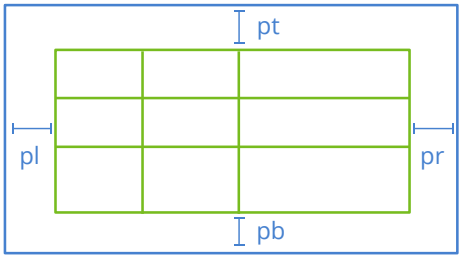


Figure 18.61: Interior padding of a cell.

18.19. Panel

A **Panel** is a control within a window that groups other controls. It defines its own reference system, that is, if we move a panel all its descendants will move in unison since their locations will be relative to its origin. It will support other (sub)-panels as descendants, which allows to form a **Window Hierarchy** (Figure 18.62). For portability, this **Gui** library does not support specific coordinates and sizes for elements linked to a panel, but the association is carried out by a `Layout` object which is responsible for calculating at runtime the final locations of controls based on the platform and window manager. In *"Hello Subpanel!Hello Subpanel!"* (page 543) you have an elementary example of using panels.

- Use `panel_create` to create a new panel.
- Use `panel_scroll` to create a panel with scroll bars.
- Use `panel_layout` to add child controls to the panel.
- Use `panel_size` to set the default size of the visible area.

Each panel supports several layouts and allows you to switch between them at runtime (Figure 18.63). This allows to create dynamic responsive interfaces with very little effort, since the panel itself is responsible for linking and sizing the controls according to the active layout in each case. In *"Hello Multi-layout!Hello Multi-layout!"* (page 544) you have an example.

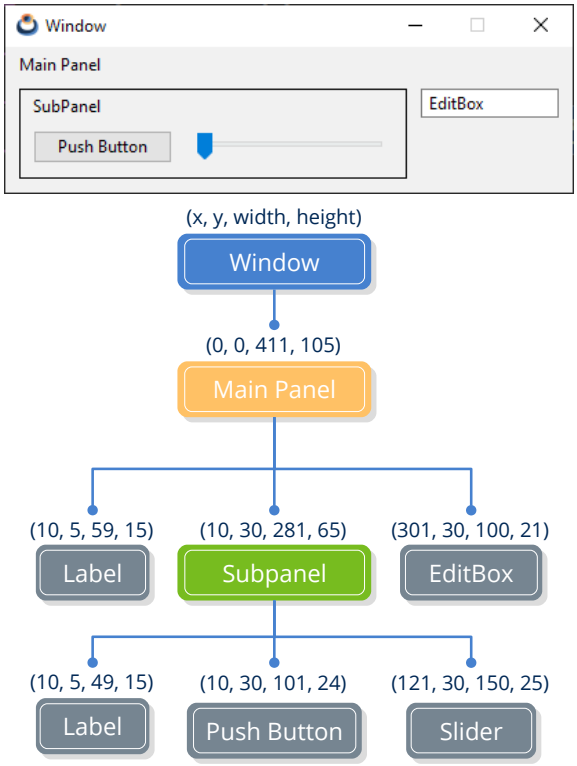


Figure 18.62: Window hierarchy.

- Use `panel_visible_layout` to change the layout.

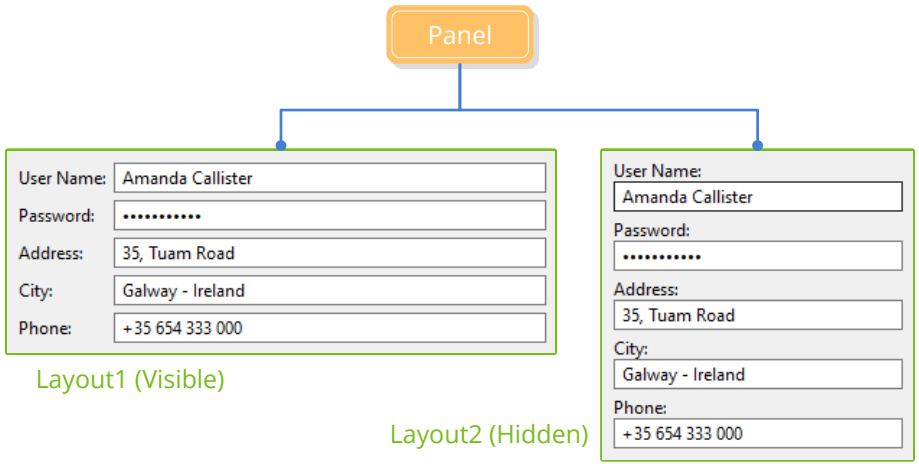


Figure 18.63: Panel with two different organizations for the same controls.

Because the layouts are logical structures outside the window hierarchy, they can share controls as they are linked to the same panel (Figure 18.64). What is not allowed is to use

the same objects in different panels, due to the hierarchy concept.

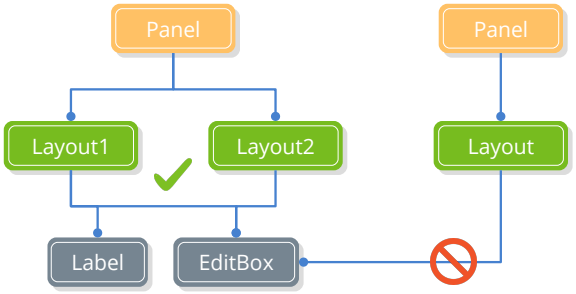


Figure 18.64: It is possible to reuse the same components between layouts of the same panel.

18.19.1. Understanding panel sizing

We are going to show, by means of an example, the logic behind the composition and dimensioning of panels. We start with (Listing 18.11) where we create a relatively large panel in height.

Listing 18.11: Composition of a panel with multiple edit rows.

```
static Window *i_window(void)
{
    uint32_t i, n = 20;
    Window *window_create(ekWINDOW_STDRES);
    Panel *panel = panel_create();
    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;
}
```

- Lines 3-6 create the window, panel, and layout.
- Loop 8-19 adds various labels and edit boxes to the layout.
- Loop 21-22 establishes a small gap between rows.
- Lines 24-25 establish a column spacing and border margin.
- Lines 26-27 link the layout to the panel and the layout to the window.

The result of this code is the *"Natural sizingNatural sizing"* (page 334) of the panel (Figure 18.65), which defaults to a width of 100 pixels for the editing controls. Labels fit to the text they contain. Separations and margins have also been applied.

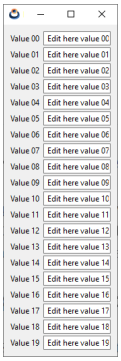


Figure 18.65: Natural sizing of the panel defined in (Listing 18.11).

In this case it is possible to resize the window, since we have used the `ekWINDOW_STDRES` flag when creating it (Figure 18.66).

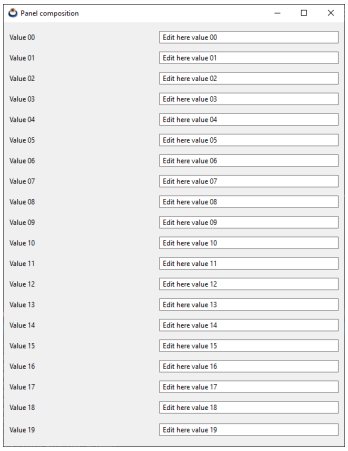


Figure 18.66: Behavior of the panel when the window grows.

This behavior may not be the most appropriate for the case at hand. By default, the layout performs the *"Cell expansionCell expansion"* (page 338) proportionally. But what we really want is to "stretch" the editing controls so that the rows keep their default height (Listing 18.12).

Listing 18.12: Change in horizontal and vertical expansion.

```
Layout *layout = layout_create(2, n + 1);
...
layout_hexpand(layout, 1);
layout_vexpand(layout, n);
```

The previous lines cause the horizontal expansion to fall exclusively on column 1 (that of the EditBoxes). On the other hand, an extra empty row has been created, pouring all the vertical expansion into it (Figure 18.67).

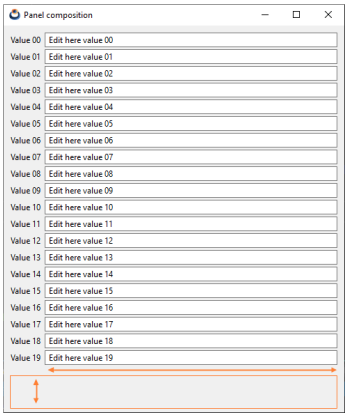


Figure 18.67: Desired behavior, when the window expands.

Although the panel now behaves correctly when the window grows, we have difficulties when we want to "shrink" it below a certain limit (Figure 18.68). This is because natural dimensioning imposes a minimum size, since there comes a time when it is impossible to reduce the controls associated with the layout.

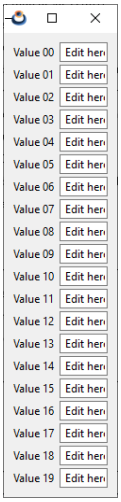


Figure 18.68: Minimum panel size.

This can be a problem as we may have panels large enough that they even exceed the

size of the monitor and cannot be fully displayed. To solve this, we can set a default size for the entire panel (Listing 18.13), which will be the one displayed when the window starts (Figure 18.69).

Listing 18.13: Panel default size.

```
...
panel_size(panel, s2df(400, 300));
...
```

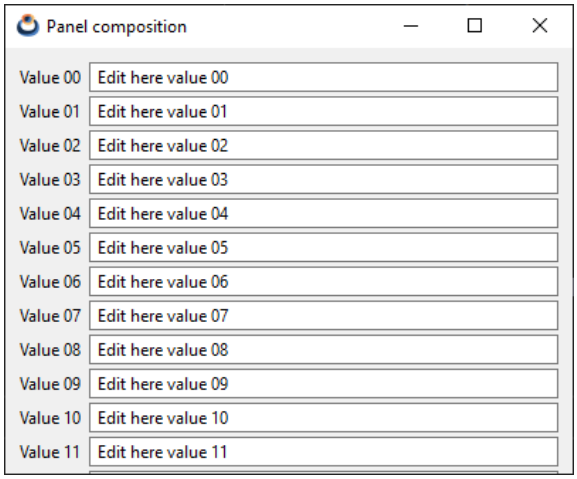


Figure 18.69: Natural sizing, forced to 400x300.

This command decouples, in a way, the size of the panel from the size of its content. In this way, the Layout is free to reduce the size of the view, regardless of whether or not it can display the entire content (Figure 18.70).

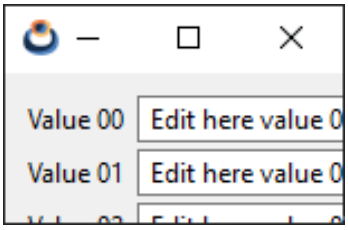


Figure 18.70: Panel boundary reduction.

And finally, if we want, we can create the panel with scroll bars (Listing 18.14) and scroll through the non-visible content (Figure 18.71).

Listing 18.14: Panel with scroll bars.

```
...
Panel *panel = panel_scroll(TRUE, TRUE);
...
```

And, of course, everything said will work the same on any platform .

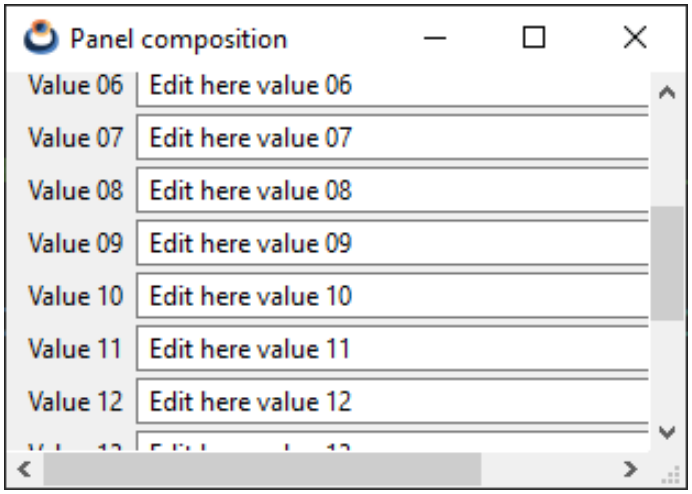


Figure 18.71: Panel with scroll bars.

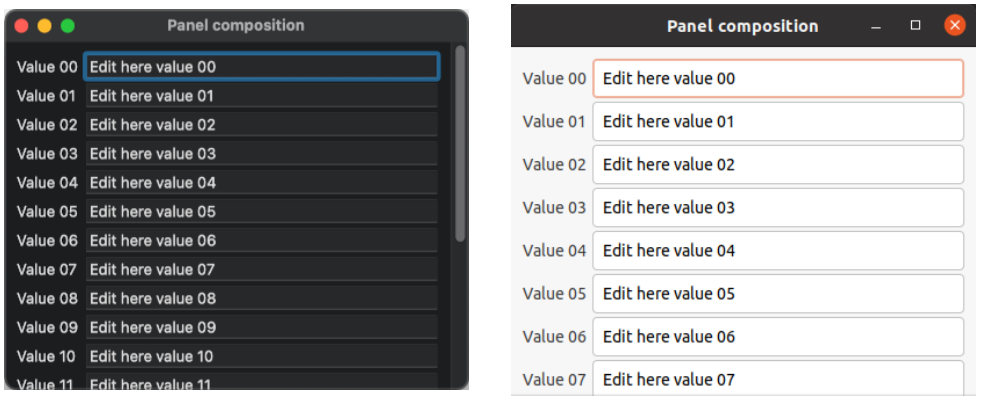


Figure 18.72: Our panel running on macOS and Linux.

18.20. Window

Window objects are the top-level containers within the user interface (Figure 18.73). They are made up of the title bar, where the close, maximize and minimize buttons are located, the interior zone and the frame. If the window supports resizing, said frame can be dragged with the mouse to change its size. The interior zone or client area of the window is configured by means of a main *"Panel"* (page 341). In *"Hello World!"* (page 23) you have a simple example of composing and displaying a window.

- Use `window_create` to create a window.
- Use `window_panel` to assign the main panel.
- Use `window_show` to show a window.
- Use the `ekWINDOW_TITLE` flag to include the title bar.

- Use `window_title` to assign a title.

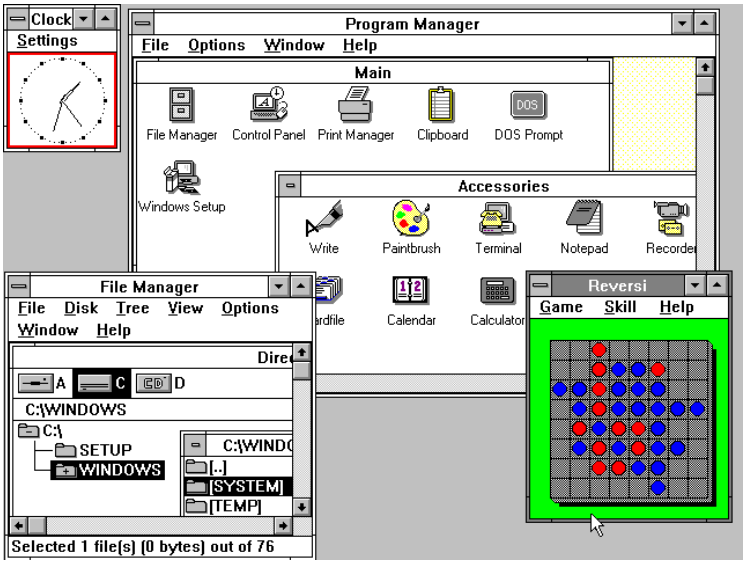


Figure 18.73: The concept of a window appears from the first desktop systems.

18.20.1. Window size

In principle, the size of the window is calculated automatically based on the *"Natural sizingNatural sizing"* (page 334) of its main panel, but it can be altered at any time.

- Use `window_size` to resize the main panel.
- Use the `ekWINDOW_MAX` flag to include the maximize button in the title bar.
- Use the `ekWINDOW_MIN` flag to include the minimize button in the title bar.
- Use the `ekWINDOW_RESIZE` flag to create a window with resizable borders.

The change in the dimensions of the client area implies a re-location and re-sizing of the interior controls. This is handled automatically by `Layout` objects based on how their *"Cell expansionCell expansion"* (page 338) has been configured, which will propagate recursively through all sublayouts. In *"Die"* (page 391) you have an example of resizing a window (Figure 18.74).

18.20.2. Closing the window

Normally a window is closed by pressing the `[X]` button located to the right of the title bar. But sometimes it can be useful to also close it with the `[ENTER]` or `[ESC]` keys.

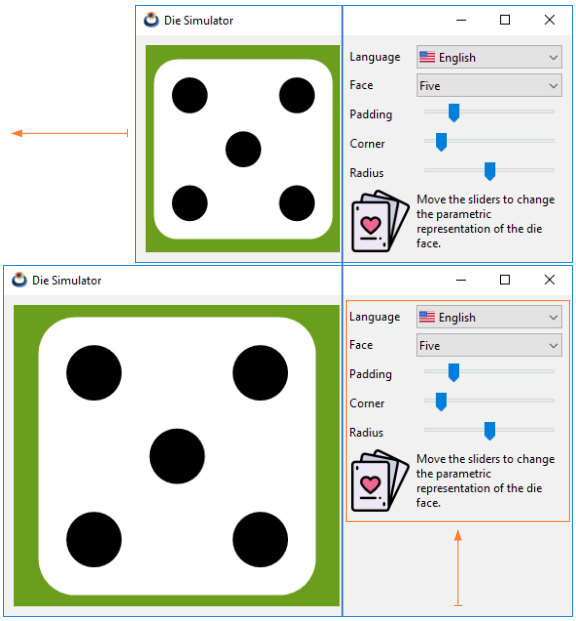


Figure 18.74: Resizing the window in the demo **Die**.

Closing a window implies hiding it, but not destroying it. That is, we can show an already closed window again using `window_show`. In the case that the closing is conditioned to a state of the application, such as saving a file for example, we must assign a handler through `window_OnClose` and decide there whether to close it or not.

- Use `window_hide` to hide a window.
- Use `window_destroy` to permanently destroy a window.
- Use the `ekWINDOW_CLOSE` flag to include the close button in the title bar.
- Use the `ekWINDOW_RETURN` flag to enable `[ENTER]` closing.
- Use the `ekWINDOW_ESC` flag to enable `[ESC]` closing.
- Use the `window_OnClose` flag to prevent the closing of a (Listing 18.15) window.

Listing 18.15: Prevents closing the window.

```
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));
```

Destroying a window implicitly destroys all of its internal elements and controls.

18.20.3. Modal windows

They are those that, when launched, block the previous window (or parent) until it is closed (Figure 18.75). Being "modal" or not is not a characteristic of the window itself, but of the way it is launched. In *"Hello Modal Window!Hello Modal Window!"* (page 523) you have an example of use.

- Use `window_modal` to display a window in modal mode.
- Use `window_stop_modal` to hide it and stop the modal loop.

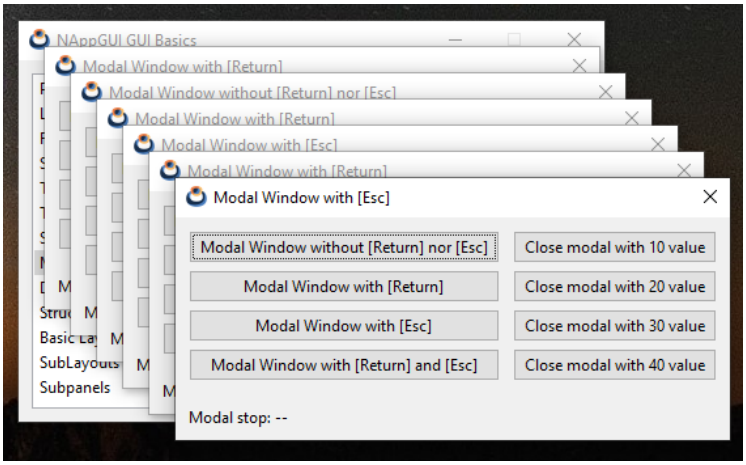


Figure 18.75: Multiple modal windows.

After calling `window_modal`, the program stops at this point, waiting for the window to close, which can be done using `[X]`, `[ENTER]`, `[ESC]` or by calling `window_stop_modal` (Listing 18.16). The value returned by this function will be:

- `ekGUI_CLOSE_ESC` (1). If the modal window was closed by pressing `[ESC]`.
- `ekGUI_CLOSE_INTRO` (2). If the modal window was closed by pressing `[ENTER]`.
- `ekGUI_CLOSE_BUTTON` (3). If the modal window was closed by pressing `[X]`.
- The value indicated in `window_stop_modal`.

Listing 18.16: Using modal windows.

```
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);
```

By default, the modal window will be hidden after receiving the call to `window_stop_modal`, but it will not be destroyed as we indicated above. On certain occasions (although not very common), we may want to relaunch the window after finishing the modal cycle without producing an unsightly "flicker" due to a new (and fast) display after closing the window.

- Use the `ekWINDOW_MODAL_NOHIDE` flag when creating the window to prevent it from being hidden after the modal loop.

18.20.4. Keyboard focus

There are applications that make intensive use of the keyboard, or even do without the mouse. This is why we have to be very clear about how the different elements behave when pressed. The control that currently receives the keystrokes is called **keyboard focus**. Bla bla bla.

- Use `window_focus` to change the keyboard focus.
- Use `window_get_focus` to get keyboard focus.

On the other hand, it may be useful to move the keyboard focus from the code itself and

not wait for the user to press `[TAB]`. In *"Hello IP-Input!Hello IP-Input!"* (page 548) you have several `Edit` that pass to the next control when exactly three numbers are entered.

- Use `window_next_tabstop` to move focus to the next control.
- Use `window_previous_tabstop` to move focus to the previous control.

Typically, tabstops will work **cyclically** (by default). That is, if the last control in the window has the keyboard focus and we press `[TAB]`, the focus will go back to the first control in the window (cycle), as we see in (**??**). It is possible to disable this behavior, leaving the focus fixed on the last control even if we repeatedly press the `[TAB]` key. Likewise, the focus will remain fixed on the first control in the window even if we press `[SHIFT]+[TAB]`.

- Use `window_cycle_tabstop` to enable/disable cycling tabstops.

18.20.5. Default button

- Use `window_defbutton` to set the default button.

18.20.6. Hotkeys

Normally, the keyboard focus will be fixed to some control inside the window like `Edit`, `Button` or `View`. But it is possible that we want to define global actions associated with a specific key.

- Use `window_hotkey` to assign an action to a key.

hotkeys will take precedence over keyboard (Figure 18.76) focus. That is, if we have an action linked to the `[F9]` key, it will be executed when the key is pressed and the `ekGUI_EVENT_KEYDOWN`(F9) event will not be received by the control that has the focus.

18.21. GUI Data binding

By **GUI Data Binding** we mean automatic mapping between program variables and user interface controls (Figure 18.77). In this way both will be synchronized without the programmer having to do any extra work such as capturing events, assigning values, checking ranges, etc. In *"Hello Gui Binding!Hello Gui Binding!"* (page 528) you have the complete source code of the example that we will show below.

18.21.1. Basic type binding

We start from a data structure composed of several basic types fields (Listing 18.17), where no other structures or objects are nested.

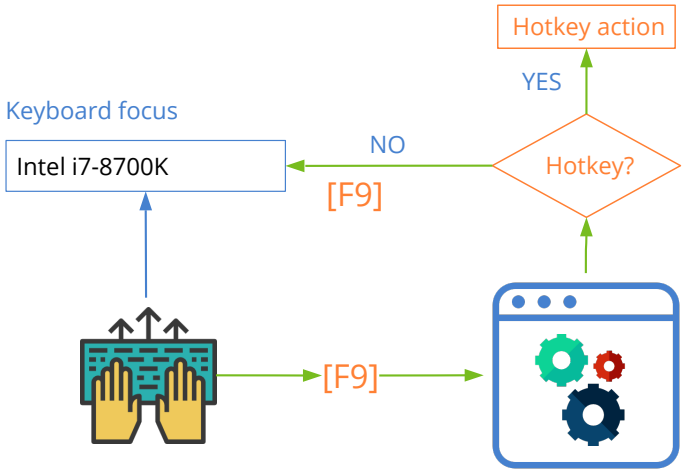


Figure 18.76: Processing a hotkey.

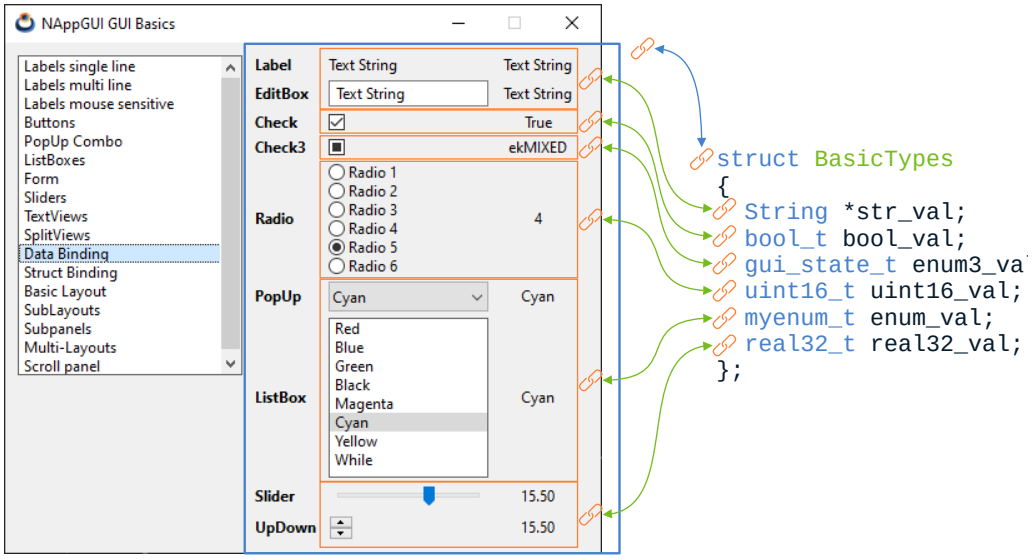


Figure 18.77: Automatic data synchronization with the user interface.

Listing 18.17: Simple data model.

```
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;
};
```

The first thing we must do is register the fields of the structure with `dbind` (Listing 18.18):

Listing 18.18: Register in `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` is a registry, within the application, that allows automating certain operations on the data, as well as establishing ranges, precisions or aliases. Its use goes beyond graphical user interfaces. More information in "Data binding" (page 225).

On the other hand, we build a *"Layout"* (page 332) that hosts the different controls of the user interface (Listing 18.19):

Listing 18.19: Interface controls organized in a layout (Figure 18.77).

```
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;
}
```

Now we will link the cells of our layout with the fields of the structure (Listing 18.20). Pay attention that we have **not yet created any object** of type `BasicTypes`. Therefore, it is a semantic link where memory positions do not intervene, but the displacements (offset) of the fields within the data structure.

- Use `cell_dbind` to bind a field to an individual cell.
- Use `layout_dbind` to link a structure with a layout.
- Use `layout_cell` to get a cell from a Layout.

Listing 18.20: Binding variables with cells in the 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);
```

When linking a data structure with `layout_dbind` we must bear in mind that the cells of said layout **can only be associated with fields of the same structure**. Otherwise,

we will get a run-time error, due to the data inconsistency that would occur. In other words, we cannot mix structures within the same layout.

> *Isolated variables cannot be used in Data Binding. They must all belong to a `struct` since, internally, the relations (Layout –> Struct) and (Cell –> Field or Variable) are established.*

Finally, we will associate an object of type `BasicTypes` with the layout created previously (Listing 18.21).

- Use `layout_dbind_obj` to bind an object to the user interface.

Listing 18.21: Binding an object to the interface.

```
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);
```

- You can change the object being "edited" at any time, with a new call to `layout_dbind_obj` .
- If we pass `NULL` to `layout_dbind_obj` the cells linked to fields of the structure will be disabled.

18.21.2. Limits and ranges

Keep in mind that the expressiveness of controls will, generally, be well below the range of values supported by data types (Listing 18.22). For example, if we link a `uint16_t` with a `RadioGroup` the latter will only support values between `0` and `n-1`, where `n` is the total number of radios. The controls are set up to handle out-of-range values as consistently as possible, but this does not exempt the programmer from getting it right. In (Table 18.5) you have a summary of the data types and ranges supported by the standard controls.

Listing 18.22: Value not representable in the RadioGroup of (Figure 18.77).

```
data->uint16_val = 1678;
cell_dbind(layout_cell(layout, 1, 4), BasicTypes, uint16_t, uint16_val);
```

Control	Data Type
"Label" (page 304)	String, Number, Enum
"Edit" (page 309)	String, Number
"Button" (page 306) (CheckBox)	Boolean
"Button" (page 306) (CheckBox3)	Enum (3 values), Integer (0,1,2)
"RadioGroupRadioGroup" (page 307)	Enum, Integer (0,1,2...n-1)
"PopUp" (page 308)	Enum, Integer (0,1,2...n-1)
"ListBox" (page 312)	Enum, Integer (0,1,2...n-1)
"Slider" (page 314)	Number (min..max)
"UpDown" (page 314)	Enum, Number

Table 18.5: Data types and ranges of GUI controls.

18.21.3. Nested structures

Let's now look at a somewhat more complicated data model, which includes nested structures in addition to the basic types (Figure 18.78). In this case we have a structure called `StructTypes` that contains instances of another structure called `Vector` (Listing 18.23). You can find the complete source code for this second example at *"Hello Struct Binding!Hello Struct Binding!"* (page 532).

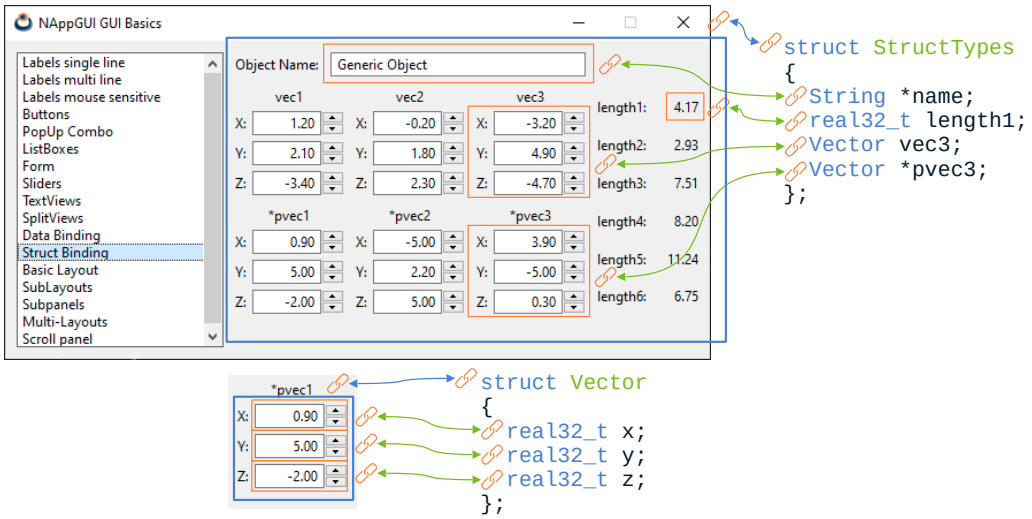


Figure 18.78: Data binding with substructures.

Listing 18.23: Data model with nested structures and registry in dbind.

```
typedef struct _vector_t Vector;
typedef struct _structtypes_t StructTypes;

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);
```

We started with the same methodology that we used with the first example. We create a layout and link it with the `Vector` structure (Listing 18.24). This does not present problems, as it is composed exclusively of basic types `real32_t`.

Listing 18.24: Layout for editing objects of type `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;
}
```

The idea now is to use this function to create *"Sub-layoutsSub-layouts"* (page 337) and associate them to cells of a higher level layout, which can support objects of type `StructTypes` (Listing 18.25). Sub-layouts of type `Vector` are linked to the fields `{Vector vec1, Vector * pvec1, ...}` using `cell_dbind`, so similar to how we did it with the basic types.

Listing 18.25: Layout that supports objects of type `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;
}
```

And finally, we only have to link objects of type `StructTypes` with the main layout (Listing 18.26). DBind will detect sub-layouts of type `Vector` and will automatically associate the corresponding sub-objects (by value or by pointer). Therefore, only one call to `layout_dbind_obj` will be necessary (the one of the main object).

Listing 18.26: Associate object and sub-objects to a 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);
```

In summary:

- For each sub-structure we create a sub-layout, linking the fields locally.
- The cells that contain these sub-layouts will be linked to the main structure.
- We assign the object to edit to the main layout.

18.21.4. Notifications and calculated fields

If we apply what was seen in the previous sections, the synchronization between data and interface is carried out in these two situations:

- When the program calls `layout_dbind_obj`. At that time the interface will reflect the state of the object.
- When the user manipulates any control, then the object's value will be updated.

However, it is possible that the program must be notified when the user modifies the object, in order to carry out certain actions (update drawings, save data in files, launch calculus algorithms, etc.). This will be resolved by events, as reflected in (Figure 18.79). On the other hand, the program can alter the values of certain fields of the object and must notify the changes to the interface (layout) so that it remains updated.

- Use `layout_dbind` to include a `listener` that notifies changes to the application.
- Use `evbind_object` to obtain, within the `callback`, the object that is being edited.
- Use `event_sender` to obtain, within the `callback`, the layout that sent the notification.
- Use `evbind_modify` to know, inside the `callback`, if a field of the object has changed or not.
- Use `layout_dbind_update` to notify the layout that a field of the object has been modified by the application.

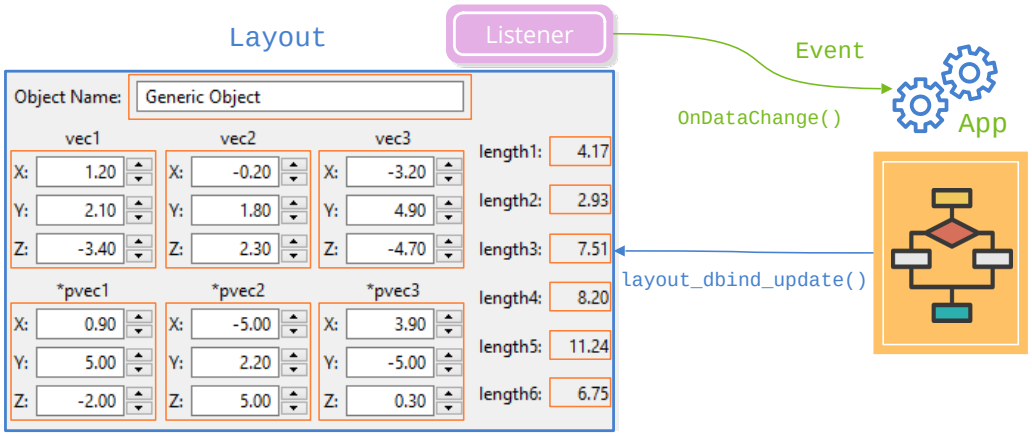


Figure 18.79: Notification of value change to main program.

All of this can be seen in (Listing 18.27). Every time the user changes any `StructTypes` value, a notification of type `ekGUI_EVENT_OBJCHANGE` will be launched that will check if the `vec1` field has changed. If so, its length will be recalculated and the GUI controls associated with that variable will be updated.

Listing 18.27: Notification of object values modification.

```
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);
```

If, for some reason, the modified value is not allowed by the application, it can be reverted by returning `FALSE` as a result of the event (Listing 18.28).

Listing 18.28: Canceling changes made by the user.

```
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;
        }
    }
}
```

18.22. Menu

A **Menu** is nothing more than a container (or window) that integrates a series of options, also called items or **MenuItems** (Figure 18.80). Each one of them have a short text, optionally an icon and optionally also a keyboard shortcut, such as the classic `Ctrl+C/Ctrl+V` to copy and paste. Additionally, an item can house a submenu forming a hierarchy with different levels of depth. In *"Products"* (page 433) you have a sample application that uses menus.

The concept of the menu, like that of the window, exists from the origin of the graphic interfaces. The first computer to incorporate them was the Xerox Alto that appeared in 1973 and its commercial successor the Xerox Star. Concepts still very alive today such as: Menu, Window, Icon, Desk, or Mouse were already present on these computers that served as inspiration to Steve Jobs in the creation of Apple Lisa (Figure 18.81), predecessor of Machintosh and inspirer of Microsoft Windows.

18.23. MenuItem

Represents an option within a *"Menu"* (page 363). They will always have an associated action that will be executed when activated.

- Use `menuitem_create` to create an item.
- Use `menuitem_text` to assign a text.
- Use `menuitem_image` to assign an icon.

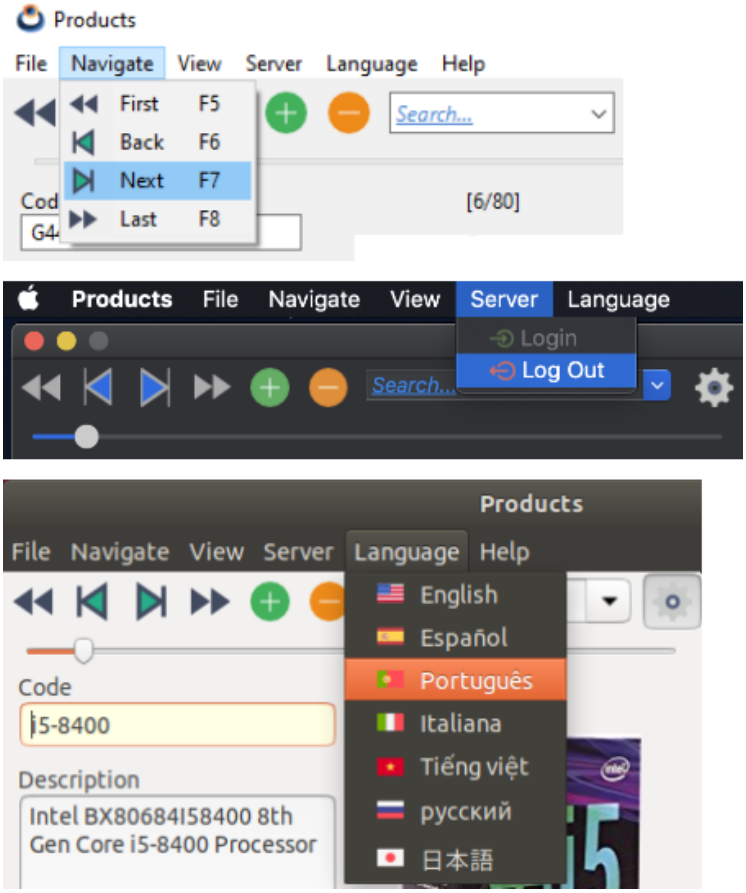


Figure 18.80: Menu bar in Windows, macOS and Linux.

18.24. Common dialogs

Common dialogs are default windows provided by the operating system to perform daily tasks such as: Open files (Figure 18.82), select colors, fonts, etc. Its use is doubly beneficial. On the one hand we avoid programming them as part of the application and, on the other, we take advantage of the user's previous knowledge since they will surely have been used in other programs.

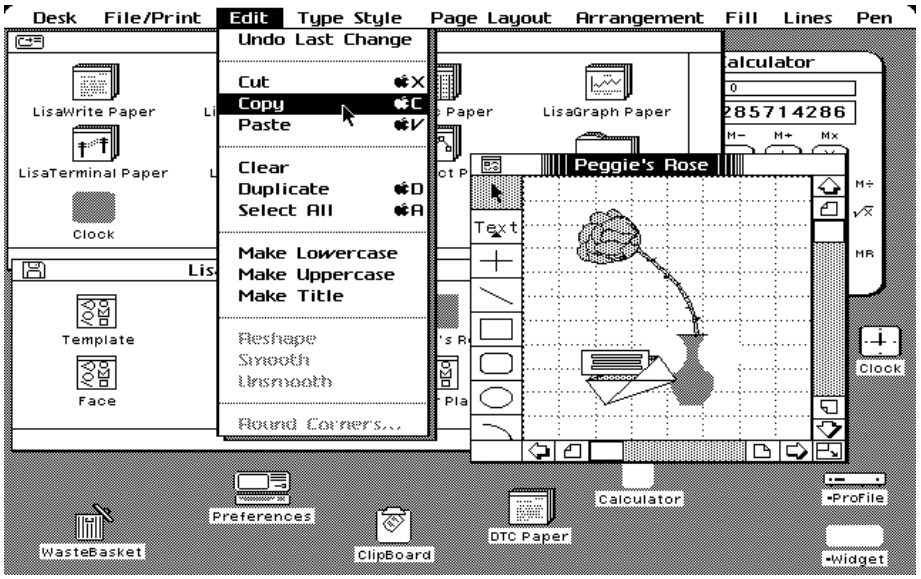


Figure 18.81: The Apple Lisa was one of the first systems to incorporate menus as part of the graphical interface.

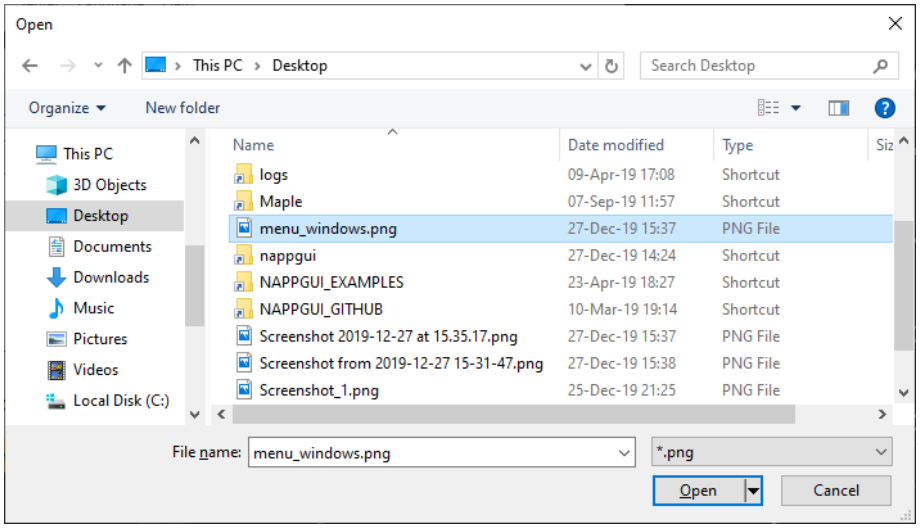


Figure 18.82: File explorer in Windows.

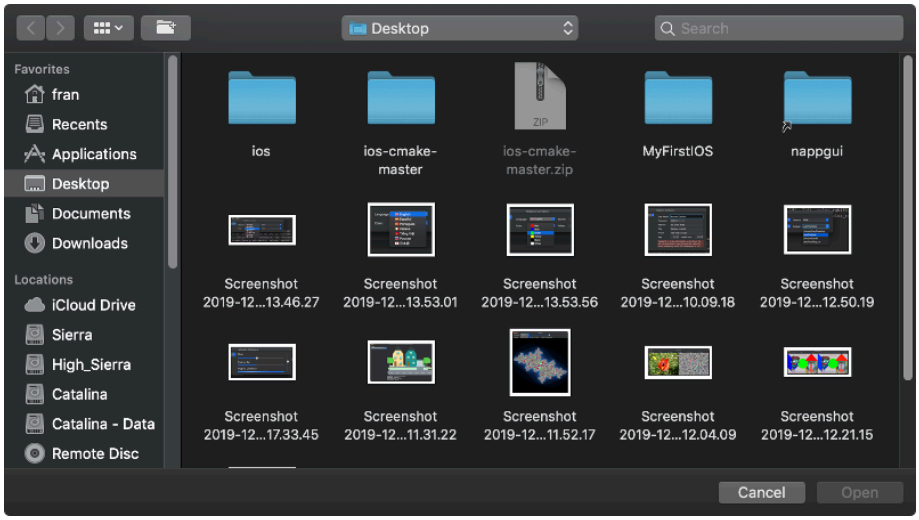


Figure 18.83: File explorer in macOS.

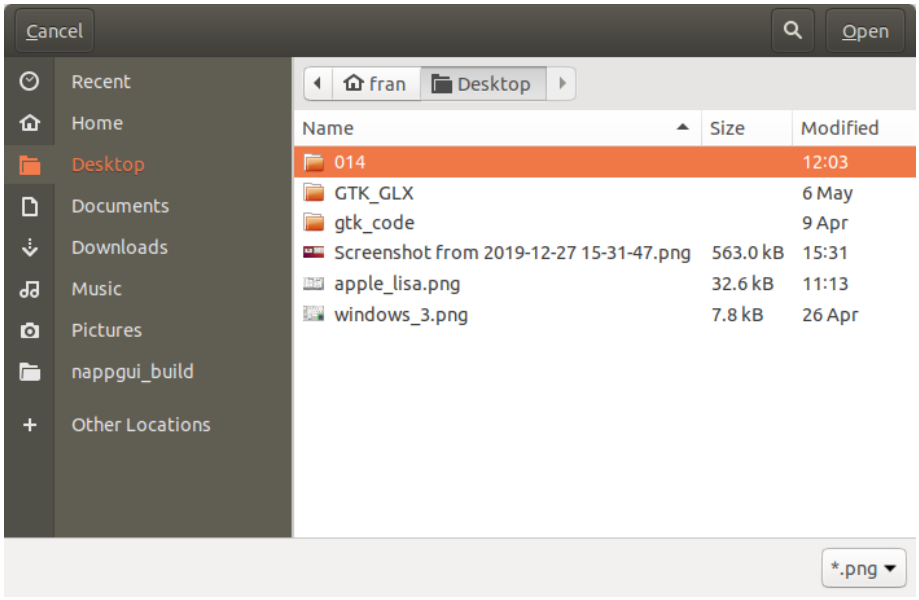


Figure 18.84: File explorer in Linux.

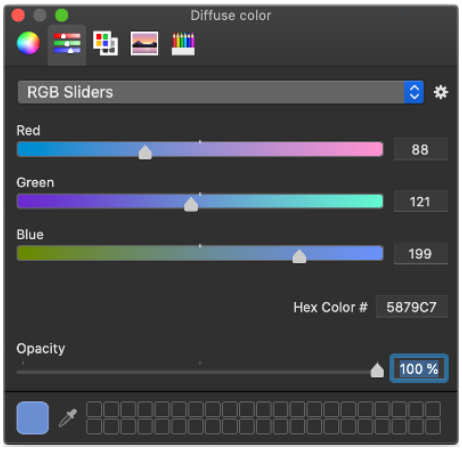


Figure 18.85: Color selection in macOS.

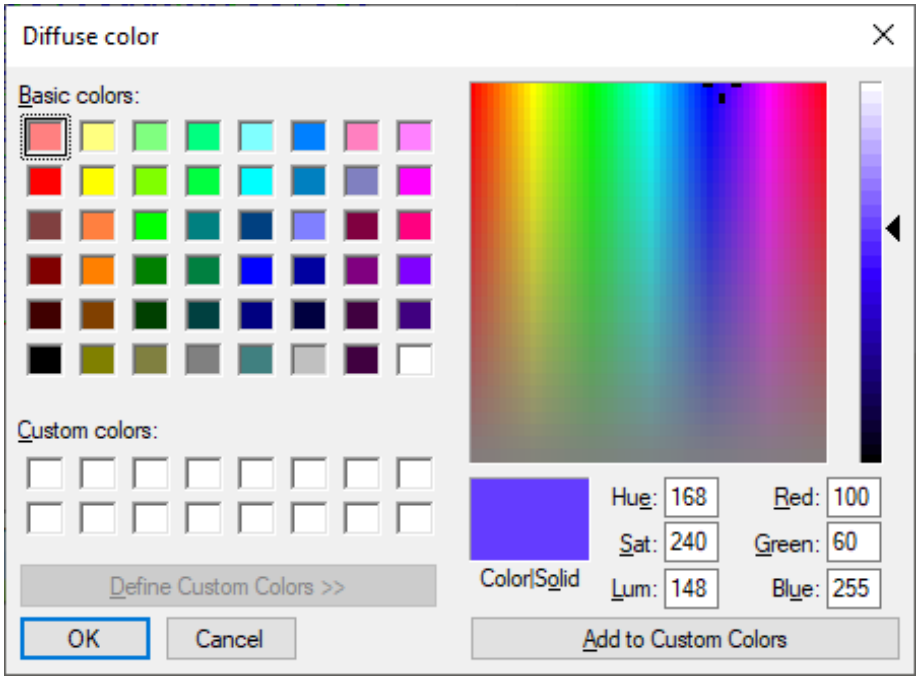


Figure 18.86: Color selection in Windows.

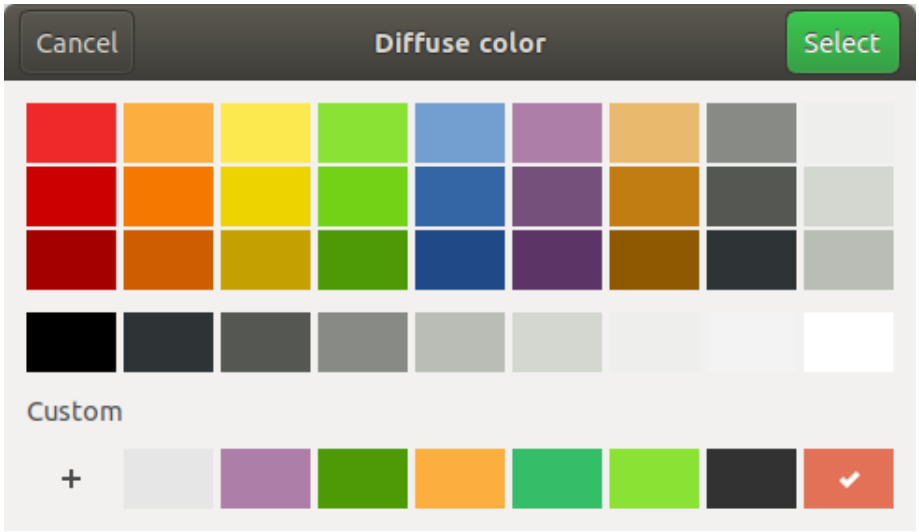


Figure 18.87: Color selection in Linux.

Chapter 19

OSApp library

19.1. OSApp

The *OSApp* library starts and manages the **message cycle** of a desktop application (Figure 19.1). Although the **Gui** library could be integrated into existing applications through a *plugin*, if we want to create an application from scratch, we will need to manage the events that the operating system sends to the program.

- Use `osmain` to start a desktop application.
- Use `osapp_finish` to end a desktop application.

19.2. main() and osmain()

The classic `main` function is the starting point of any C/C++ command line program (Figure 19.2). Its operation does not involve any difficulty and can be summarized in:

- (1) The operating system loads the program into memory and calls the function `main()` to start its execution.
- (2) The sentences are executed sequentially and in the order in which they are written. This order can be altered by means of control sentences (`for`, `if`, `switch`, etc.) or function calls.

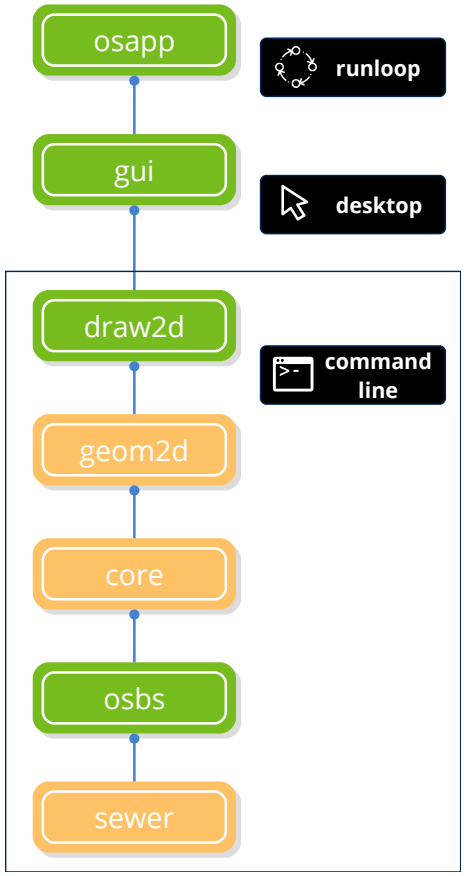


Figure 19.1: *OSApp* dependencies. See *"NAppGUI API"* (page 145).

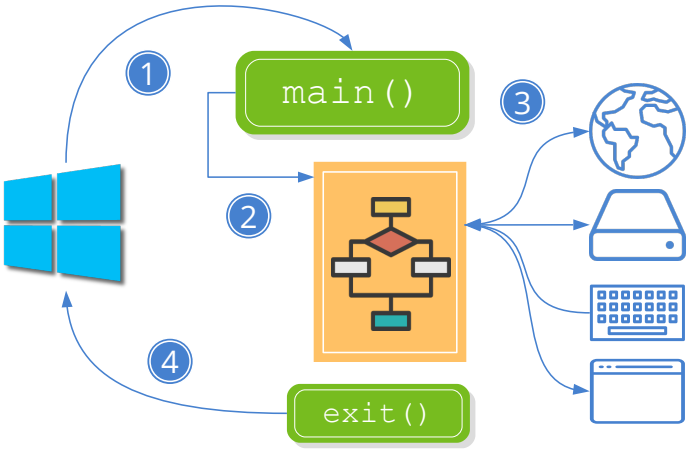


Figure 19.2: Running a console C application.

- (3) If input/output is necessary, the program will wait for the communication to end and continue with the execution.

- 4 When the end of the function is reached `main ()` or an `exit()` sentence is executed, the program will end and the operating system will download it from memory.

However, in desktop applications (event driven), the execution cycle is a bit more complicated. In essence, the program is continuously executing a loop waiting for the user to perform some action (Figure 19.3) (Listing 19.1). In *"Hello World!"* (page 23) you have a simple example:

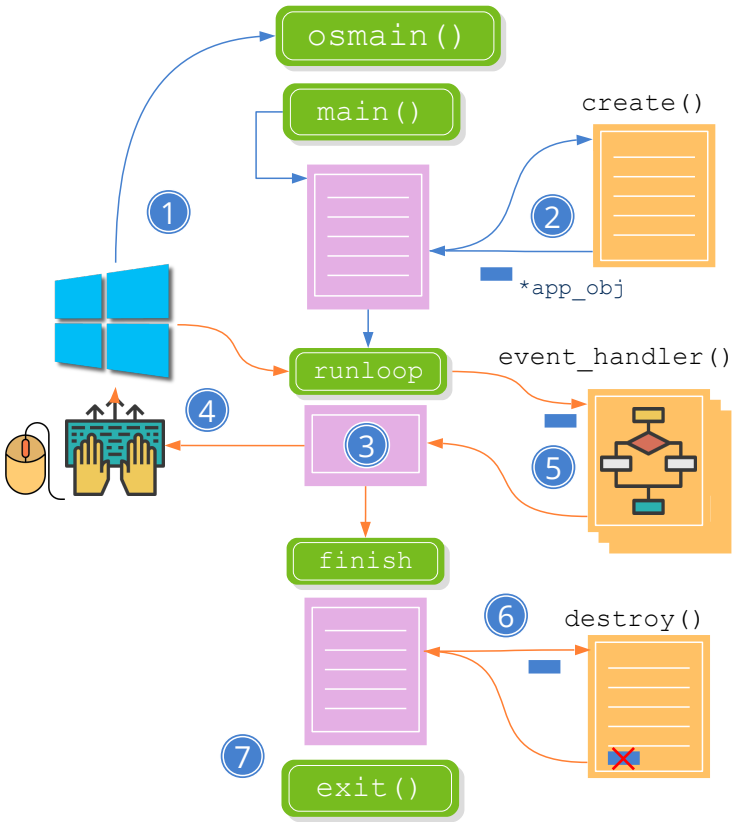


Figure 19.3: Running a desktop C application.

- 1 The operating system loads the program into memory and calls the `main()` function. Now it is encapsulated inside the `osmain` macro which initiates certain structures necessary for event capture and management.

- 2 At some point in this initial process, the application constructor will be called (the first parameter of `osmain()`) that the main object should create. Since the program is continuously returning control to the operating system, the state of the data and windows will be maintained in this object.

- ③ Once initialized, the application will enter a loop known as a **message cycle** (Figure 19.4), while waiting for the user to perform some action on the program interface.

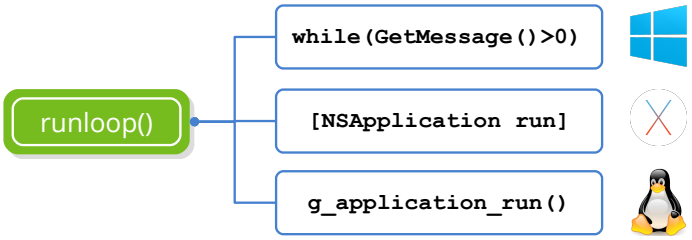


Figure 19.4: Message cycle implementation.

- ④ When this occurs, the operating system will capture the event and send it to the application.
- ⑤ If the application has defined a handle for that event, it will be invoked and the response code will be executed. An application can receive hundreds of messages but will only respond to those it deems necessary, ignoring the rest.
- ⑥ There is a special **exit** event that is generated by calling `osapp_finish`. When this happens, `osmain()` start freeing up resources and preparing a clean exit. At some point the destructor of the application will be called (second parameter of `osmain()`) to do its part of the job, closing possible open files and destroying the main object.
- ⑦ The operating system unload the application from memory.
- The pink blocks are platform dependent and are implemented within NAppGUI.
- The orange blocks are multiplatform (fully portable) and are implemented within the application.

Listing 19.1: Elementary skeleton of a desktop application.

```
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. Synchronous applications

Certain types of applications including video games, media players or simulators, need to be updated at regular intervals, whether or not the user intervenes (Figure 19.5) (Listing 19.2). For these cases we will need a variant of `osmain`, which accepts an update function and a time interval. In *"Bricks"* (page 407) you have an example.

- Use `osmain_sync` to start a synchronous application.

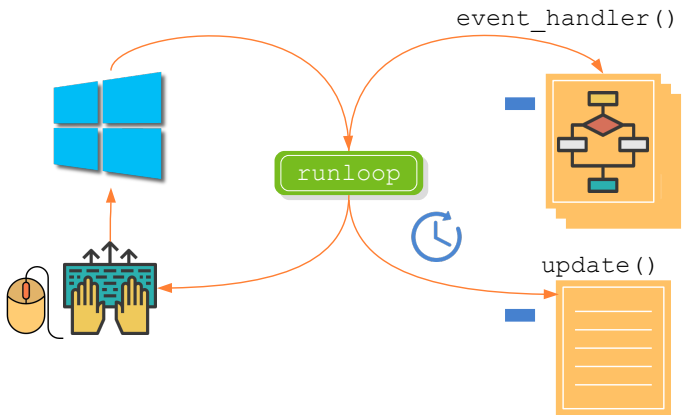


Figure 19.5: Events in synchronous applications.

Listing 19.2: Elemental skeleton of a synchronous application.

```
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. Multi-threaded tasks

Both synchronous and asynchronous applications execute the message cycle on a single CPU thread. This means that if, in response to an event, a relatively slow task must be executed, the application will be "frozen" until it is finished (Figure 19.6)**(a)**. This will produce an unwanted effect since the program will not respond for a few seconds, giving the impression that it has been blocked. The solution is to launch a task in parallel (Figure 19.6)**(b)** (Listing 19.3), quickly release the thread that manages the GUI. In *"Multi-threaded loginMulti-threaded login"* (page 448) you have an example of the use of tasks.

- Use `osapp_task` to launch a new task in a parallel thread.

Listing 19.3: New task in a parallel thread.

```
// 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!
}
```

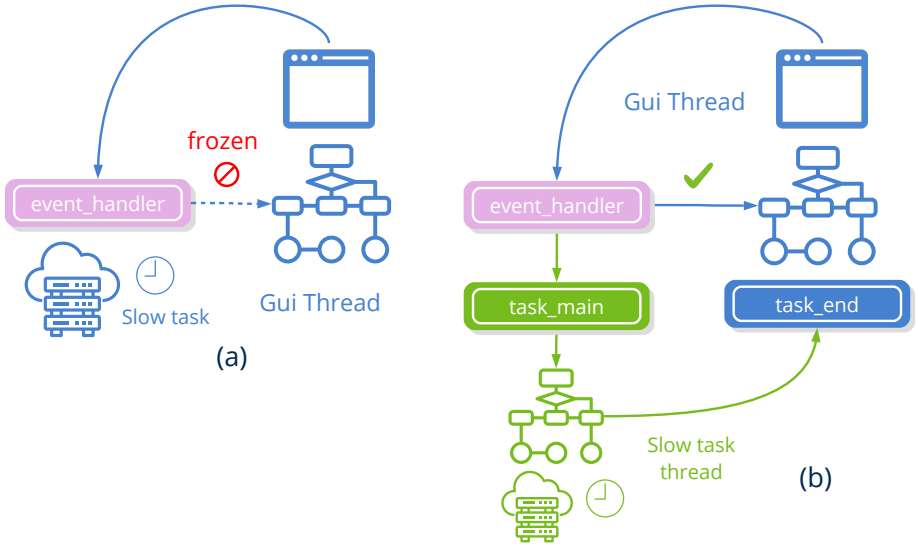


Figure 19.6: (a) Interface lock due to a slow function. (b) Slow function in a parallel thread.

```
osapp_task(tdata, .04, i_task_main, i_task_update, i_task_end, TaskData);
```

The new thread will begin its execution in `task_main`. This function **should not** access the interface elements, just perform calculations or input/output tasks. If it is necessary to update the GUI for the duration of the task (increasing a progress bar or similar), it must be done in `task_update`, indicating in `updtime` the update interval. The new thread will end when it returns from `task_main`, moment to be called `task_end` in the main thread. Obviously, if both threads access shared variables, they must be protected by a `Mutex`.

Chapter 20

INet library

20.1. INet

The **INet** library implements general Internet protocols. Although *"Sockets"* (page 179) allow us to open a communication channel between two remote machines, it is necessary to define a format for the messages that both interlocutors will exchange, in order for communication to be carried out satisfactorily. Any modern operating system provides APIs to use the most popular Internet services, like HTTP. INet accesses this functionality under a common unified and simplified interface (Figure 20.1).

20.2. HTTP

It is common for an application to need information beyond that stored on the computer itself. The simplest and most common way to share information is to store it on a Web Server and publish a URL that provides the desired content (Figure 20.2). This client/server scheme uses the HTTP/HTTPS protocol, which was originally designed to transmit HTML documents between web servers and browsers. Due to the great impact

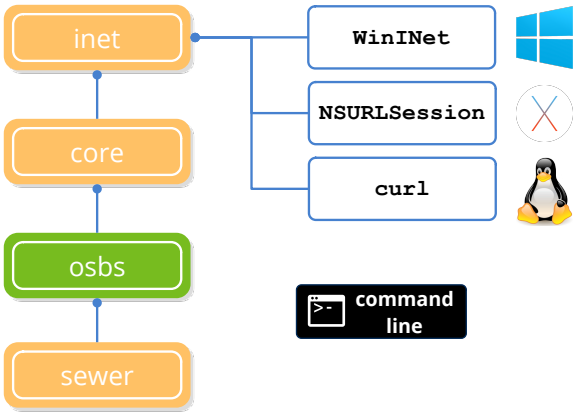


Figure 20.1: *INet* dependencies. See *"NAppGUI API"* (page 145) .

it has had over the years, its use has been expanding for the exchange of structured information between any application that "understands" HTTP. The response from the server will usually be a block of text formatted in JSON or XML.

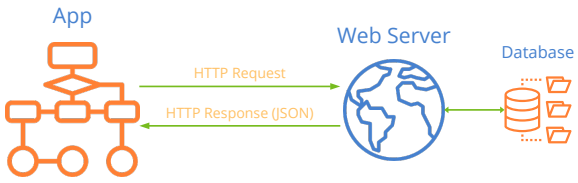


Figure 20.2: Requesting a remote resource using HTTP.

- Use `http_dget` to download a resource from its *"URL"* (page 387) (Listing 20.1).
- Use `http_create` to create an HTTP session.
- Use `http_secure` to create an HTTPS session (encrypted).

Listing 20.1: Direct download of content from a 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);
}
```

On the other hand, if we are going to make successive calls to the same server or if we need more control over the HTTP headers, we must create a session (Listing 20.2).

Listing 20.2: HTTP session.

```
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*, is a data format in text mode that allows to easily represent basic types, objects and arrays. Although its use has become popular in the Web environment, it can also be used for other purposes, such as configuration files or local exchange. Its syntax is easy to understand for humans and simple to process for machines. In (Listing 20.3) we reproduce a small fragment of the JSON response of a Web service:

Listing 20.3: JSON fragment returned by a 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..."
    },
    ...
    ]
}
```

In its structure we can find these data types:

- **Booleans:** Represented by constants `true` or `false`.
- **Numbers:** Use the exponential notation of C for floating-point values: `2.3`, `.76` , `-0.54` or `5.6e12` they are valid examples of numerical values. JSON does not distinguish between integers, negatives or reals.
- **Strings:** Any text in quotes is considered a string. Supports any Unicode character in *"UTF-8UTF-8"* (page 158) or through the escape sequence <c >\uXXXX</c >to indicate the codepoint.
- **Arrays:** Lists of items delimited by brackets `[...]` and separated by commas. The values do not have to be the same type as usually happens in some programming languages (Listing 20.4).

Listing 20.4: JSON array

```
[
    "Red", "Green", "Blue", "Yellow"
]
```

- **Objects:** They are delimited by keys and composed of several fields separated by commas. Each field is formed by an identifier (string) followed by a colon and a value that can be any simple type, object or array (Listing 20.5).

Listing 20.5: JSON object

```
{
    "field1" : true,
    "field2" : 24.67,
    "field3" : "Hello Pibe",
    "field4" : [1, 2, 4, 8.4],
    "field5" : { "x" : 34.32, "y" : -6.19 }
}
```

- **null:** Indicates the absence of value.
- **Binaries:** JSON does not support binary data so opaque objects (images, for example) must be encoded in text and transmitted as a string type value. The most

widespread and globally supported format is the *"Base64"* (page 388) where each character represents 6 bits of information.

NAppGUI's JSON parser automatically transforms `Image` objects to Base64 and vicev-ersa, allowing images to be embedded as data fields.

20.3.1. JSON parsing and conversion to data in C

NAppGUI allows automatic parsing of Json information.

- Use `json_read` to translate a Json to C.
- Use `json_destroy` to destroy a previously read object.

Next we will show different examples with basic types, arrays and objects. In *"Read-/Write Json"* (page 649) you have the complete code. The first step is to create a `Stream` with the content of the Json (Listing 20.6):

Listing 20.6: Create a Stream with Json data.

```
/* 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));
```

The Stream should be destroyed with `stm_close` at the end of the analysis.

Later we will use `json_read` indicating the expected data type of the Json.

Listing 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);
```

Listing 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);
```

Listing 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);
```

Listing 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);
```

Listing 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));
```

Listing 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));
```

For the analysis of objects it is necessary that we register with *"Data binding"* (page 225) their structure, in such a way that the types and names of the fields of the Json object coincide with the `struct` from C. Given this Json:

Listing 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
        }
    ]
}
```

We define these `structs` and register them:

Listing 20.14: Structures that will hold the data of the 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);

dbind(Product, String*, description);
dbind(Product, real32_t, price);
dbind(Products, uint32_t, size);
dbind(Products, ArrSt(Product)*, data);
```

This way we can now call `json_read`:

Listing 20.15: Reading the Json object.

```
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()** ignores (skips) those fields of Json objects that are not registered with **dbind**. In no case will they generate caches or dynamic memory.*

20.3.2. Mapping between Json and C

`json_read` recognizes the basic NAppGUI types, as well as `String`, `Image`, `ArrSt`, and `ArrPt`. **Will not work with other data types** such as `int` or `float`. It will also not recognize the STL structures `vector`, `map`, etc. In (Table 20.1) we show the equivalence between the fields of a Json and the C types that we need to map it correctly.

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/4QB4RXhpZgAASUkqAAg
array	`ArrSt`(uint16_t)	[12, 111, 865]
array	`ArrSt`(real32_t)	[-34.89, 0.0001, 567.45, 1e6
array	`ArrPt`(String)	[“red”, “green”, “blue”]
array	`ArrPt`(Image)	[“/9j/4QB4RXh...”, “/9j/4QB4RX
object	`struct Product` (*“Data binding”* (page 225))	{ “description” : “i7-8700K”, “ ”price
array	`ArrSt`(Product)	[{ “description” : “i7-8700K”, “ ”price“
array	`ArrPt`(Product)	[{ “description” : “i7-8700K”, “ ”price“

Table 20.1: Equivalence between Json and NAppGUI types.

20.3.3. Convert from C to JSON

- Use `json_write` to write data/objects from C to Json.

Based again on (Table 20.1), let’s do the reverse process and generate Json data from C types and objects. First, create a write stream to hold the result (Listing 20.16):

Listing 20.16: Create a write Stream.

```
/* Write stream in memory */
Stream *stm = stm_memory(2048);

/* Write stream in disk */
Stream *stm = stm_to_file("/home/fran/appdata/products.json", NULL);
```

The Stream should be destroyed with `stm_close` when it is no longer needed.

Later we will use `json_write` indicating the expected data type of the Json.

Listing 20.17: Write boolean to 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
```

Listing 20.18: Write integer to 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
```

Listing 20.19: Write String to 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"
```

Listing 20.20: Write Image to 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: "iVBORw0KGgoAAAANSUhEUgAAAIA..."
```

Listing 20.21: Write ArrSt(int16_t) to 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 ]
```

Listing 20.22: Write ArrPt(String) to 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" ]
```

Listing 20.23: Write Products object to 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 is the acronym for *Uniform Resource Locator* that identifies a unique resource on the Internet. The most common use is found when making requests to a Web server. For example `https://www.google.com` is a widely recognized and used URL. Being somewhat more specific, we can say that it is a string of characters with a specific format composed of a series of fields that allow unambiguously locating a unique global resource (Listing 20.24) (Figure 20.3).

Listing 20.24: Parsing a URL string.

```
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:** Communication protocol used. **http**, **https**, **ftp**, **smtp**, **mailto**, etc.
- **Authority:** Access string to the server composed of several fields, where only the host name is required. The rest are optional.
  - **Host:** Server name or IP address.
  - **User:** User name. Optional, only if the service requires it.
  - **Password:** Password. Optional, only if the service requires it.
  - **Port:** Access port. Each protocol has a default port, which will be the one used if none is specified. 80 = http, 413 = https.
- **Resource:** Path within the server where the resource we are looking for is located. The *pathname* is the only one required.
  - **Pathname:** Directory and name of the file or resource.
  - **Parameters:** List of `name = value` arguments that the service may need. Not normally used. If there are multiple values, they are separated by the character `'&'`.

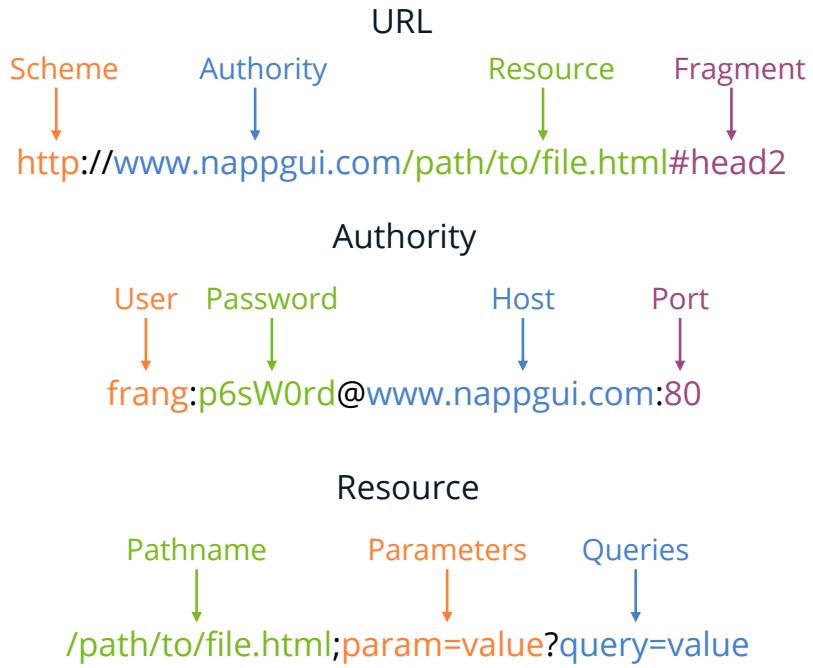


Figure 20.3: The different fields that make up a URL.

- **Queries:** List of `name = value` arguments that the service may need. These are the ones normally used by Web services. That is, in the URL you must use the `'?'` separator instead of `';'` after the *pathname*. If there are multiple values, they are separated by the character `'&'`.

- **Fragment:** It is an anchor to a specific part of the document that we request from the server. Normally used to access a specific point in an HTML page.

20.5. Base64

Part 3

Sample Applications

Chapter 21

Die

Beautiful code is likely to be simple – clear and easy to understand. Beautitful code is likely to be compact – just enough code to do the job and no more – but not cryptic, to the point where it cannot be understood. Beautiful code may well be general, solving a broad class of problems in a uniform way. One might even describe it as elegant, showing good taste and refinement.

Brian Kernighan

As the road is made by walking, we will devote a few chapters to deepen the use of NAppGUI hand in hand with real applications. Our goal is to present programs of a certain level, halfway between the simple "book examples" and the commercial applications. In this first demo we have a program that allows us to draw the silhouette of a die (Figure 21.1) and that will serve as an excuse to introduce concepts of parametric drawing, composition of *layouts* and use of resources. The **source code** is in folder `/src/demo/die` of the SDK distribution. In *"Create new application"* (page 99) and *"Resources"* (page 129) we saw how to create the project from scratch.

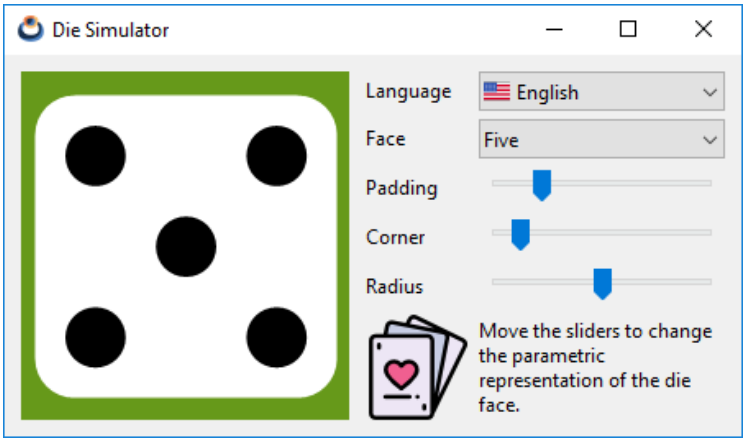


Figure 21.1: *Die Simulator* application, Windows version. Inspired by DieView *(Cocoa Programming for OSX, Hillegass et al.)*

Figure 21.2: MacOS version.

21.1. Use of sublayouts

We started working on the user interface, which we have divided into two areas: a customized view (`View`) where we will draw the representation of the die in 2D, and a zone of controls where we can interact with this drawing. As we already saw in *"Hello World!"* (page 23) we will use `Layout` objects to locate the controls inside the main window. However, we observe that this arrangement of elements does not fit well in a single table, therefore, we will use two horizontal cells as the main container and a *grid* of two columns and six rows for the controls (Listing 21.1) (Listing 21.1). This second layout will be located in the right cell of the first container and we will say that it is a **sublayout** of the main layout.

Figure 21.3: Linux/GTK+ version.

Listing 21.1: Composition through 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);
```

In the same way that we did in *"Layout formatLayout format"* (page 29) we have established certain margins and a fixed width for the controls column.

Listing 21.2: Layout format

```
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);
```

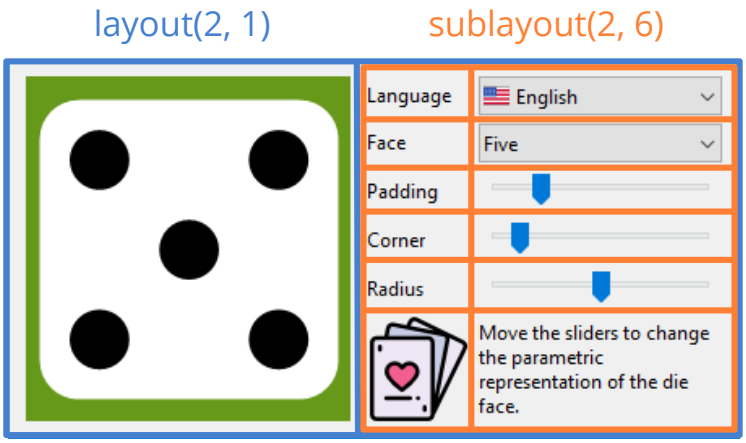


Figure 21.4: The use of sublayouts adds flexibility when designing the *gui* .

```
layout_vmargin(layout1, 3, 5.f);
layout_vmargin(layout1, 4, 5.f);
```

21.2. Use of Custom Views

`View` are controls that will allow us to design our own *widgets*. On the contrary that happens with another type of components, like *"Slider"* (page 314) or *"Button"* (page 306), here we will have total freedom to draw anything. We can interact with the control by capturing its events (mouse, keyboard, etc) and implementing the appropriate handlers. These views are integrated into the layout like any other component (Listing 21.3).

Listing 21.3: Creating a custom view.

```
View *view = view_create();
view_size(view, s2df(200.f, 200.f));
layout_view(layout, view, 0, 0);
```

We can not draw inside a `View` whenever we want. We will have to make a request to the operating system through the method `view_update` (Listing 21.4), since the drawing area can affect overlapping windows and this must be managed centrally. When the control is ready to refresh, the system will send an event `EvDraw` that we must capture through `view_OnDraw`.

Listing 21.4: Code basic of View refresh.

```
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));
```

Each time the user moves a slider (padding parameter, for example) the operating system captures the action and informs the application through the method `i_OnPadding` (Figure 21.5). Because the action involves a change in the drawing, this method calls `view_update` to inform the system again that the view must be updated. When it considers it appropriate, send the event `EvDraw`, which is captured by `i_OnDraw` where the drawing is regenerated with the new parameters.

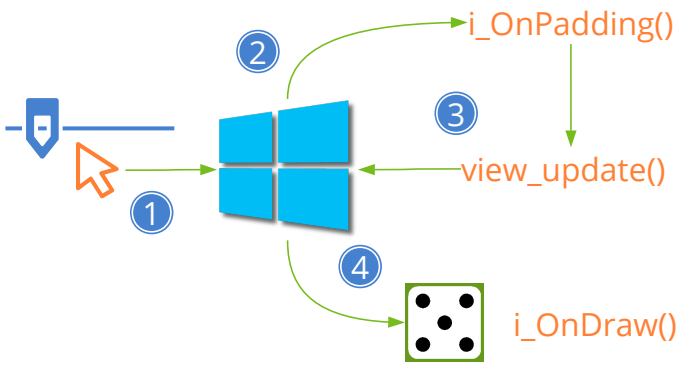


Figure 21.5: Understanding the event flow in interactive drawings.

21.3. Parametric drawing

Under this concept we describe the ability to generate vector images from a few numerical values known as parameters (Figure 21.6). It is used a lot in the computer-aided design (CAD), it allows you to make adjustments easily in planes or models without having to edit, one by one, a lot of primitives.

In our application, the representation of the die can change at runtime as the user manipulates the sliders or sizes the window, so we calculate the position and size of their primitives using parametric formulas. Once resolved, we created the drawing with three simple API commands *"Drawing primitives"* (page 265).

- `draw_clear`. Clear the entire drawing area using a solid color.

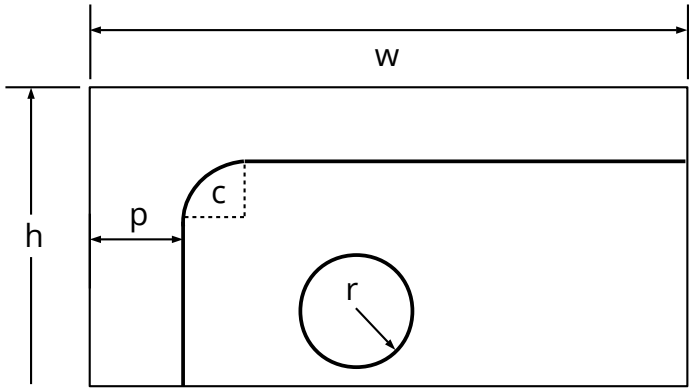


Figure 21.6: Principles of parametric drawing, applied in Die.

- `draw_rndrect`. Draw a rectangle with rounded corners.
- `draw_circle`. Draw a circle.

Listing 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);
    }
}
```

The drawing commands are reflected on a canvas, also known as context `DCtx`. This object reaches to `i_OnDraw` as parameter of the event `EvDraw`. In this case, the canvas is provided by the `View` control itself, but it is also possible to create contexts to draw directly in memory.

21.4. Resizing

In this application, the window can be resized by stretching the cursor over its edges, which is common in desktop programs. Let's see some basic aspects about this feature not present in *"Hello World!"* (page 23), which had a static window. The first thing is to enable the option inside the window's constructor.

```
window_create(ekWINDOW_STDRES, &panel);
```

When a window changes in size, the inner controls should do so proportionally as well as

change its location within the panel. This management is carried out within each `Layout` object. When the window starts, the default size of each layout is calculated by applying the **natural sizing**, which is the result of the initial size of the controls plus the margins, as we saw in *"Layout formatLayout format"* (page 29). When we stretch or contract the window, the pixel difference between natural and real dimensioning is distributed between the columns of the layout (Figure 21.7). The same happens with the vertical difference, which is distributed among its rows. If a cell contains a sublayout, this increment will be recursively distributed by its own columns and rows.

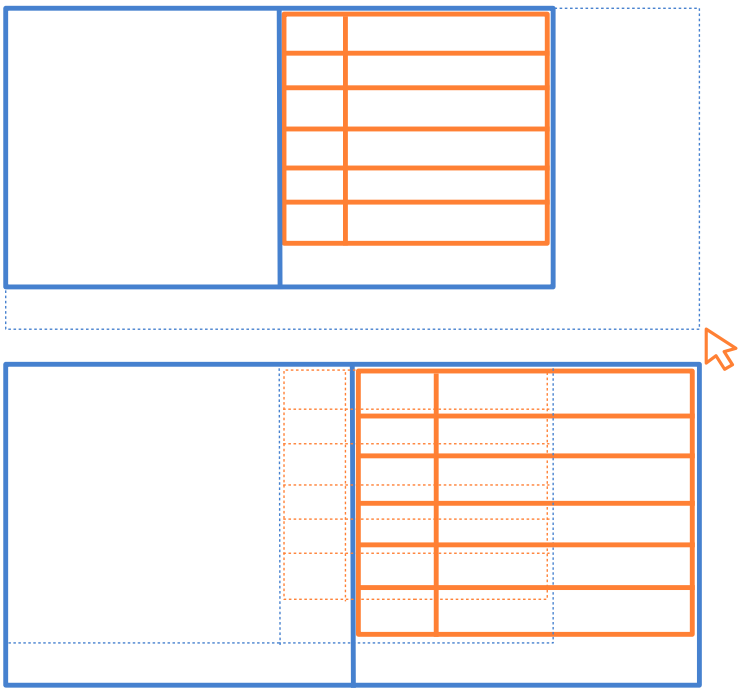


Figure 21.7: When resizing, the excess of pixels is distributed proportionally by the rows and columns of the `Layout`.

But in this particular case, we want the whole increment to go to the drawing area (column 0). In other words, we want the column of the controls to remain fixed and not grow (Figure 21.8). For this we must change the proportion of the resized:

```
layout_hexpand(layout, 0);
```

With this function 100% of the horizontal surplus will go to column 0. By default, they had a proportion of (50%, 50%) since they are two columns (33% for three, 25% for four, etc). With this we would have resolved the resizing for the X dimension of the window, but what happens with the vertical? In the main layout, we only have one row that, when expanded, will change the height of the custom view. But this expansion will also affect the

cell on the right, where the controls will also grow vertically due to the recursive increase of pixels in the sublayout rows. To solve it, we force the vertical alignment `ekTOP` in the right cell of the layout.

```
layout_valign(layout, 1, 0, ekTOP);
```

instead of `ekJUSTIFY`, which is the default alignment for sublayouts. In this way, the content of the cell (the entire sublayout) will not expand vertically, but it will adjust to the upper edge leaving all the free space in the lower part of the cell. Obviously, if we use `ekCENTER` or `ekBOTTOM`, the sublayout will center or adjust to the bottom edge.

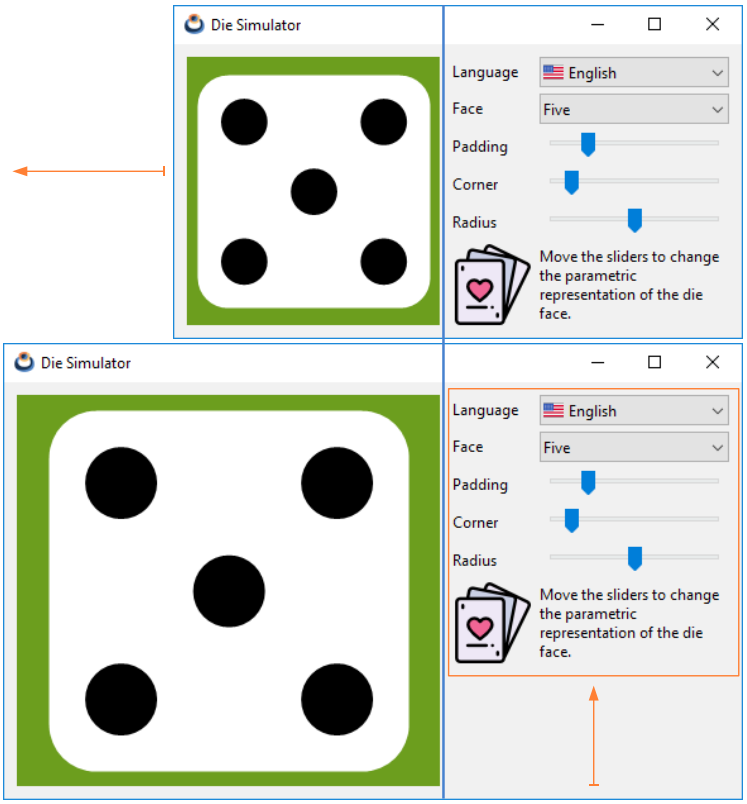


Figure 21.8: Playing with the horizontal ratio and vertical alignment, only the drawing area will be affected by the size changes.

21.5. Use of resources

Both the text and the icons that we have used in *Die* have been outsourced in the resource package `all`. Thanks to this, we can perform an automatic translation of the interface between the English and Spanish languages. You can check *"Resources"* (page 129)

to get detailed information on how text and images have been assigned in the program interface.

Listing 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
```

Listing 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 and Dice

This application has been used as a guiding thread of the *"Create new application"* (page 99) chapter and following from the NAppGUI tutorial. The complete example consists of two applications (**Die** and **Dice**), as well as the **casino** library that groups the

common routines for both programs (Figure 21.9). You have the three complete projects ready to compile and test in the folder `src/demo` of SDK distribution.

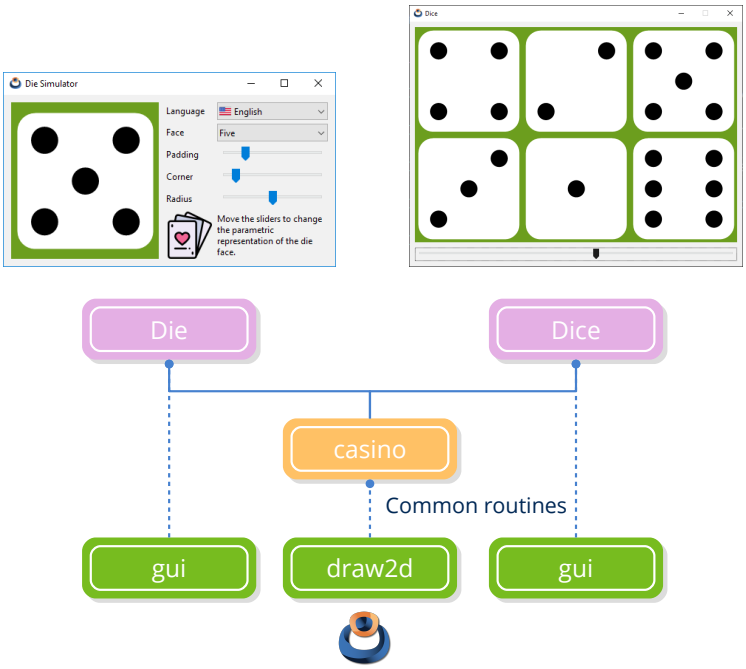


Figure 21.9: Common routines for both applications are shared through the **casino** library.

21.7. The complete Die program

Listing 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
```

Listing 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)
```

Listing 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, gui_image(USA_PNG));
    popup_add_elem(popup1, TEXT_SPANISH, gui_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;
```

```
    }
}
```

Listing 21.11: demo/die/dgui.h

```
/* Die Gui */

#include "die.hxx"

__EXTERN_C

Window *dgui_window(App *app);

__END_C
```

CHAPTER 22

Bricks

Briks is a very simplistic imitation of the **Atari Breakout** video game, which will allow us to make an introduction to the world of *"Synchronous applicationsSynchronous applications"* (page 373). Any real-time application must be constantly updating whether or not the user intervenes. The **source code** is in folder `/src/demo/bricks` of the SDK distribution.

- Use `osmain_sync` to start a synchronous application, indicating an interval and update *callback* function. NAppGUI will periodically launch time events that will update the program.

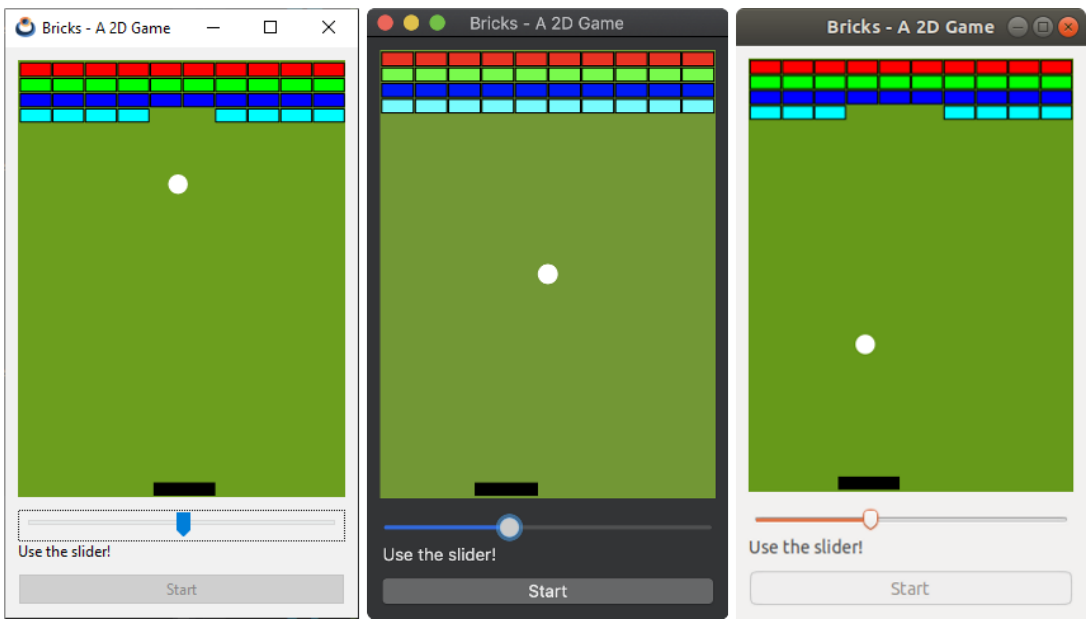


Figure 22.1: Bricks video game on Windows, macOS and Linux.

This application is managed by two events (Figure 22.2). On the one hand the slider movement, which can occur at any time (asynchronous event), and will update the player position. On the other a synchronous event produced by `osmain_sync` every 40 milliseconds and will be notified through `i_update()` to update the game state and graphic view.

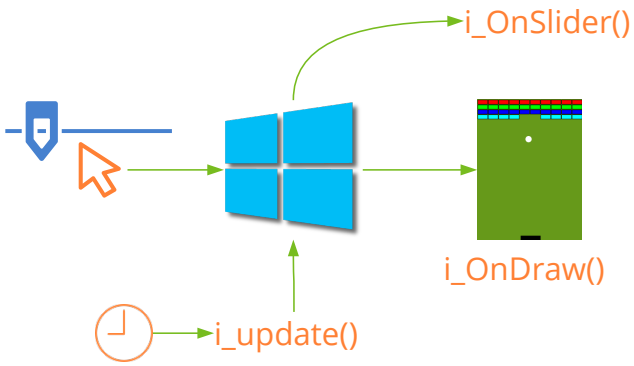


Figure 22.2: Synchronous and asynchronous events.

Listing 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)
```

Chapter 23

Fractals

In this application we create an procedural image by calculating the color of each pixel using fractal algorithms . Some of the most fascinating results produced by a dynamic system occur when we iterate a complex variable function instead of a real one. This is the case of **Julia's sets**. The **source code** is in folder `/src/demo/fractals` of the SDK distribution.

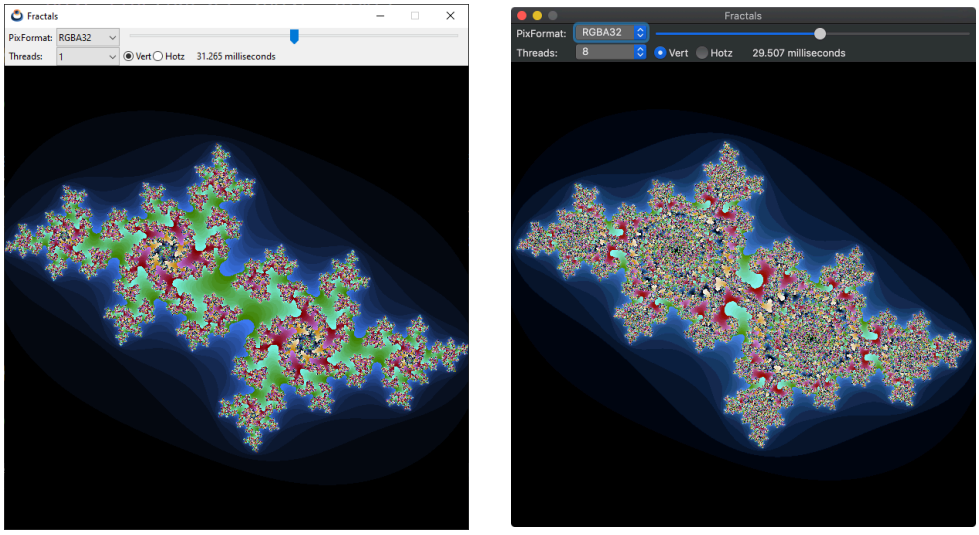


Figure 23.1: **Fractals** application Windows and macOS version.

Due to the large computational load of this algorithm we have divided the calculation into several threads (Figure 23.3). This problem is easily parallelizable simply by fractioning the image, because each pixel is obtained independently.

Listing 23.1: demo/fractals/fractals.c

```
/* Multi-threaded fractals */
```

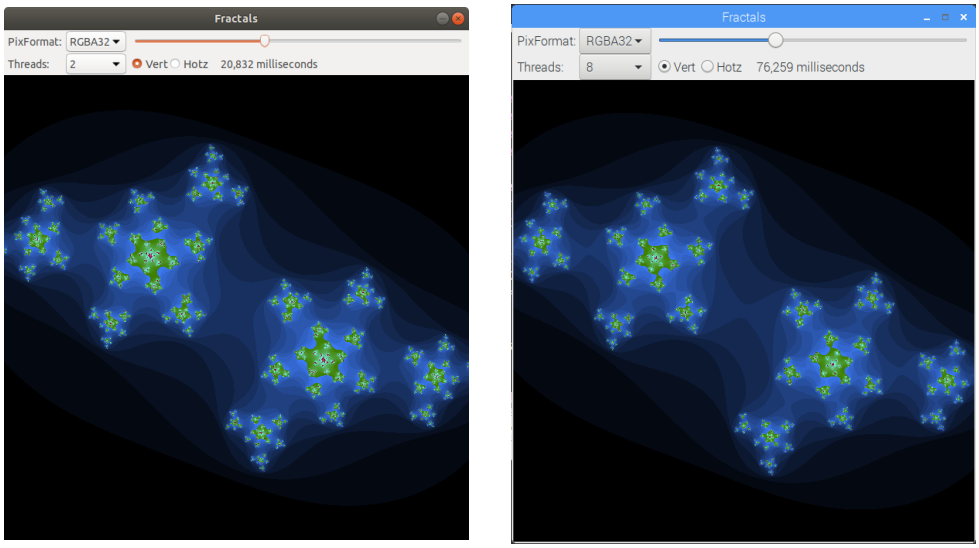


Figure 23.2: Ubuntu and Raspbian version.

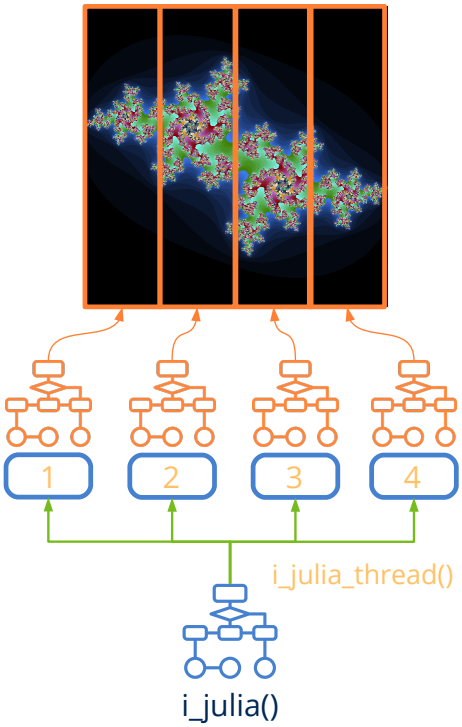


Figure 23.3: Collaboration of several threads.

```
#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)
```

Chapter 24

Bode

In this project we approach the construction of an interactive user interface for **Bode Plots**, a tool widely used in Control Engineering (Figure 24.1). The calculus module has been written in C language by Javier Gil Chica[1], Phd of Physics Department of the University of Alicante. The complete source code is available in folder `/src/demo/bode` of the SDK distribution.

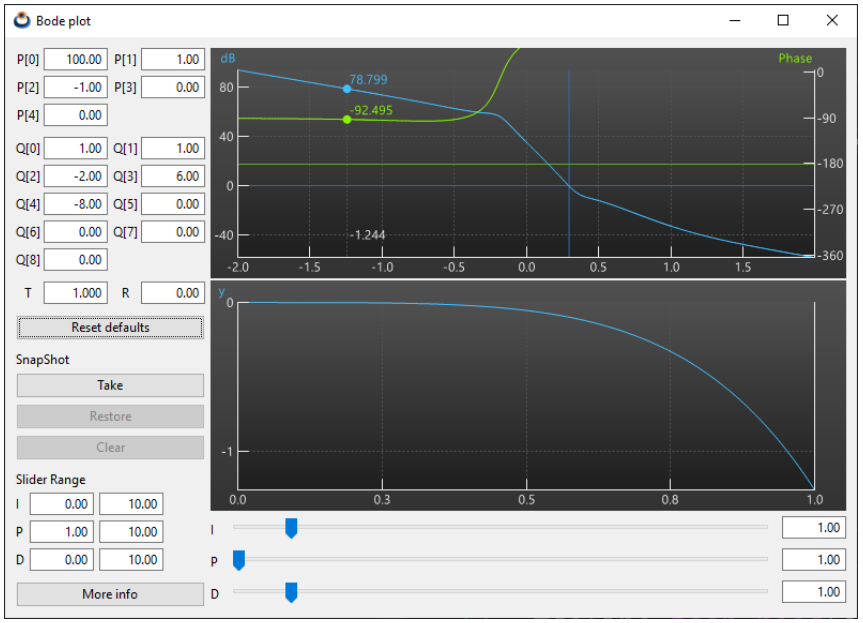


Figure 24.1: Windows version.

The main window has been divided vertically into two parts, using a `layout(2,1)` (Figure 24.4). On the left side we have the parameters `P`, `Q`, `T`, `R` and some buttons.

[1] `mailto:francisco.gil@ua.es`

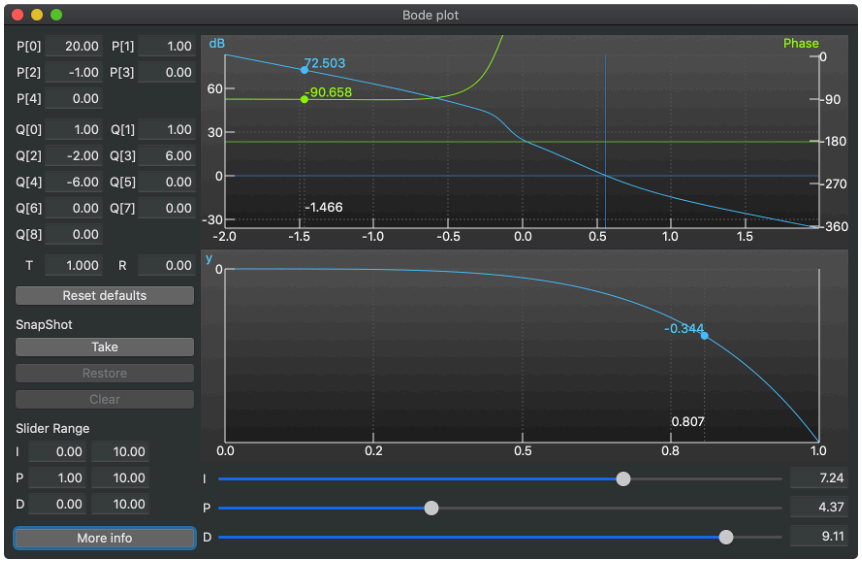


Figure 24.2: macOS version.

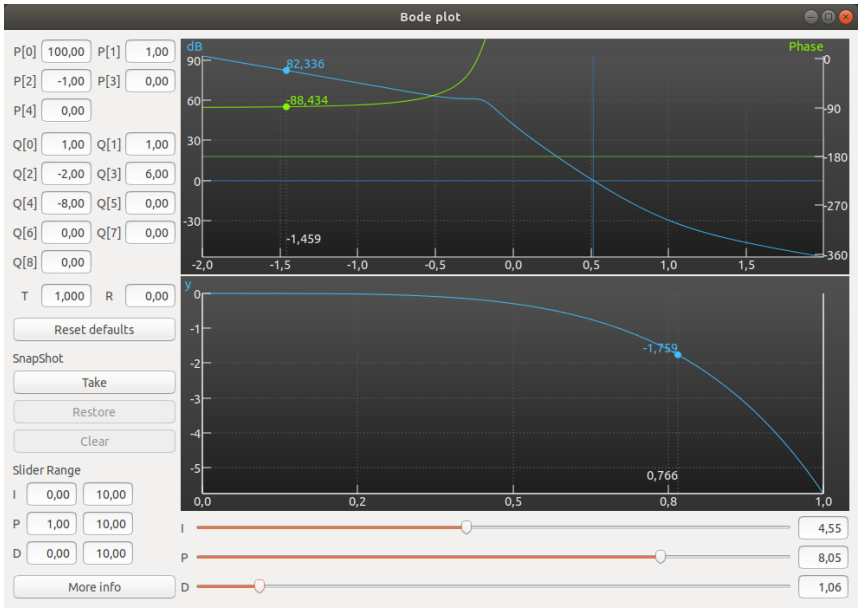


Figure 24.3: Ubuntu version.

Sublayouts have been used `i_coeffs(4,9)` and `i_ranges(3,3)` to group controls. In the right area are two `View` drawing controls for graphics and other sublayout `i_sliders(3,3)` with the parameters `I, P, D`.

The horizontal resizing is done entirely on the right cell (graphs and sliders), keeping

the parameter area a constant horizontal size. During the vertical resizing the graphs will grow with a proportion of 50% each. For the left part, an empty cell has been reserved, which will expand horizontally, aligning the button `[More Info]` to the bottom edge of the window.

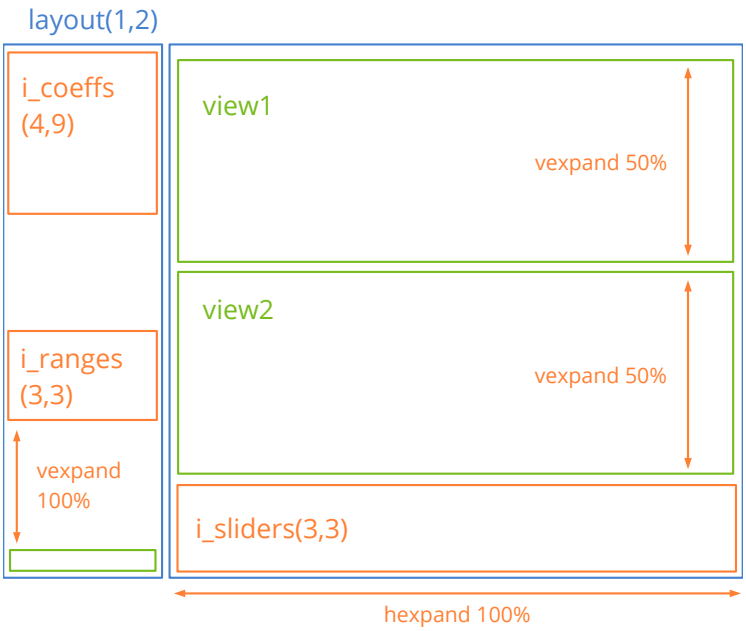


Figure 24.4: Bode user interface distribution.

Listing 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;
}
```

Chapter 25

Products

In this project we will face the construction of an application that allows browsing through a database of products obtained from a Web server (Figure 25.1). This client-server pattern is widely used today, so we will have a stable base to create any application based on this model. The **source code** is in folder `/src/demo/products` of the SDK distribution.

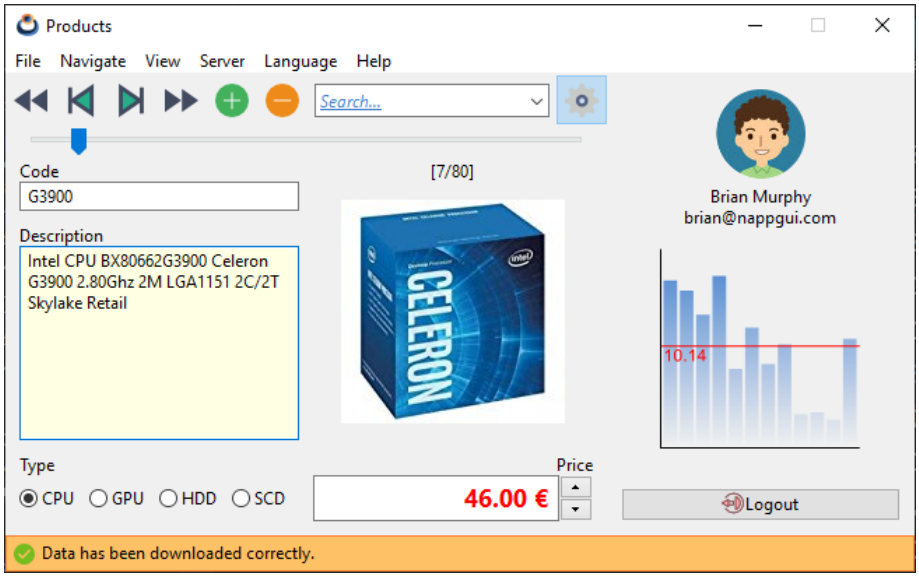


Figure 25.1: Application *Products*, Windows version.

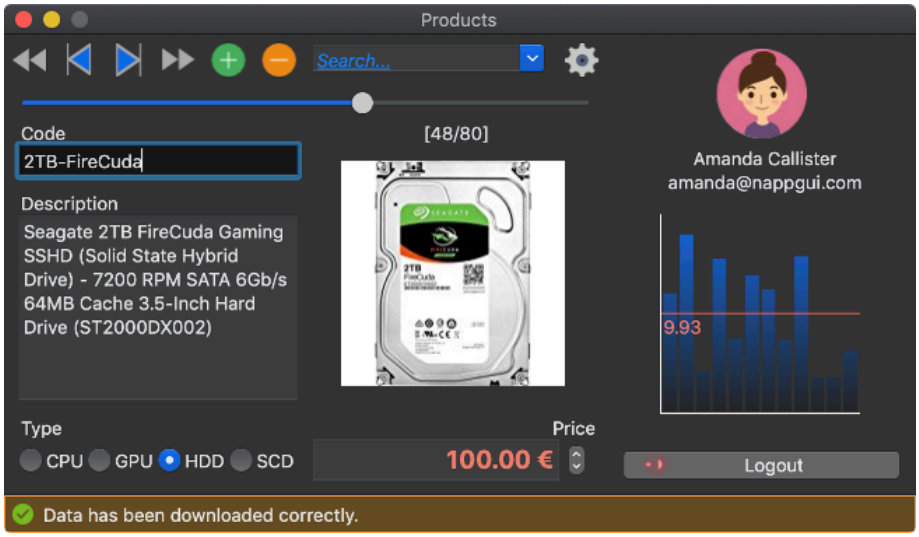


Figure 25.2: macOS version.

25.1. Specifications

- The database is remote and we will access it through Web services that will encapsulate the data in JSON. To obtain the products we will use this service[1] and to

[1] `http://serv.nappgui.com/dproducts.php`

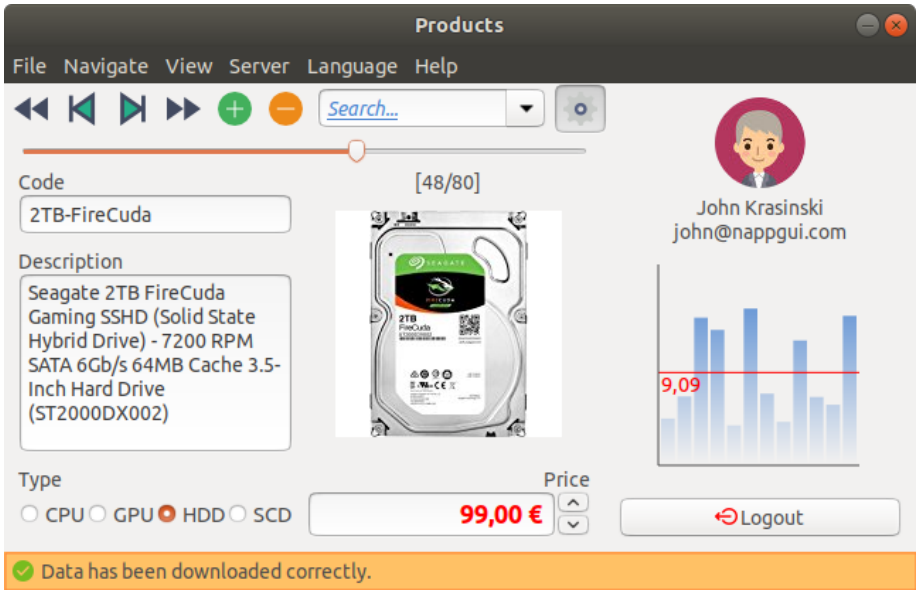


Figure 25.3: Linux/GTK+ version.

register a user this other[2]. We have four **users** registered in our database: *amanda*, *brenda*, *brian* and *john* all with **password** 1234.

- The remote database is read-only. We do not have web services to edit it.
- The moment a user registers, all articles will automatically be downloaded.
- A small graph with the sales statistics of each product will be displayed.
- You can edit the database locally, as well as add or delete records.
- You can export the local database to disk, as well as import it.
- We will have the typical navigation controls: First, last, next, previous.
- We can establish a filter by description. Only those products whose description matches partially with the filter will be displayed.
- The interface will be in seven languages: English, Spanish, Portuguese, Italian, Vietnamese, Russian and Japanese. We can change the language without closing the application.
- The application must run on Windows, macOS and Linux.

[2] `http://serv.nappgui.com/duser.php?user=amanda&pass=1234`

25.2. Model-View-Controller

Since this program has a medium level of complexity, we will fragment it into three parts using the well-known pattern model-view-controller **MVC** (Figure 25.4).

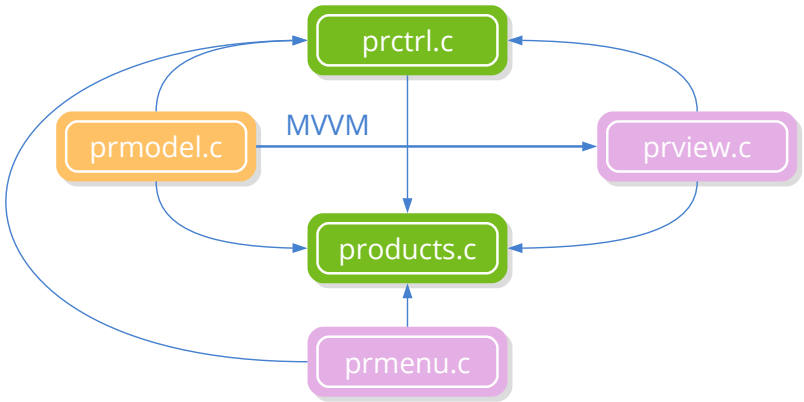


Figure 25.4: MVC modules that make up the application.

- **Model:** It will deal with the data itself, the connection with the server and the reading/writing on disk. It will be implemented in `prmodel.c`.
- **View:** Here we will implement the data presentation layer, composed of the main window (in `prview.c`) and the menu bar (in `prmenu.c`).
- **Controller:** Will take care of the logic of the program `prctrl.c`. It will respond to user events and maintain consistency between the model and the view. Due to the amount of extra work involved in synchronizing each field of the structure with the interface controls, we will use the pattern *Model-View-ViewModel* **MVVM** where the model data will be automatically synchronized with the interface and the I/O channels.
- **Main:** module `products.c`. It contains the function `osmain` and load the three previous actors.

25.3. Model

The data model of this application is quite simple (Listing 25.1), since it only requires manipulating an array of structures of type `Product`.

Listing 25.1: Structures that make up the data model.

```
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;
};
```

As a previous step, we will register the model structures which will allow us to automate I/O tasks without having to explicitly coding them thanks to *"Data binding"* (page 225) (Listing 25.2).

Listing 25.2: Registration of data model `struct` fields.

```
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

We will get the articles data from the Web server in two steps. On the one hand we will download a `Stream` with the JSON using HTTP and, later, we will parse it to a C object (Listing 25.3).

Listing 25.3: JSON data download and processing.

```
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);
        ...
```

The JSON of this web service[3] consists of a header and a list of products (Listing 25.4), so we must register a new structure in order to `json_read` can create the object correctly (Listing 25.5). Note that JSON-C pairing is carried out by the field name, so these must be identical (Figure 25.5).

Listing 25.4: Web service format.

```
{
    "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....
    },
    ...
}
```

Listing 25.5: JSON header registration.

```
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. Write/Read on disk

Serialization (Listing 25.6) and de-serialization (Listing 25.7) of objects using binary streams can also be performed automatically simply by registering the data types (Figure 25.6). We do not need to explicitly program reading and writing class methods.

[3] http://serv.nappgui.com/dproducts.php

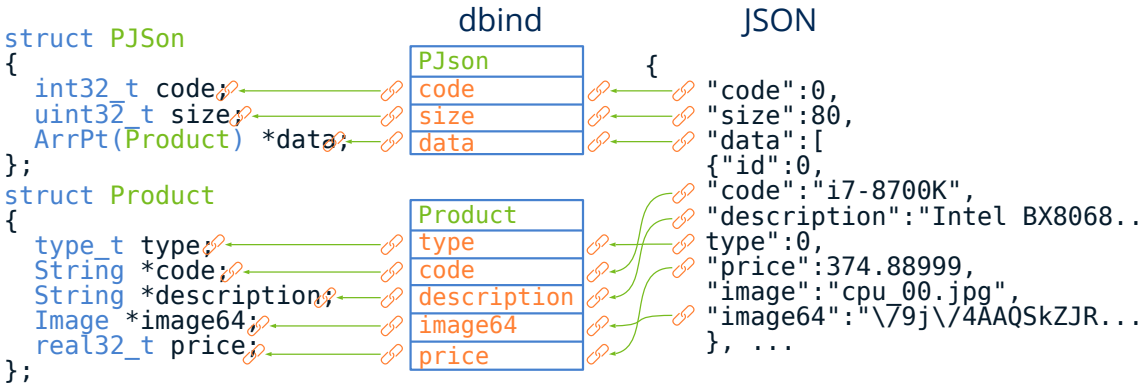


Figure 25.5: `json_read` access `dbind` registry to create a C object from a JSON stream.

Listing 25.6: Export of the database to disk.

```
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;
}
```

Listing 25.7: Importing the database from disk.

```
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;
}
```

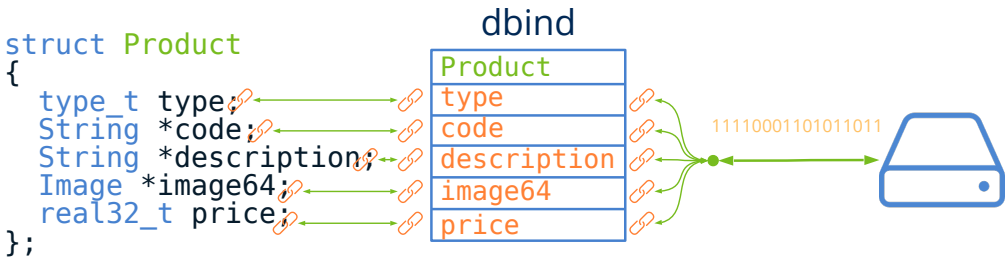


Figure 25.6: (De)serialization of binary objects by `dbind`.

25.3.3. Add/Delete records

And finally we will see how to add or delete records to the database using the constructors and destructors provided `dbind` by default. In (Listing 25.8) we create a new article and in (Listing 25.9) we destroy another existing one from its index.

Listing 25.8: Default constructor.

```
void model_add(Model *model)
{
    Product *product = dbind_create(Product);
    arrpt_append(model->products, product, Product);
}
```

Listing 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. View

We have fragmented the design of the main window into several blocks, each one implemented in its own *sublayout*. In *"Use of sublayoutsUse of sublayouts"* (page 392) and *"Sub-layoutsSub-layouts"* (page 337) you have examples about it. We start with a layout of a column and two rows (Listing 25.10) (Figure 25.7). In the upper cell we will place a sublayout with two other cells horizontally: one for the form and one for the login panel. The lower cell will be used for the *status bar*.

Listing 25.10: Composition of the main 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);
    return layout;
}
```

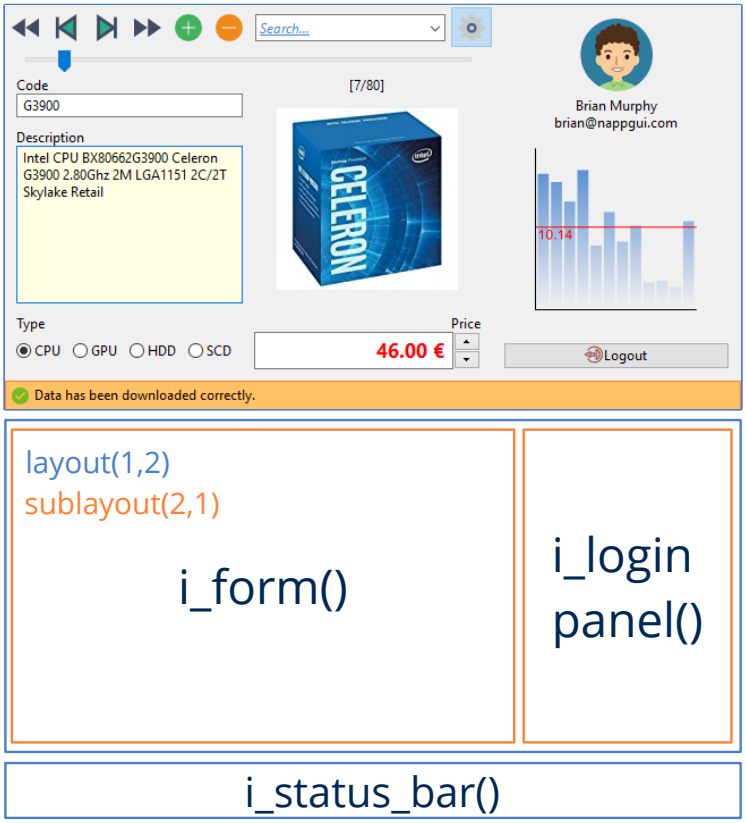


Figure 25.7: Main window layout.

In turn, the layout that integrates the form, implemented in `i_form()`, is composed of three cells in vertical (Figure 25.8): One for the toolbar `i_toolbar()`, another for the selection slider and another for the article data `i_product()`. This last cell is a sublayout of two columns and three rows. In the central row we locate the labels `Type` and `Price` and,

in the other two, four sublayout created by the functions `i_code_desc()` , `i_n_img()`, `i_type()` and `i_price()`.

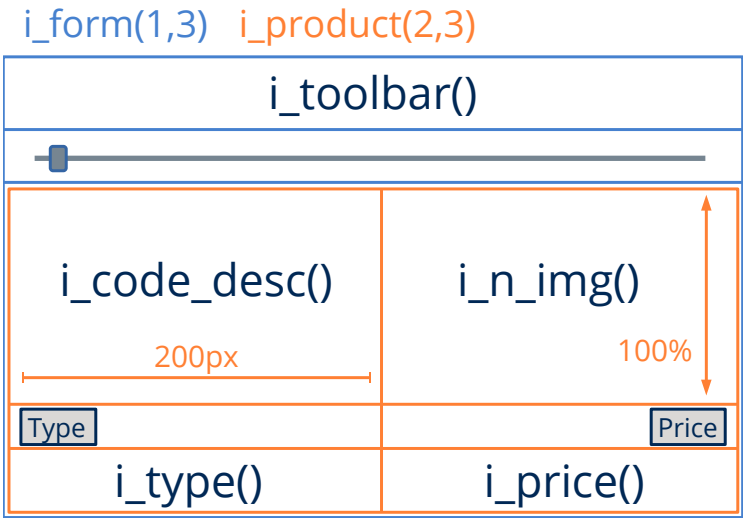


Figure 25.8: Layout que implementa el formulario.

If we look at the code of `i_product()`, reproduced partially in (Listing 25.11), we have made a *"Layout formatLayout format"* (page 29), assigning a minimum width and height for the upper cells. We also indicate that the vertical expansion is performed on row 0, avoiding the expansion of rows 1 and 2, corresponding to the *label*, the *radiobutton* and the price.

Listing 25.11: Format of 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);
    ...
}
```

25.4.1. Multi-layout panel

For user *login* we have used a panel with two different layouts: One for registration and another to show user data once registered (Listing 25.12) (Figure 25.9). This way, the controller can easily switch between them by calling `panel_visible_layout`. This

function will be responsible for displaying/hiding controls and recalculating the size of the window, since it may have suffered variations due to the change in layout.

Listing 25.12: Creation of a multi-layout panel.

```
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;
}
```

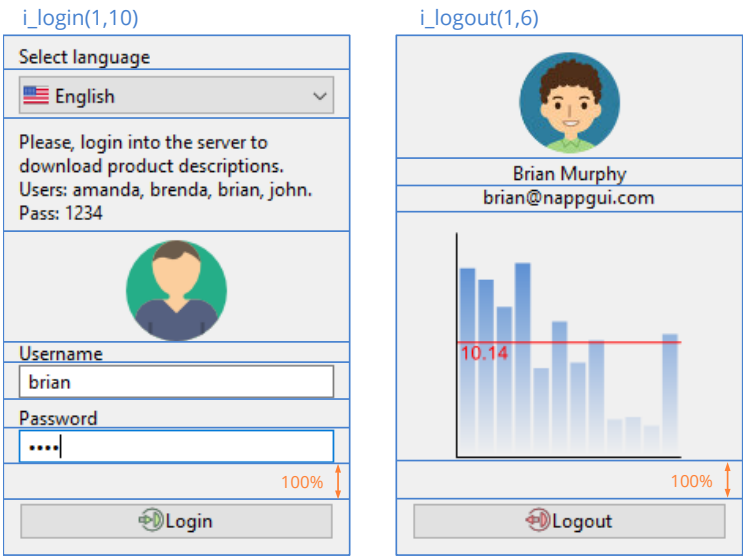


Figure 25.9: Login panel with two layouts.

25.4.2. Hide columns

It is also possible to hide the login panel through the menu or the corresponding button (Figure 25.10). This is simple to do inside the controller, acting on the column that contains said panel.

```
layout_show_col(ctrl->layout, 1, state == ekGUI_ON ? TRUE : FALSE);
```

25.4.3. Bar graphs

One of the requirements is that the interface includes a small bar chart that shows the sales statistics of each product (Figure 25.11). The code generated by this graphic is in

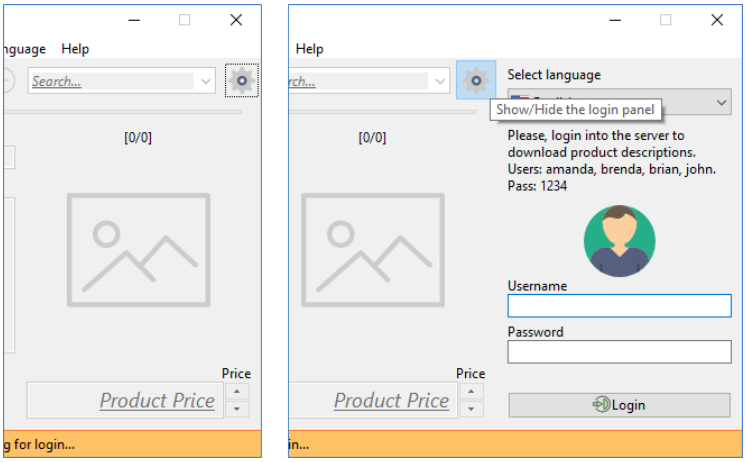


Figure 25.10: Show/Hide the login panel.

(Listing 25.13). In *"Use of Custom ViewsUse of Custom Views"* (page 394), *"Parametric drawingParametric drawing"* (page 395) and *"2D Contexts"* (page 257) you have more information about interactive graphics.

Listing 25.13: Parametric drawing of a bar graph.

```
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);
}
```

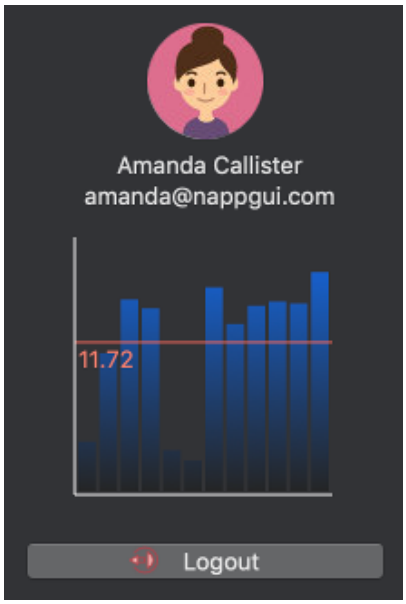


Figure 25.11: Dynamic graphs in the login panel.

25.4.4. Translations

The interface has been translated into seven languages, with English as default (Figure 25.12). To change the language, we call to `gui_language` within the `PopUp` event handler (Listing 25.14). In *"Resources"* (page 129) you have a step-by-step guide to locating and translating applications.

Listing 25.14: Code that changes the language of the program.

```
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]);
}
```

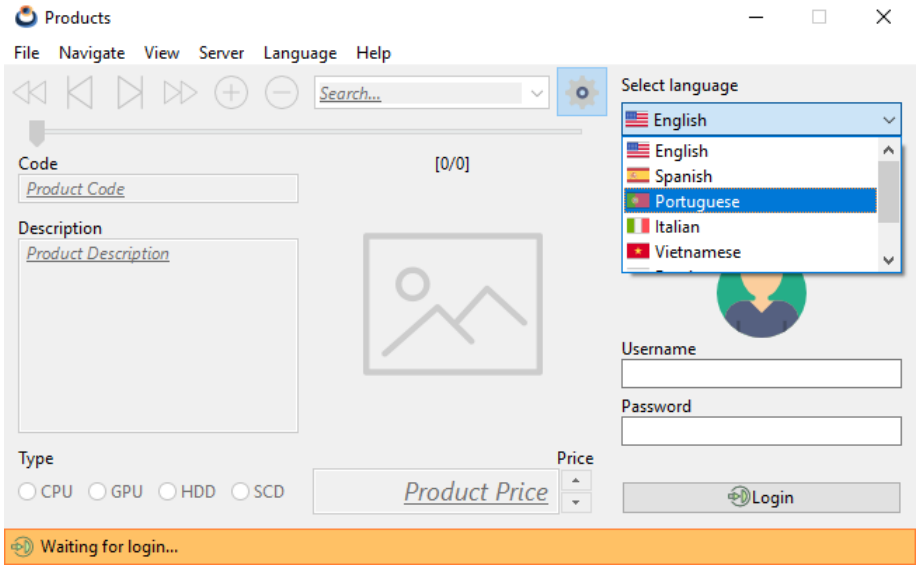


Figure 25.12: Automatic translations.

25.4.5. *Dark Mode* themes

NAppGUI uses native interface controls, which causes windows to integrate seamlessly with the active desktop theme on each machine. However, if we use custom icons or colors, these may not always be consistent when porting to other systems.

- In *"Gui"* (page 297) a series of "system" colors are defined, for example `gui_label_color`, whose RGB value will be resolved at runtime depending on the target platform. Using these functions, we will be certain that our applications will always look good and present a consistent color scheme. In *"Color table"* (page 643) you have a demo that shows these colors.

- Use `gui_alt_color` to define colors with two versions: One for light themes and one for dark ones. NAppGUI will be responsible for resolving the RGB whenever necessary (Listing 25.15).

Listing 25.15: Custom colors used in `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));
```

- For the images, we must include two versions in the program resources and select one or the other depending on the `gui_dark_mode` value (Listing 25.16).

Listing 25.16: Icon selection for *Light* or *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)
        ↪ ;
}
```

- Use `gui_OnThemeChanged` to update custom icons at runtime (Listing 25.17) (Figure 25.13).

Listing 25.17: Runtime icon update.

```
static void i_OnThemeChanged(App *app, Event *e)
{
    ctrl_theme_images(app->ctrl);
    unref(e);
}

gui_OnThemeChanged(listener(app, i_OnThemeChanged, App));
```

25.5. Controller

The controller is responsible for maintaining consistency between the Model and the View, as well as for implementing the **business logic**. Specifically, this program does virtually nothing with the data, regardless of downloading and displaying, which presents a good opportunity to practice.

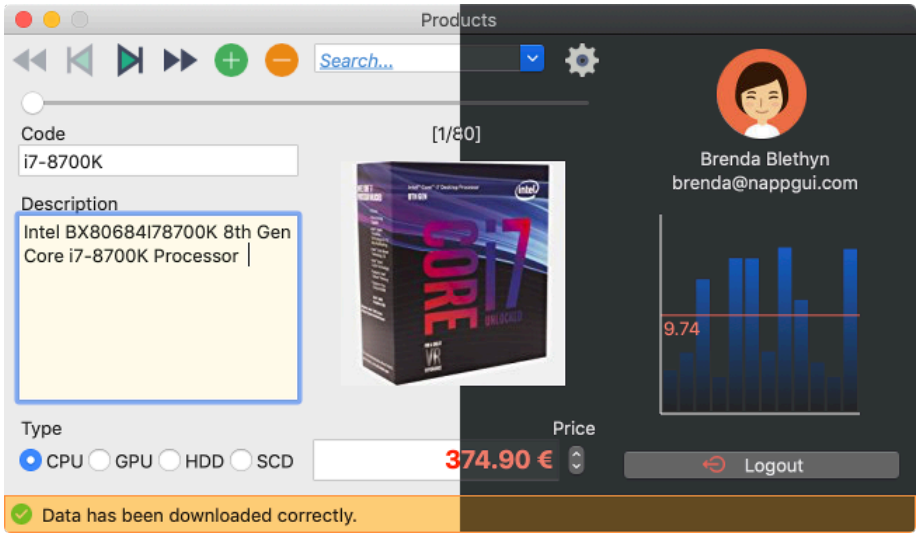


Figure 25.13: Desktop theme change.

25.5.1. Multi-threaded login

When the user presses the button `[Login]` the program calls two Web services. One to register the user and another to download the data. This process lasts about a second, which is an eternity from the point of view of a process. During this time you will come to appreciate that the program remains "frozen" waiting for the calls to the server to be resolved. This occurs because a "slow" task is running on the same thread that manages the program message loop (Figure 25.14)**(a)**.

To avoid this unpleasant effect, which can be aggravated if the request lasts longer, we will use *"Multi-threaded tasksMulti-threaded tasks"* (page 374) by `osapp_task` (Listing 25.18) (Figure 25.14)**(b)**. This creates a new execution thread that begins in `i_login_begin`. At the time the data has been downloaded, the NAppGUI task manager will call `i_login_end` (already in the main thread) and the program will continue with its (mono-thread) execution.

Listing 25.18: Multi-thread login process.

```
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);
}
```

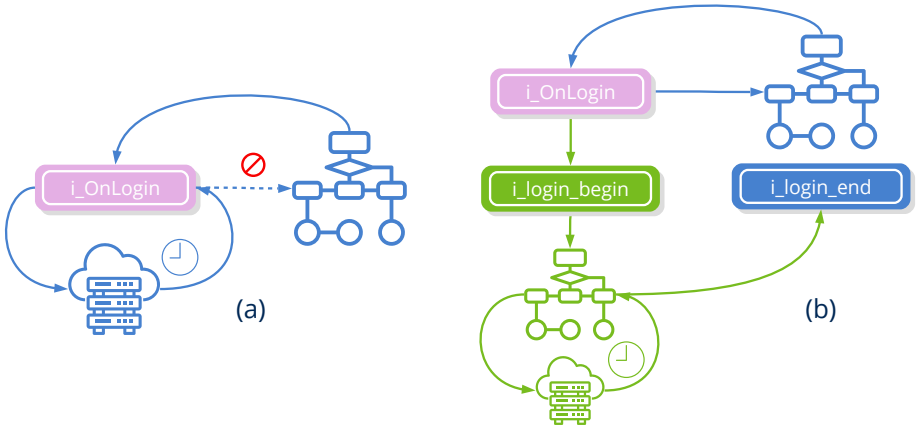


Figure 25.14: Execution of a "slow" task. Single-thread (a), Multi-thread (b). With a single thread the interface will be "frozen".

25.5.2. Synchronize Model and View

Keeping the Data Model and the View synchronized is also the controller's task. As the user interacts with the interface, it must capture the events, filter data and update the model objects. Similarly, every time the model changes it has to refresh the interface. This bidirectional synchronization can be done using **dbind**, saving a lot of extra programming code (Figure 25.15).

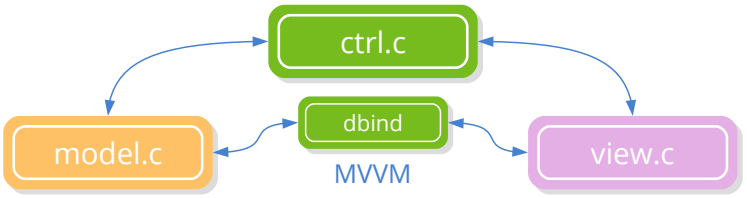


Figure 25.15: DBind helps the controller in the recurring task of synchronizing objects with the interface.

The implementation of this **MVVM** pattern *Model-View-ViewModel* is quite simple and we have it summarized in (Listing 25.19) (Figure 25.16).

- Use `cell_dbind` to link a layout cell with a model field.
- Use `layout_dbind` to link the layout containing the previous cells with the `struct` which contains the fields.
- Use `layout_dbind_obj` to assign an object to the layout. From here the Model-View updates will be made automatically.

Listing 25.19: Binding `struct` with `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);
```

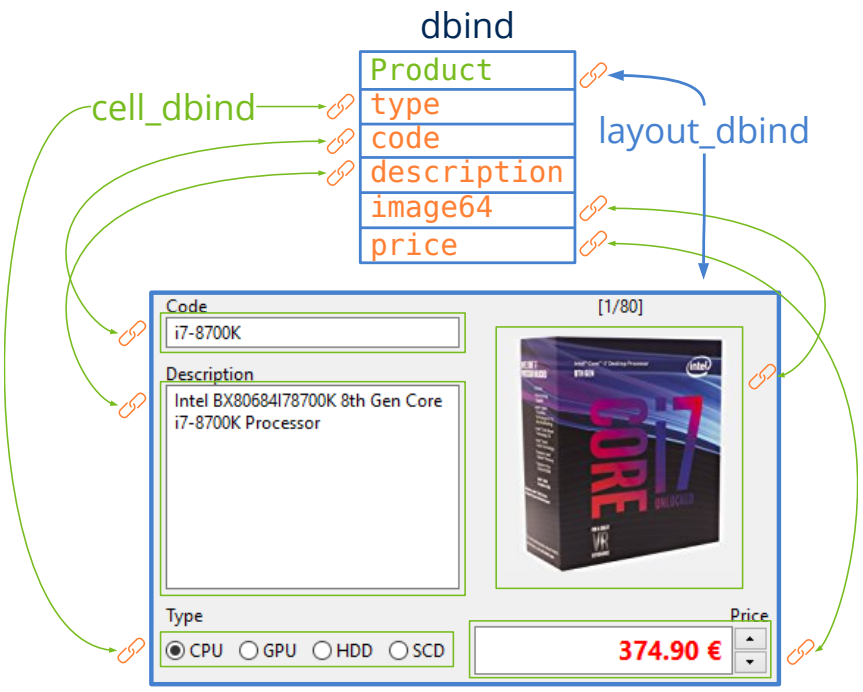


Figure 25.16: Data binding in GUI.

It is common for data to be reviewed (filtered) after editing to verify that the values are consistent with the model. **dbind** supports different formats for registered fields. In (Listing 25.20) we have applied formatting to the field `price`from `Product`.

Listing 25.20: Field format `price` from `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. Change the image

To change the image associated with the product, the controller has slightly modified the operation of the `ImageView`, which will show an edit icon each time the mouse is placed on top of the image (Listing 25.21), (Figure 25.17).

Listing 25.21: Drawing an *overlay* when the mouse is over the image.

```
static void i_OnImgDraw(Ctrl *ctrl, Event *e)
{
    const EvDraw *params = event_params(e, EvDraw);
    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));
```

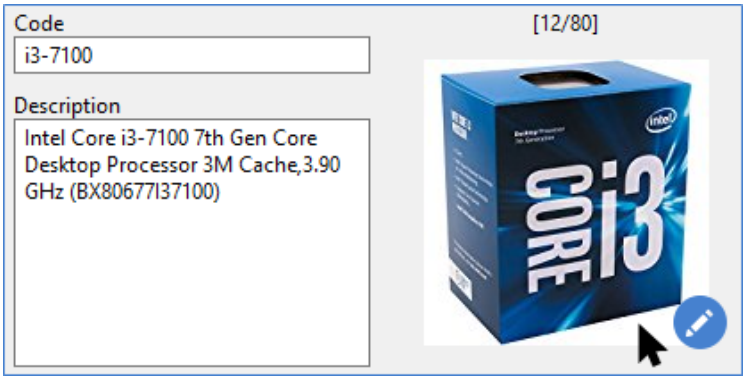


Figure 25.17: Superimposed icon on image control.

Clicking on the image will open the file opening dialog that will allow us to select a new one. If the dialog is accepted, the image will be loaded and assigned to control (Listing 25.22). The object will update automatically.

Listing 25.22: Drawing an *overlay* when the mouse is over the image.

```
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)
        {
            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. Memory management

After closing the program, a report will be printed with the use of memory, alerting us to possible *memory leaks* (Listing 25.23). It does not hurt to check it periodically in order to detect anomalies as soon as possible.

Listing 25.23: Memory usage statistics, generated at the close of any NAppGUI application.

```
[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] ============================
```

If we want more detailed information about the use of memory, we can pass the parameter `"-hv"` in the `options` field of `osmain` (Listing 25.24).

```
osmain(i_create, i_destroy, "-hv", App)
```

Listing 25.24: Detailed output of memory usage.

```
[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. The complete program

Listing 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
```

Listing 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)
```

Listing 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);
}
```

Listing 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;
}
```

Listing 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;
}
```

Listing 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);
    window_focus(ctrl->window, cell_control(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);
    window_focus(ctrl->window, cell_control(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);
        window_focus(ctrl->window, cell_control(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);
    window_focus(ctrl->window, cell_control(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));
}
```

Chapter 26

Hello GUI!

GuiHello is an application, which by examples, shows *"Gui"* (page 297) library features for the creation of user interfaces. The **source code** is in folder `/src/howto/guihello` of the SDK distribution.

26.1. Hello Label!

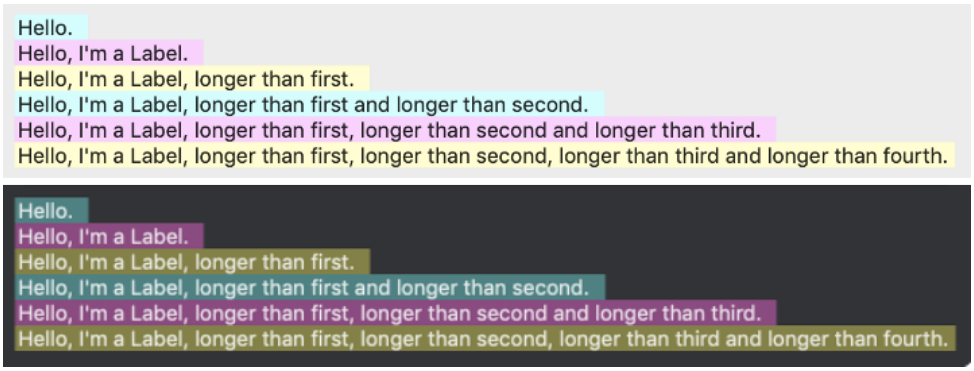


Figure 26.1: `Label` controls.

Listing 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. Hello Button!

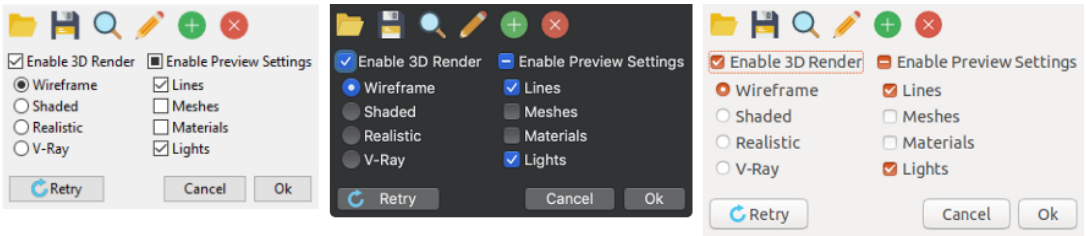


Figure 26.2: `Button` controls.

Listing 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, gui_image(FOLDER24_PNG));
    button_image(button2, gui_image(DISK24_PNG));
    button_image(button3, gui_image(SEARCH24_PNG));
    button_image(button4, gui_image(EDIT24_PNG));
    button_image(button5, gui_image(PLUS24_PNG));
    button_image(button6, gui_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, gui_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. Hello PopUp and Combo!

Figure 26.3: PopUp controls.

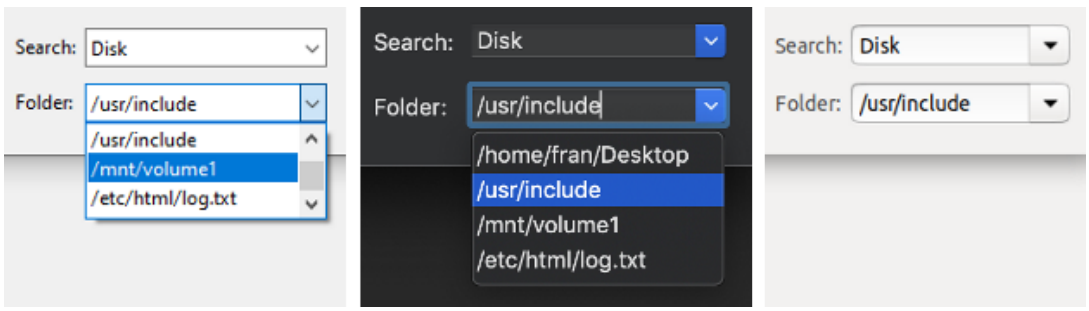


Figure 26.4: Combo controls.

Listing 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, "России", (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. Hello Edit and UpDown!

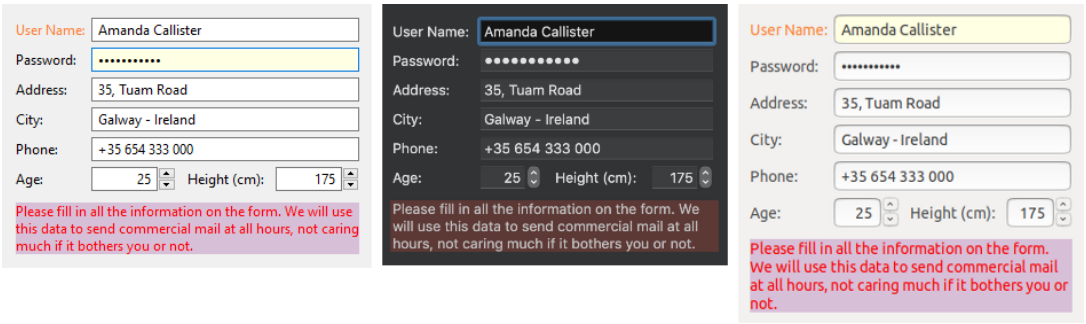


Figure 26.5: `Edit` and `UpDown` controls.

Listing 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. Hello ListBox!

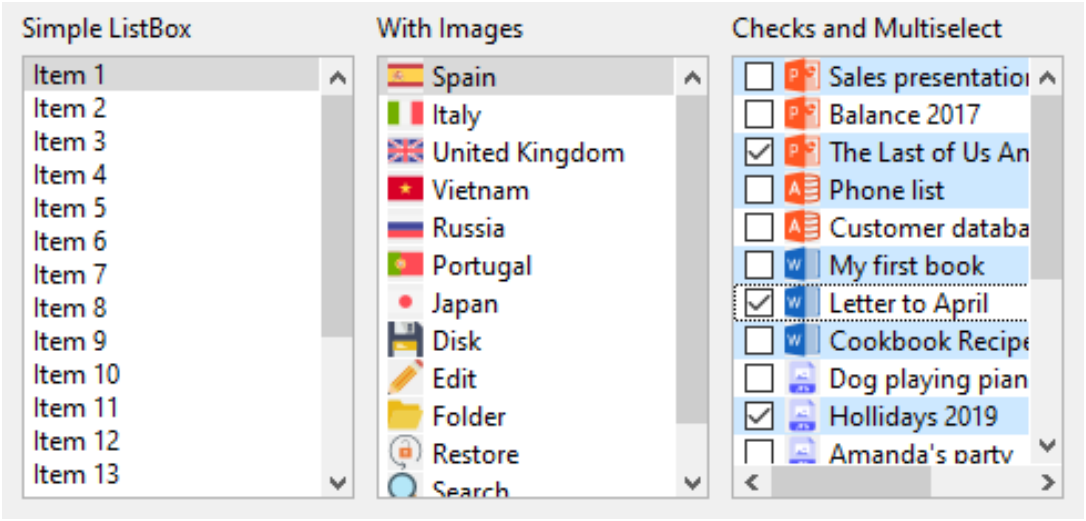


Figure 26.6: `ListBox` controls.

Listing 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", gui_image(POWERPOINT_PNG));
    listbox_add_elem(listbox, "Balance 2017", gui_image(POWERPOINT_PNG));
    listbox_add_elem(listbox, "The Last of Us Analysis", gui_image(
        ↪ POWERPOINT_PNG));
    listbox_add_elem(listbox, "Phone list", gui_image(ACCESS_PNG));
    listbox_add_elem(listbox, "Customer database", gui_image(ACCESS_PNG));
    listbox_add_elem(listbox, "My first book", gui_image(WORD_PNG));
    listbox_add_elem(listbox, "Letter to April", gui_image(WORD_PNG));
    listbox_add_elem(listbox, "Cookbook Recipes", gui_image(WORD_PNG));
    listbox_add_elem(listbox, "Dog playing piano", gui_image(JPG_PNG));
    listbox_add_elem(listbox, "Hollidays 2019", gui_image(JPG_PNG));
    listbox_add_elem(listbox, "Amanda's party", gui_image(JPG_PNG));
    listbox_add_elem(listbox, "Flying", gui_image(JPG_PNG));
    listbox_add_elem(listbox, "The C Programing Language", gui_image(PDF_PNG));
    listbox_add_elem(listbox, "Graphics Programing with GDI+", gui_image(
        ↪ PDF_PNG));
    listbox_add_elem(listbox, "Personal finances", gui_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Stocks 2017", gui_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Website Dashboard", gui_image(EXCEL_PNG));
    listbox_add_elem(listbox, "Open Issues", gui_image(DOCUMENT_PNG));
    listbox_add_elem(listbox, "TODO List", gui_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", gui_image(SPAIN_PNG));
    listbox_add_elem(listbox, "Italy", gui_image(ITALY_PNG));
    listbox_add_elem(listbox, "United Kingdom", gui_image(UKING_PNG));
    listbox_add_elem(listbox, "Vietnam", gui_image(VIETNAM_PNG));
    listbox_add_elem(listbox, "Russia", gui_image(RUSSIA_PNG));
    listbox_add_elem(listbox, "Portugal", gui_image(PORTUGAL_PNG));
    listbox_add_elem(listbox, "Japan", gui_image(JAPAN_PNG));
    listbox_add_elem(listbox, "Disk", gui_image(DISK16_PNG));
    listbox_add_elem(listbox, "Edit", gui_image(EDIT16_PNG));
    listbox_add_elem(listbox, "Folder", gui_image(FOLDER16_PNG));
    listbox_add_elem(listbox, "Restore", gui_image(RESTORE16_PNG));
    listbox_add_elem(listbox, "Search", gui_image(SEARCH16_PNG));
    listbox_add_elem(listbox, "Error", gui_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. Hello Slider and Progress!

Listing 26.6: demo/guihello/sliders.c

```
/* Sliders */

#include "sliders.h"
#include <gui/guiall.h>
```

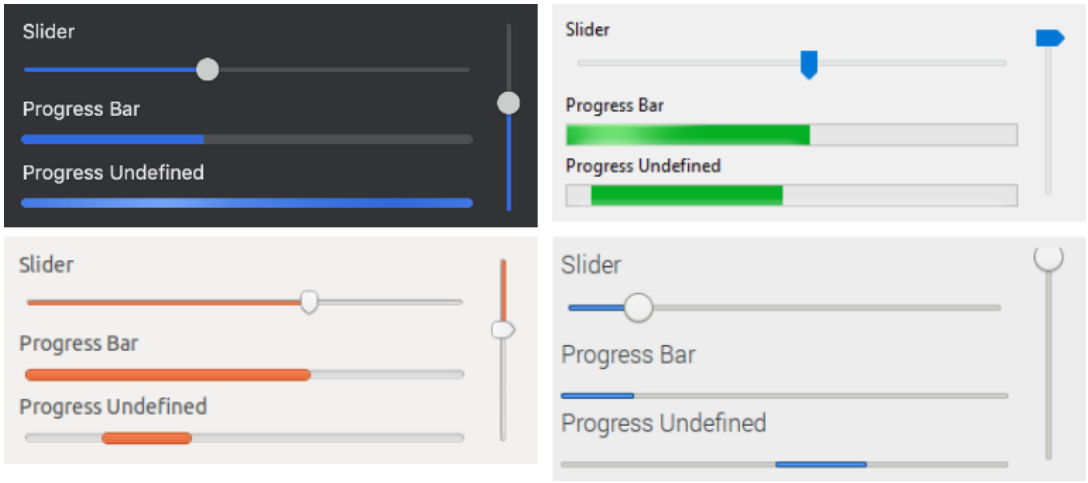


Figure 26.7: `Slider` and `Progress` controls.

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

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. Hello TextView!

Listing 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);
}

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

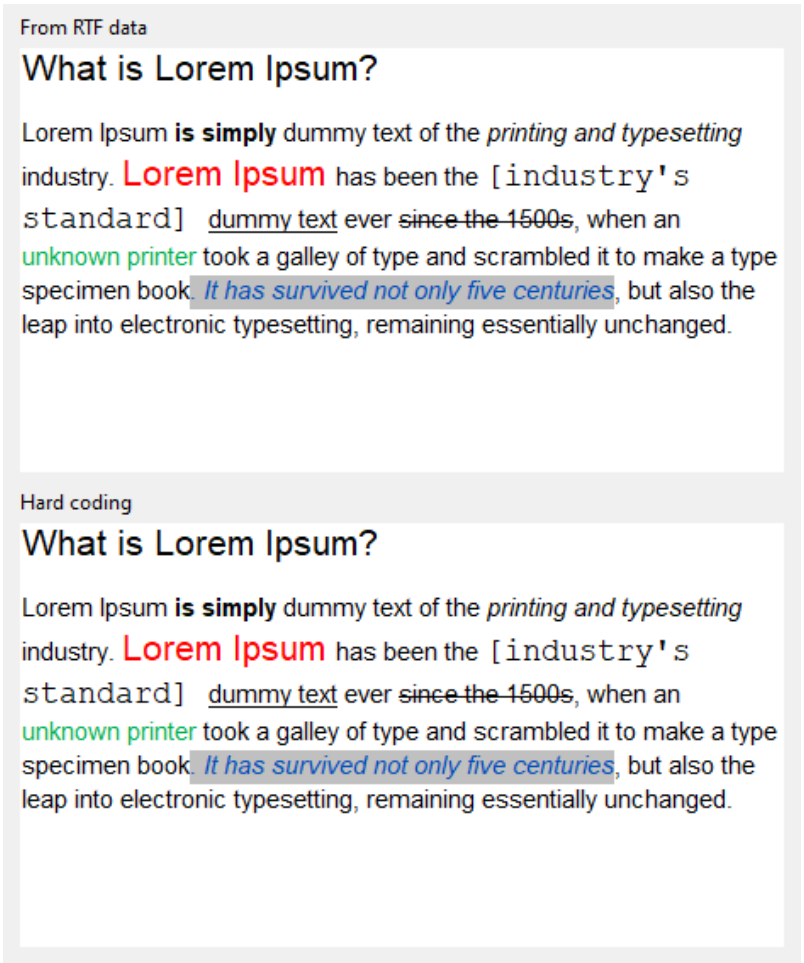


Figure 26.8: Rich text control.

```
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. Hello TableView!

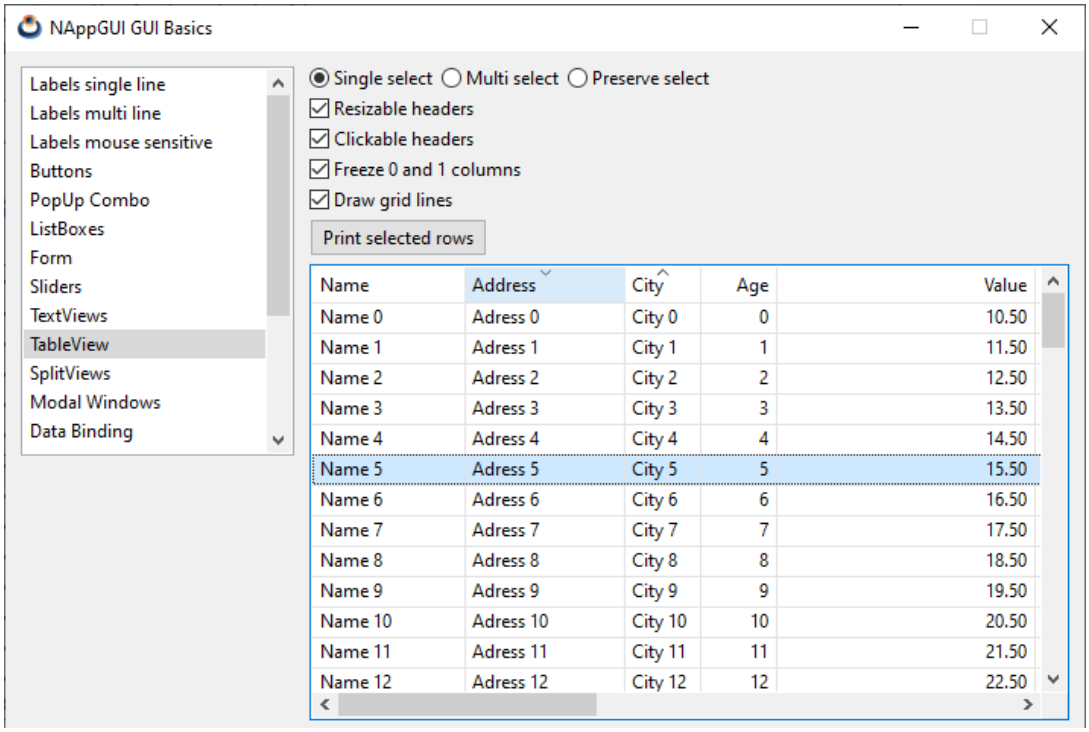


Figure 26.9: Table control.

Listing 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. Hello SplitView!

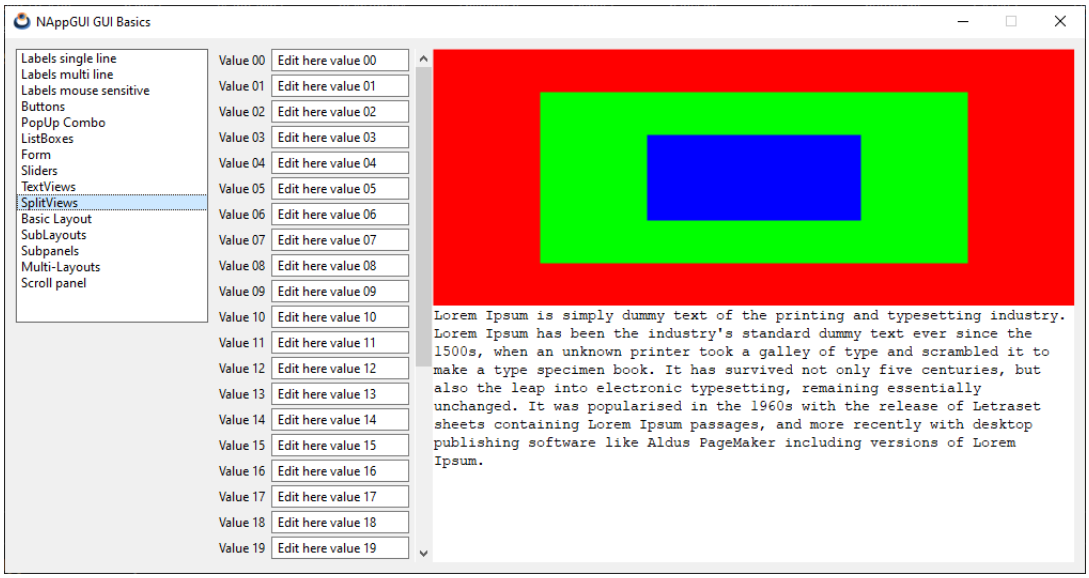


Figure 26.10: SplitView.

Listing 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, FALSE);
    splitview_text(split2, text, FALSE);
    splitview_panel(split1, panel2);
    splitview_split(split1, split2);
    layout_splitview(layout, split1, 0, 0);
    panel_layout(panel1, layout);
    return panel1;
}
```

26.10. Hello Modal Window!

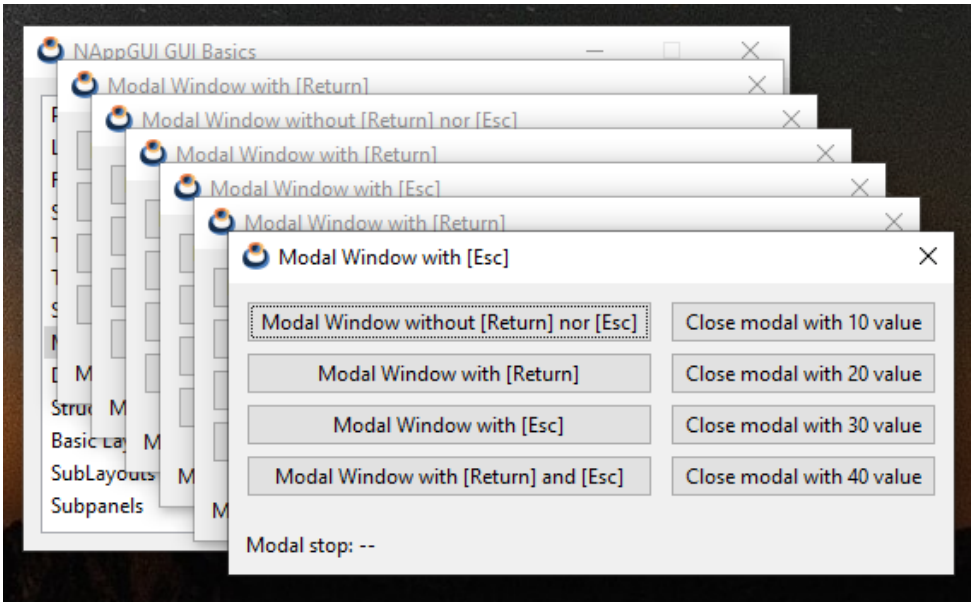


Figure 26.11: Modal windows.

Listing 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. Hello Gui Binding!

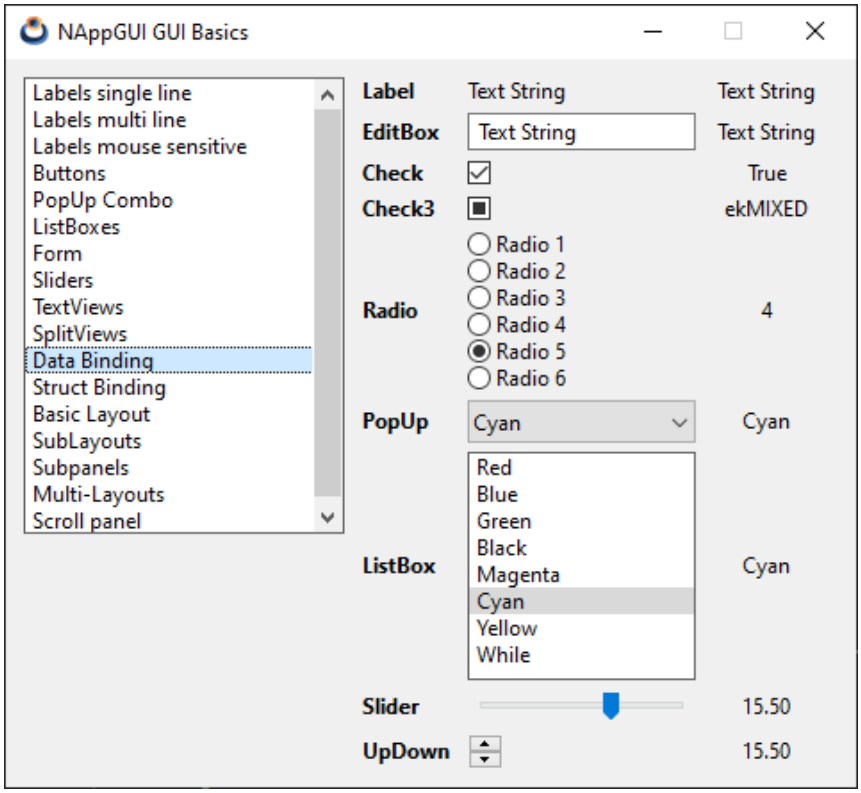


Figure 26.12: Gui Data binding.

Listing 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. Hello Struct Binding!

Listing 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;
};

struct _structtypes_t
```

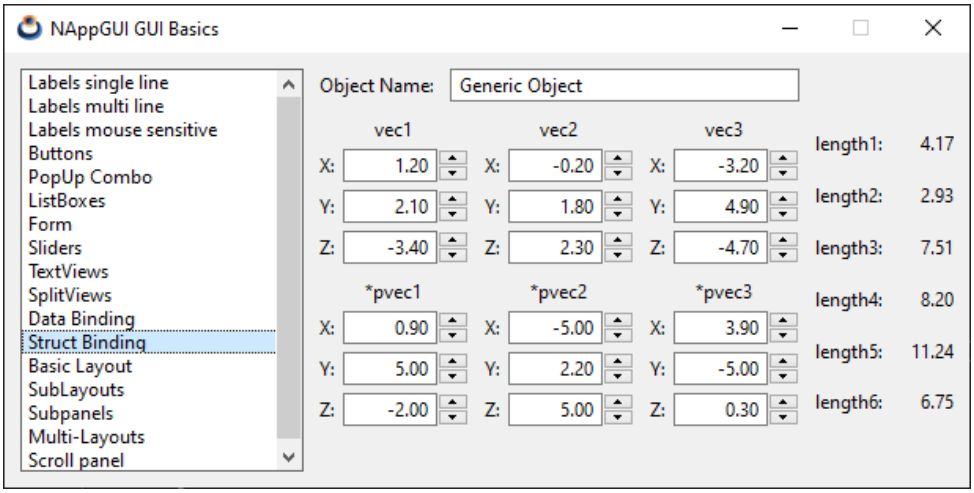


Figure 26.13: Gui Struct binding.

```
{
    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. Hello Sublayout!

Listing 26.13: demo/guihello/sublayout.c

```
/* Sublayouts */
```

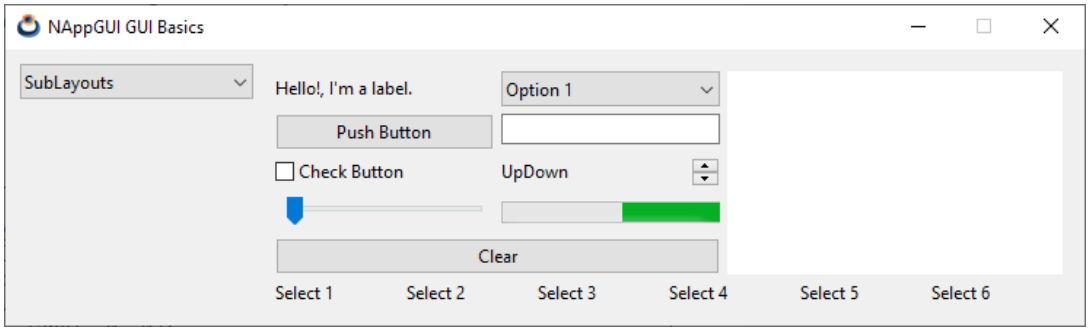


Figure 26.14: Sublayout composition.

```
#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. Hello Subpanel!

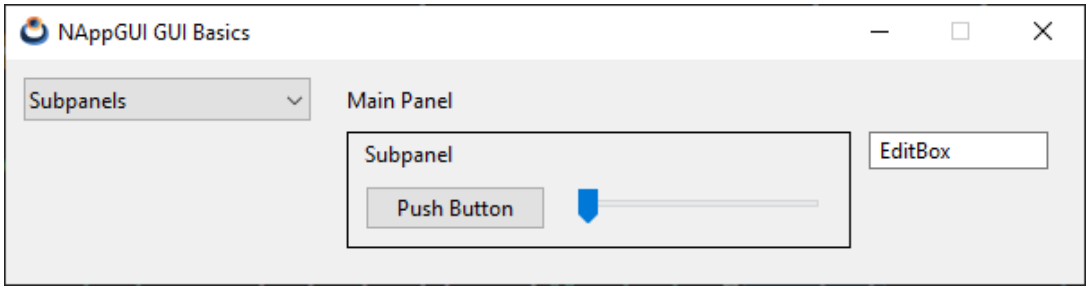


Figure 26.15: Subpanels.

Listing 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. Hello Multi-layout!

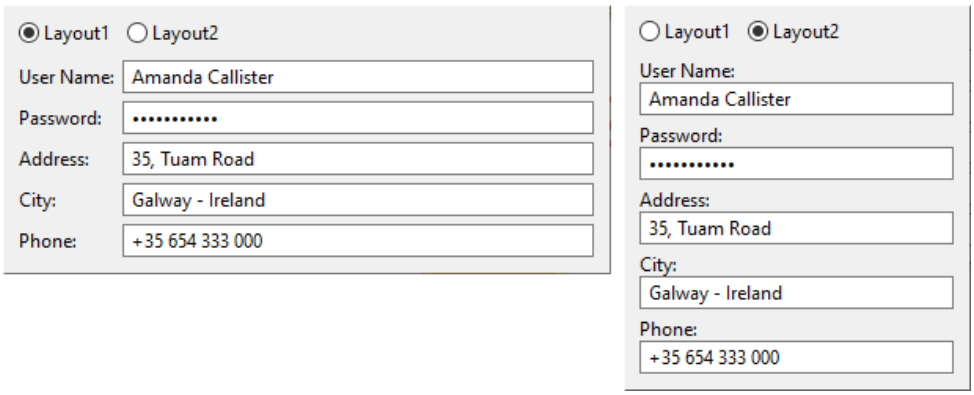


Figure 26.16: `Panel` with two layouts.

Listing 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. Hello Scroll-Panel!

Listing 26.16: demo/guihello/scrollpanel.c

```
/* Panel with scroll */

#include "scrollpanel.h"
#include <gui/guiall.h>

static const uint32_t i_ROWS = 100;

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

Panel *scrollpanel(void)
{
```

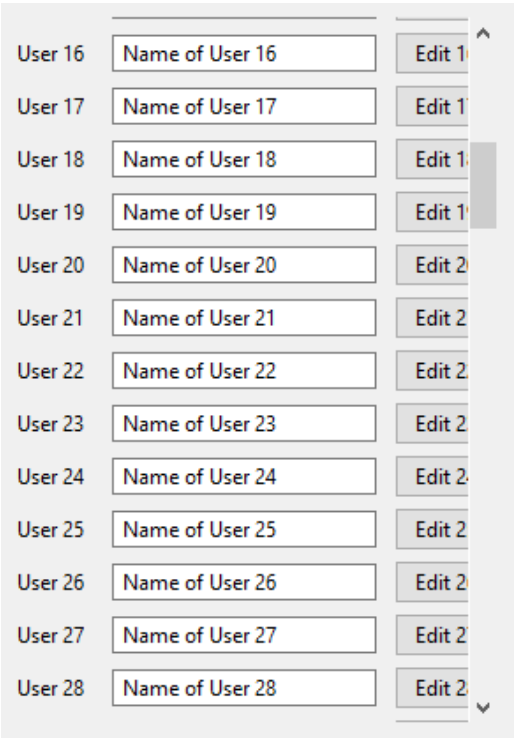


Figure 26.17: `Panel` with scroll bars.

```
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. Hello IP-Input!

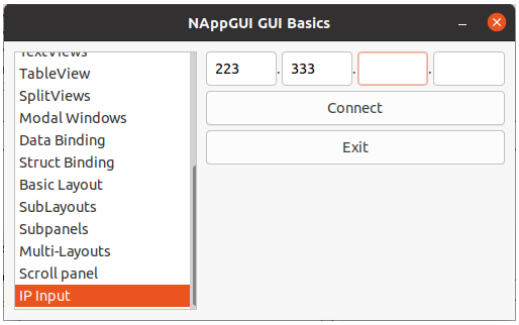


Figure 26.18: The `Edit` commands automatically change the keyboard focus after inserting the third character.

Listing 26.17: demo/guihello/ipinput.c

```
/* IP input */

#include "ipinput.h"
#include <gui/guiall.h>

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

static void i_OnEditFilter(Window *window, 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)
        window_next_tabstop(window);
}

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

Panel *ip_input(Window *window)
{
    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(window, i_OnEditFilter, Window));
    edit_OnFilter(edit2, listener(window, i_OnEditFilter, Window));
    edit_OnFilter(edit3, listener(window, i_OnEditFilter, Window));
    edit_OnFilter(edit4, listener(window, i_OnEditFilter, Window));
    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;
}
```

Chapter 27

Hello Draw2d!

DrawHello is an application, which by example, shows the *"Draw2D"* (page 256) library features for 2D vector drawing. Implements line drawing, region fill, texts and images. The **source code** is in folder `/src/howto/drawhello` of the SDK distribution.

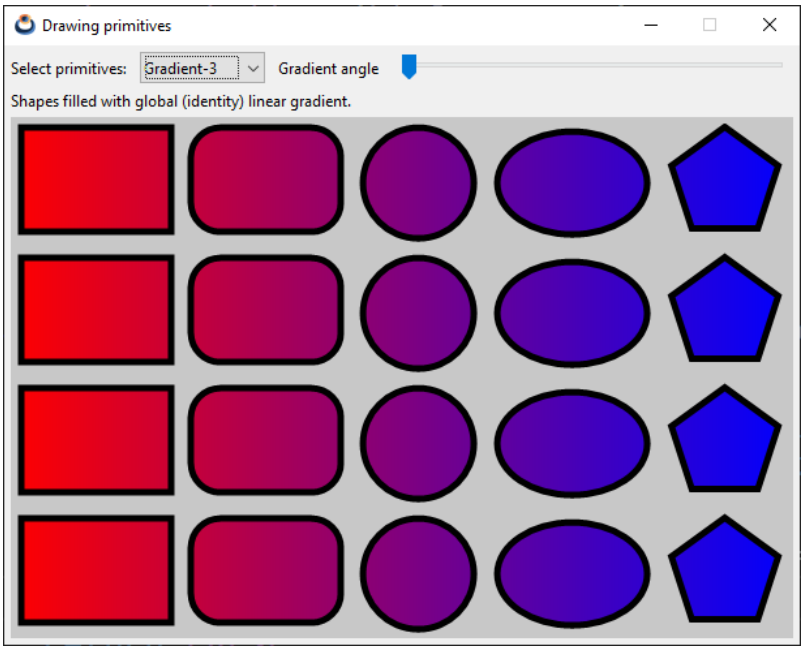


Figure 27.1: Windows version.

Listing 27.1: demo/drawhello/drawhello.c

```
/* Drawing primitives */

#include "res_drawhello.h"
#include <nappgui.h>
```

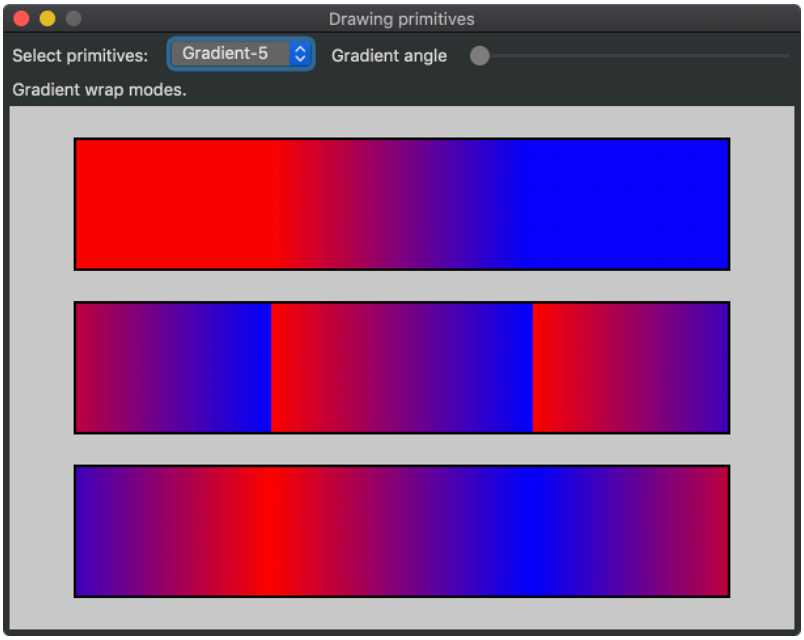


Figure 27.2: macOS version.

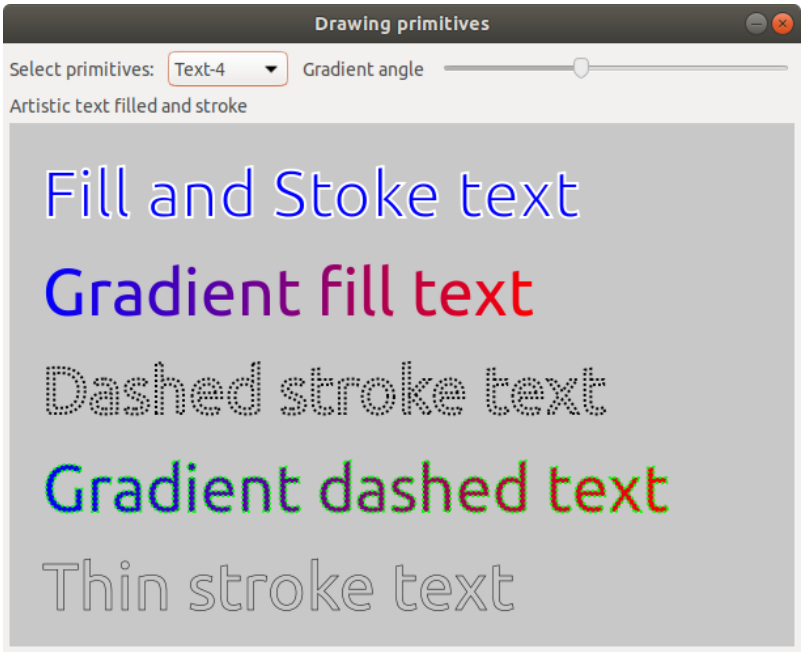


Figure 27.3: Linux version.

```
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)
```

Chapter 28

Hello 2D Collisions!

Col2dHello is a small environment for experimentation with 2D collision detection algorithms. It allows you to create different types of volumes, move them with the mouse and edit them through the side panel. The details of the functions can be found in *"2D Collisions"* (page 253).

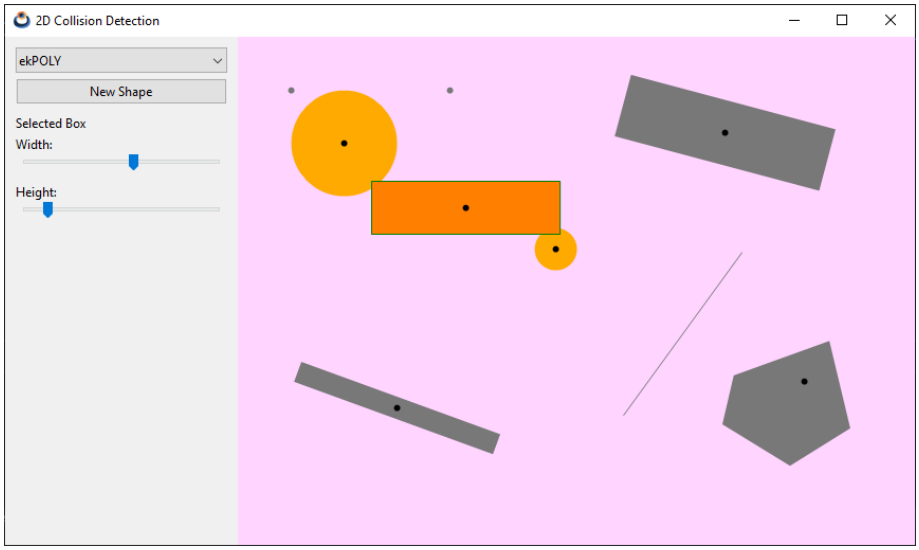


Figure 28.1: Windows version.

Listing 28.1: demo/col2dhello/col2dhello.c

```
/* 2D collision detection demo */

#include "col2dgui.h"
#include <nappgui.h>

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

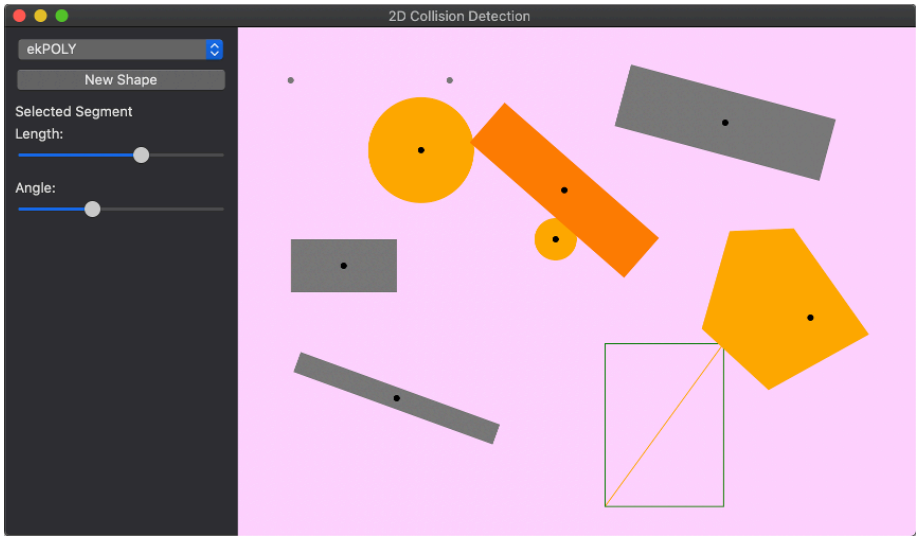


Figure 28.2: MacOS version.

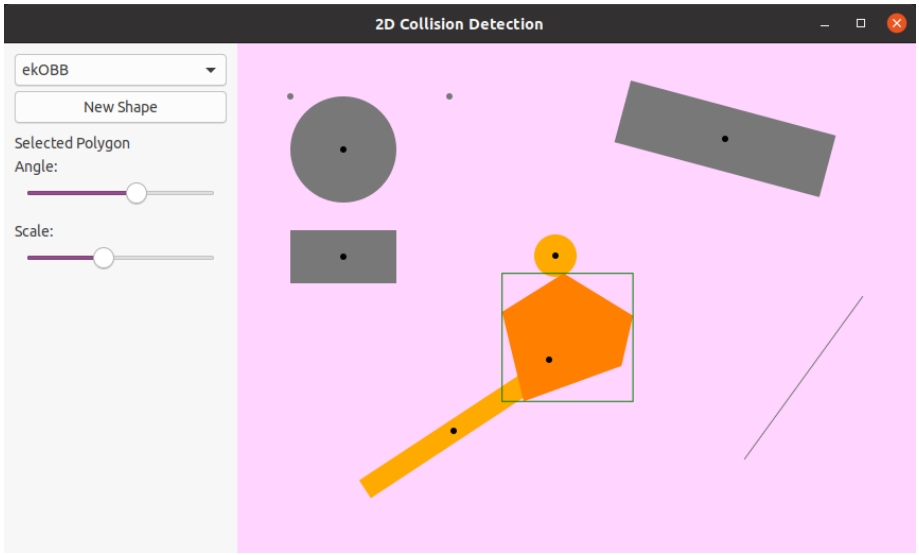


Figure 28.3: Linux version.

```
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;
}

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

```
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)
```

Listing 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
```

Listing 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);
}
```

CHAPTER 29

Drawing on an image

In this example we see how to generate vector graphics in two different contexts using the same drawing code (Figure 29.1). On the left side we render directly into the window through a `View` control. On the right side generate an image using different resolutions. To show it we use a `ImageView` control configured to stretch the image in case it is smaller than the control itself, which makes clear the loss of quality. The **source code** is in folder `/src/howto/drawimg` of the SDK distribution.

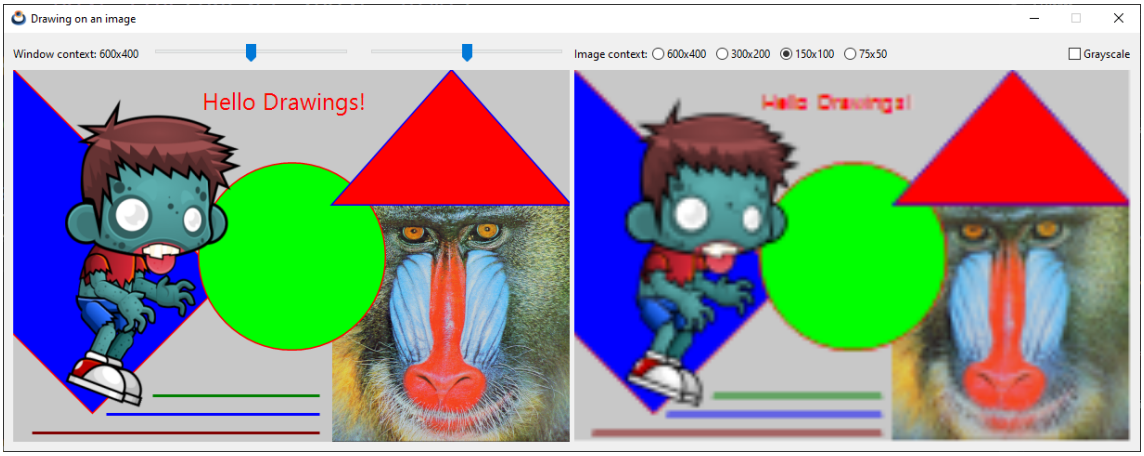


Figure 29.1: 2D Contexts: Window (left), Image (right).

Listing 29.1: demo/drawimg/drawimg.c

```
/* Drawing on an image */

#include "res_drawimg.h"
#include <nappgui.h>

typedef struct _app_t App;
```

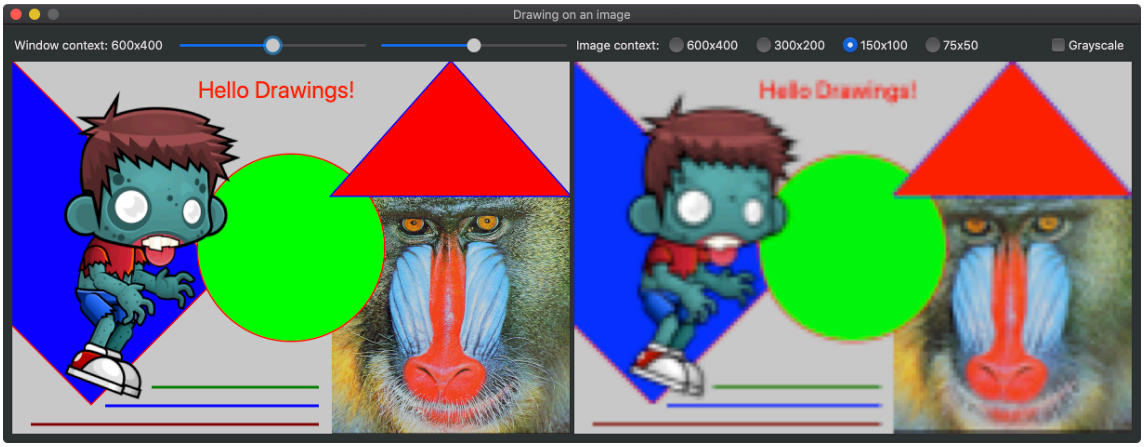


Figure 29.2: macOS version.

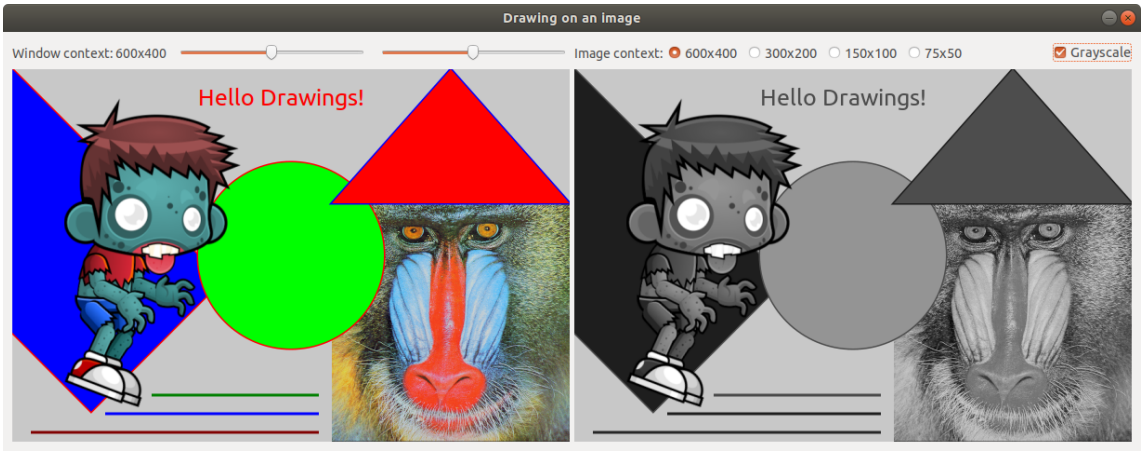


Figure 29.3: Linux version.

```
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)
```

Chapter 30

Scroll drawings

The next application shows how to manage a very large drawing area, of which only a small portion is visible. We will represent a grid of 2000x2000 cells, using a `View` control with scroll bars. The objectives we are pursuing with this example are:

- Optimize the `OnDraw` event to draw only the visible area, avoiding launching unnecessary commands.
- Size scroll bars with `view_content_size`.
- Move the visible area using `view_scroll_x`, `view_scroll_y`.
- Get the visible area with `view_viewport`.
- Use of the mouse: To be able to click on a cell or highlight it when the cursor is hover it.
- Using the keyboard: Allow the view to capture the focus and move the active cell with the `[Left]`, `[Right]`, `[Up]` and `[Down]` keys. Keyboard navigation requires this cell to always be visible.

Listing 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;
    uint32_t row_id;
```

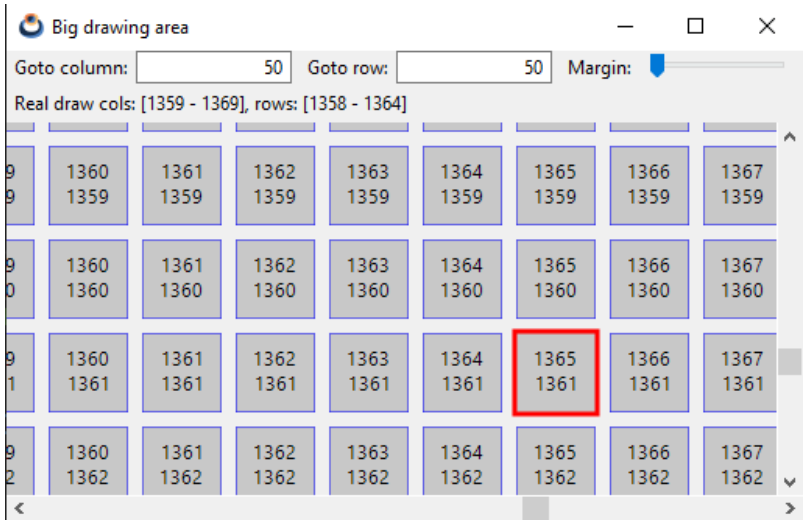


Figure 30.1: Windows version.

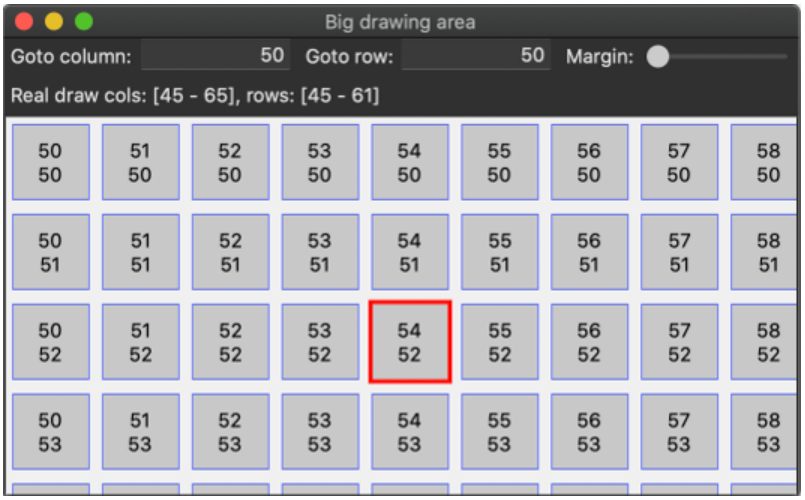


Figure 30.2: macOS version.

```
    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;
static const uint32_t i_NUM_ROWS = 2000;
```

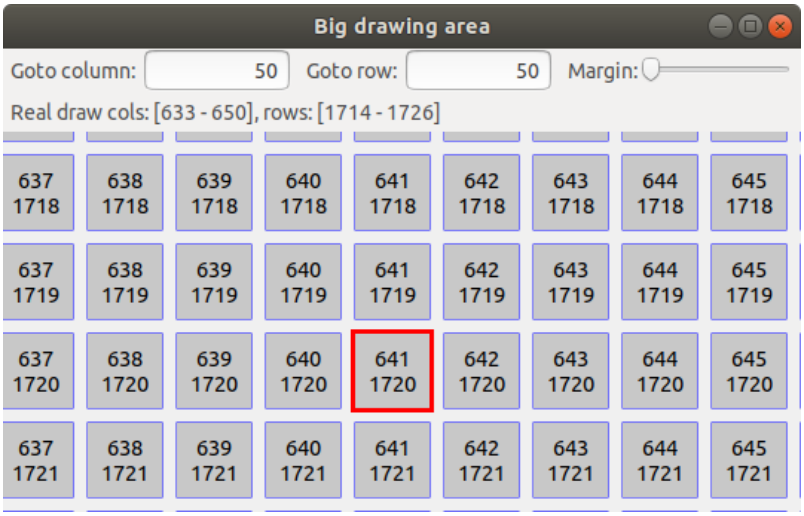


Figure 30.3: Linux version.

```
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)
```

Chapter 31

Images from URLs

In this demo we build a simple web image viewer. The program allows you to download and view them through a list. The **source code** is in folder `/src/howto/urlimg` of the SDK distribution.

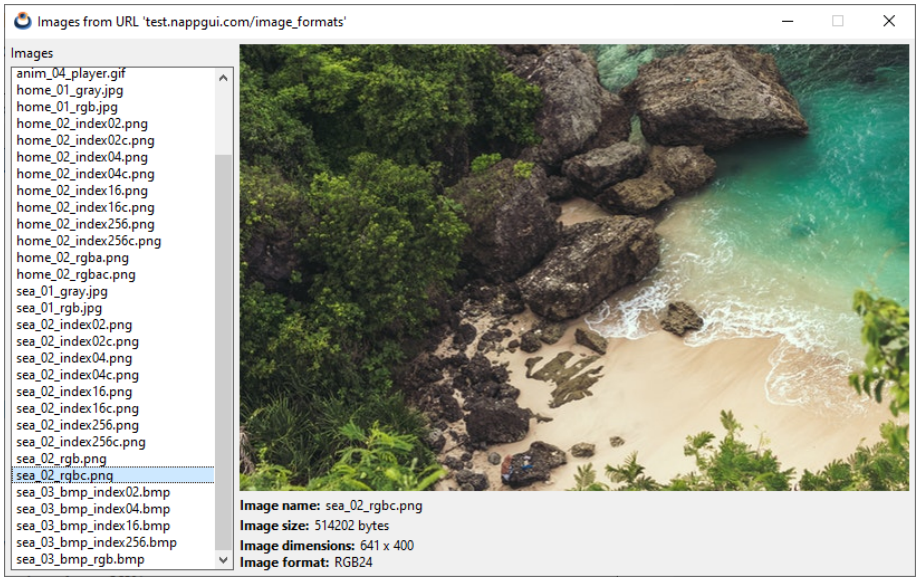


Figure 31.1: Windows version

Listing 31.1: demo/urlimg/urlimg.c

```
/* Images from URL */

#include <inet/inet.h>
#include <inet/httpreq.h>
#include <nappgui.h>
```

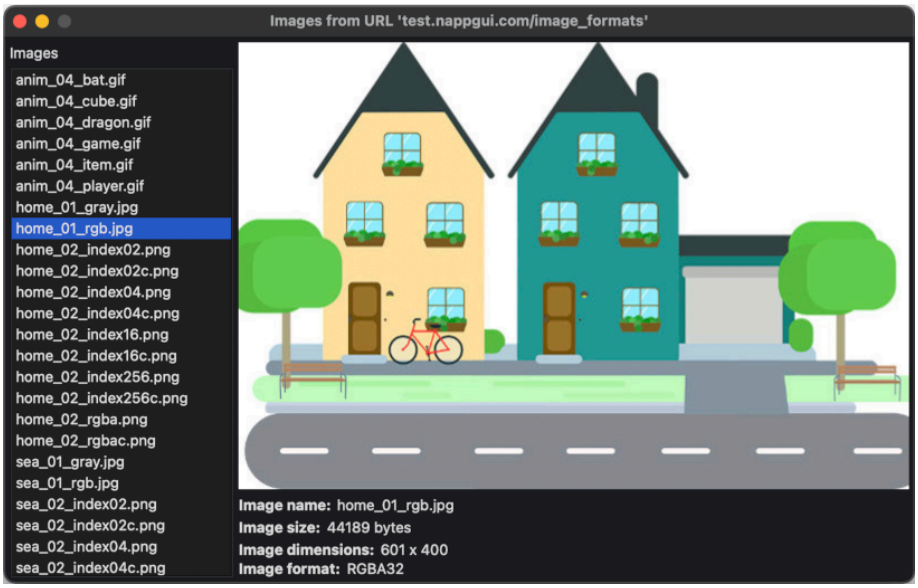


Figure 31.2: macOS version

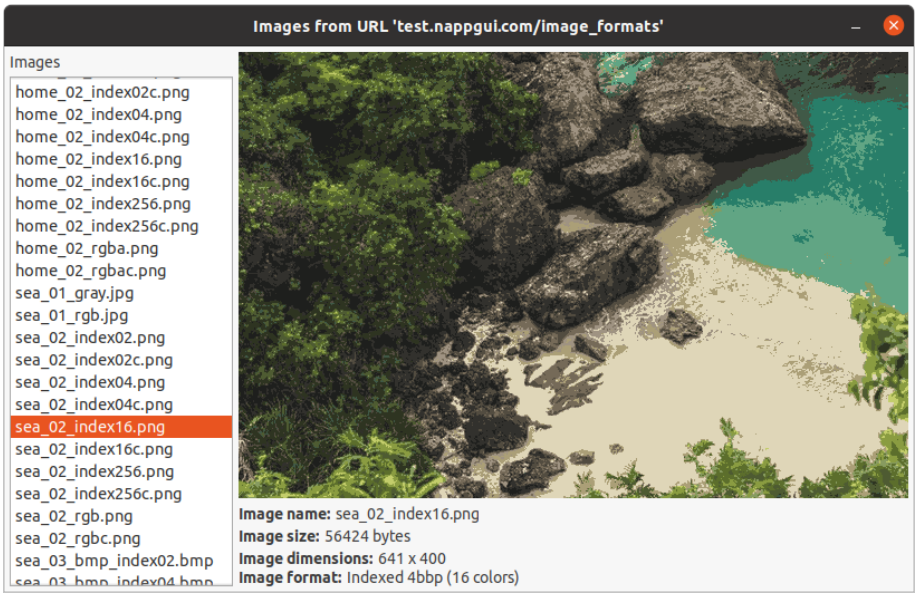


Figure 31.3: Linux version

```
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)
```

Chapter 32

Color table

The choice of arbitrary RGB colors for use in graphic interfaces will not always be consistent with the desktop theme of the target platform. In *"Colors"* (page 277) a series of "system" colors are defined and the possibility of creating alternative versions for light or dark themes. This demo shows this repertoire depending on the platform where the program runs. The **source code** is in folder `/src/howto/colorview` of the SDK distribution.

Figure 32.1: Color table.

Listing 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)
```

Chapter 33

Read/Write Json

Listing 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;
}
```

Program output.

```
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 } ] }
```

Chapter 34

Alternative to STL

The C++ *Standard Template Library* provides generic containers and algorithms as part of the language. The problem is that they cannot be used from "pure" C code, so NAppGUI provides an implementation of Arrays and Set at least as efficient as those of STL.

Result in 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
```

Listing 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;
}
```

Part 4

Library reference

Chapter 35

Sewer library

35.1. Types and Constants

int8_t

8-bit signed integer. It can represent a value between `INT8_MIN` and `INT8_MAX`.

int16_t

16-bit signed integer. It can represent a value between `INT16_MIN` and `INT16_MAX`.

int32_t

32-bit signed integer. It can represent a value between `INT32_MIN` and `INT32_MAX`.

int64_t

64-bit signed integer. It can represent a value between `INT64_MIN` and `INT64_MAX`.

uint8_t

8-bit unsigned integer. It can represent a value between 0 and `UINT8_MAX`.

uint16_t

16-bit unsigned integer. It can represent a value between 0 and `UINT16_MAX`.

uint32_t

32-bit unsigned integer. It can represent a value between 0 and `UINT32_MAX`.

uint64_t

64-bit unsigned integer. It can represent a value between 0 and `UINT64_MAX`.

char_t

8-bit character type (Unicode). A single character may need 1, 2, 3 or 4 elements (bytes), depending on *"UTF encodingsUTF encodings"* (page 157).

byte_t

8-bit type to store generic memory blocks.

bool_t

8-bit boolean. Only two values are allowed `TRUE` (1) and `FALSE` (0).

real

32 or 64-bit floating point number.

real32_t

32-bit floating point number. The C `float` type.

real64_t

64-bit floating point number. The C `double` type.

TRUE

True.

```
const bool_t TRUE = 1;
```

FALSE

False.

```
const bool_t FALSE = 0;
```

NULL

Null pointer.

```
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

Euler's number.

```
const real32_t kBMATH_Ef = 2.718281828459045f;
const real64_t kBMATH_Ed = 2.718281828459045;
const real BMath::kE;
```

kLN2

The natural logarithm of 2.

```
const real32_t kBMATH_LN2f = 0.6931471805599453f;
const real64_t kBMATH_LN2d = 0.6931471805599453;
const real BMath::kLN2;
```

kLN10

The natural logarithm of 10.

```
const real32_t kBMATH_LN10f = 2.302585092994046f;
const real64_t kBMATH_LN10d = 2.302585092994046;
const real BMath::kLN10;
```

kPI

The number Pi.

```
const real32_t kBMATH_PIf = 3.141592653589793f;
const real64_t kBMATH_PId = 3.141592653589793;
const real BMath::kPI;
```

kSQRT2

Square root of 2.

```
const real32_t kBMATH_SQRT2f = 1.414213562373095f;
const real64_t kBMATH_SQRT2d = 1.414213562373095;
const real BMath::kSQRT2;
```

kSQRT3

Square root of 3.

```
const real32_t kBMATH_SQRT3f = 1.732050807568878f;
const real64_t kBMATH_SQRT3d = 1.732050807568878;
const real BMath::kSQRT3;
```

kDEG2RAD

Conversion from one degree to radians.

```
const real32_t kBMATH_DEG2RADf = 0.017453292519943f;
const real64_t kBMATH_DEG2RADd = 0.017453292519943;
const real BMath::kDEG2RAD;
```

kRAD2DEG

Conversion of a radian to degrees.

```
const real32_t kBMATH_RAD2DEGf = 57.2957795130823f;
const real64_t kBMATH_RAD2DEGd = 57.2957795130823;
const real BMath::kRAD2DEG;
```

kINFINITY

Infinite, represented by a very large value.

```
const real32_t kBMATH_INFINITYf = ∞f;
const real64_t kBMATH_INFINITYd = ∞;
const real BMath::kINFINITY;
```

enum unicode_t

Represents the *"UTF encodingsUTF encodings"* (page 157).

ekUTF8	UTF8 encoding.
ekUTF16	UTF16 encoding.
ekUTF32	UTF32 encoding.

struct REnv

"Random numbersRandom numbers" (page 160) environment.

```
struct REnv;
```

35.2. Functions

FPtr_destroy

Destructor function prototype.

```
void
(*FPtr_destroy)(type **item);
```

item Double pointer to the object to destroy. It must be assigned to `NULL` after the destruction to invalidate its use.

FPtr_copy

Copy constructor function prototype.

```
type*
(*FPtr_copy)(const type *item);
```

item Pointer to the object to be copied.

Return:

The new object that is an exact copy of the input.

FPtr_scopy

Unallocated memory copy constructor prototype.

```
void
(*FPtr_scopy)(type *dest,
              const type *src);
```

dest Destination object (copy).

src Pointer to the object to be copied (source).

Remarks:

In this copy operation, the memory required by the object has already been allocated. We must create dynamic memory for the fields of the object that require it, but not for the object itself. Usually used to copy arrays of objects (not pointers to objects).

FPtr_compare

Comparison function prototype.

```
int
(*FPtr_compare)(const type *item1,
                const type *item2);
```

item1 First item to compare.

item2 Second item to compare.

Return:

Comparison result.

FPtr_compare_ex

Similar to `FPtr_compare`, but receive an additional parameter that may influence the comparison.

```
int
(*FPtr_compare_ex)(const type *item1,
                   const type *item2,
                   const dtype *data);
```

item1	First item to compare.
item2	Second item to compare.
data	Additional parameter.

Return:

Comparison result.

FPtr_assert

Callback function prototype called when an `assert` occurs.

```
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	User data passed as the first parameter.
group	0 = Fatal error, 1 = Execution can continue.
caption	Title.
detail	Detailed message.
file	Source file where the assert occurred.
line	Line inside the source file.

unref

Mark the parameter as non-referenced, disabling the compiler's warnings.

```
void
unref(param);
```

```
static void i_OnClick(App *app, Event *e)
{
    unref(e);
    app_click_action(app);
}
```

param Parameter.

cassert

Basic *assert* sentence. If the condition is evaluated at `FALSE` , a "continuable" *assert* will be launched. The message shown will be the literal of the condition itself.

```
void
cassert(bool_t cond);
```

```
// "row < arrpt_size(layout->rows)"
// will be shown in the assert window
cassert(row < arrpt_size(layout->rows));
```

cond Boolean expression.

cassert_msg

Same as the `cassert()` sentence, but using a custom message, instead of the literal condition.

```
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 Boolean expression.

msg Message related to the *assert*.

cassert_fatal

Same as the `cassert()` sentence, but throwing a **critical** assert (not "continuable").

```
void
cassert_fatal(bool_t cond);
```

```
// "gravity > 0."
// will be shown in the assert window
cassert_fatal(gravity > 0.);
```

cond Boolean expression.

cassert_fatal_msg

Same as the `cassert_msg()` sentence, but throwing a **critical** assert (not "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 Boolean expression.

msg Message related to the *assert*.

cassert_no_null

Triggers a critical *assert* if a pointer has `NULL` value.

```
void
cassert_no_null(void *ptr);
```

ptr Pointer to evaluate.

cassert_no_nullf

Triggers a critical *assert* if **a function** pointer has `NULL` value.

```
void
cassert_no_nullf(void *fptr);
```

fptr Pointer to evaluate.

cassert_default

Triggers a "continuable" *assert* if the **switch** statement reaches the `default:` state. Useful to ensure that, for example, all the values of an enum have been considered.

```
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

Set a custom function to execute an alternative code when an *assert* occurs. By default, in desktop applications, an informative window is displayed (Figure 13.4) and the message is saved in a *"Log"* (page 184) file.

```
void
cassert_set_func(void *data,
                 FPtr_assert func_assert);
```

data	User data or application context.
func_assert	*Callback* function called after the activation of an *assert*.

Remarks:

When using this function, the previous *asserts* management will be deactivated.

ptr_get

Access to the content of the pointer (dereference), verifying previously that it is not 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	Pointer.
type	Pointer type.

ptr_dget

Access the content of a double pointer, invalidating it later.

```
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 pointer.

type Pointer type.

ptr_dget_no_null

Like `ptr_dget`, but the content of the double pointer (`*dptr`) can not be `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 pointer.

type Pointer type.

ptr_assign

Assign content from one pointer to another, if the destination is not `NULL`.

```
void
ptr_assign(dest,
           src);
```

dest Destination pointer.

src Source pointer.

ptr_destopt

Destroy an object if not 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 Double pointer to the object to destroy.

type Object type.

ptr_copyopt

Copy the object if not NULL.

```
void
ptr_copyopt(FPtr_copy func_copy,
            type ptr,
            type);
```

```
if (ptr != NULL)
    return func_copy(ptr);
else
    return NULL;
```

func_copy Copy constructor.

ptr Object to copy (source).

type Object type.

unicode_convers

Converts a Unicode string from one encoding to another.

```
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	Source string (terminated in null character '\0').
to_str	Destination buffer.
from	Source string encoding.
to	Coding required in `to_str`.
osize	Size of the output buffer. Maximum number of bytes that will be written in `to_str`, including the null character (`'\0'`). If the original string can not be copied entirety, it will be cutted and the null character added.

Return:

Number of bytes written in `to_str` (including the null character).

unicode_convers_n

Like `unicode_convers`, but indicating a maximum size for the input string.

```
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	Source string.
to_str	Destination buffer.
from	Source string encoding.
to	Coding required in `to_str`.
isize	Size of the input string (in bytes).
osize	Size of the output buffer.

Return:

Number of bytes written in `to_str` (including the null character).

unicode_convers_nbytes

Gets the number of bytes needed to convert a Unicode string from one encoding to another. It will be useful to calculate the space needed in dynamic memory allocation.

```
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	Origin string (null-terminated).
from	Encoding of `str`.
to	Required encoding.

Return:

Number of bytes required (including the null character).

unicode_nbytes

Gets the size (in bytes) of a Unicode string.

```
uint32_t
unicode_nbytes(const char_t *str,
               const unicode_t format);
```

str	Unicode string (null-terminated '\0').
format	Encoding of `str`.

Return:

The size in bytes (including the null character).

unicode_nchars

Gets the length (in characters) of a Unicode string.

```
uint32_t
unicode_nchars(const char_t *str,
               const unicode_t format);
```

str	Unicode string (null-terminated '\0').
format	Encoding of `str`.

Return:

The number of characters ('\0' **not included**).

Remarks:

In ASCII strings, the number of bytes is equal to the number of characters. In Unicode it depends on the coding and the string.

unicode_to_u32

Gets the value of the first *codepoint* of the Unicode string.

```
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	Unicode string (null-terminated '\0').
format	Encoding of `str`.

Return:

The code of the first `str` character.

unicode_to_u32b

Like `unicode_to_u32` but with an additional field to store the number of bytes occupied by the codepoint.

```
uint32_t
unicode_to_u32b(const char_t *str,
                const unicode_t format,
                uint32_t *bytes);
```

str	Unicode string (null-terminated '\0').
format	Encoding of `str`.
bytes	Saves the number of bytes needed to represent the codepoint by `format`.

Return:

The code of the first `str` character.

unicode_to_char

Write the `codepoint` at the beginning of `str`, using the `format` encoding.

```
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	Character code.
str	Destination string.
format	Encoding for `codepoint`.

Return:

The number of bytes written (1, 2, 3 or 4).

Remarks:

To write several *codepoints*, combine `unicode_to_char` with `unicode_next`.

unicode_valid_str

Check if a string is a valid Unicode.

```
bool_t
unicode_valid_str(const char_t *str,
                  const unicode_t format);
```

str String to be checked (ending in '\0').

format Expected Unicode encoding.

Return:

`TRUE` if it is valid.

unicode_valid_str_n

Like `unicode_valid_str`, but indicating a maximum size for the input string.

```
bool_t
unicode_valid_str_n(const char_t *str,
                    const uint32_t size,
                    const unicode_t format);
```

str String to be checked (ending in '\0').

size Maximum size of the string (in bytes).

format Expected Unicode encoding.

Return:

`TRUE` if it is valid.

unicode_valid

Check if a *codepoint* is valid.

```
bool_t
unicode_valid(const uint32_t codepoint);
```

codepoint The Unicode code of the character.

Return:

`TRUE` if the parameter is a valid *codepoint*. `FALSE` otherwise.

unicode_next

Advance to the next character in a Unicode string. In general, random access is not possible as we do in ANSI-C (`str[i ++]`). We must iterate a string from the beginning. More in *"UTF encodingsUTF encodings"* (page 157).

```
const char_t*
unicode_next(const char_t *str,
             const unicode_t format);
```

```
char_t str[] = "ááéíóúÄ";
char_t *iter = str;                  /* iter == "ááéíóúÄ" */
iter = unicode_next(iter, ekUTF8);   /* 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	Unicode string.
format	`str` encoding.

Return:

Pointer to the next character in the string.

Remarks:

It does not verify the end of the string. We must stop the iteration when `codepoint == 0`.

unicode_back

Go back to the previous character of a Unicode string.

```
const char_t*
unicode_back(const char_t *str,
             const unicode_t format);
```

str	Unicode string.
format	`str` encoding.

Return:

Pointer to the previous character of the string.

Remarks:

It does not verify the beginning of the string.

unicode_isascii

Check if `codepoint` is a US-ASCII 7 character.

```
bool_t
unicode_isascii(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

unicode_isalnum

Check if `codepoint` is an alphanumeric character.

```
bool_t
unicode_isalnum(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isalpha

Check if `codepoint` is an alphabetic character.

```
bool_t
unicode_isalpha(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_iscntrl

Check if `codepoint` is a control character.

```
bool_t
unicode_iscntrl(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isdigit

Check if `codepoint` is digit (0-9).

```
bool_t
unicode_isdigit(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isgraph

Check if `codepoint` is a printable character (except white space ' ').

```
bool_t
unicode_isgraph(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isprint

Check if `codepoint` is a printable character (including white space ' ').

```
bool_t
unicode_isprint(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_ispunct

Check if `codepoint` is a printable character (expect white space ' ' and alphanumeric).

```
bool_t
unicode_ispunct(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isspace

Check if `codepoint` is a spacing character, new line, carriage return, horizontal or vertical tab.

```
bool_t
unicode_isspace(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isxdigit

Check if `codepoint` is a hexadecimal digit **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 The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_islower

Check if `codepoint` is a lowercase letter.

```
bool_t
unicode_islower(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_isupper

Check if `codepoint` is a capital letter.

```
bool_t
unicode_isupper(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

Test result.

Remarks:

Only consider US-ASCII characters.

unicode_tolower

Convert a letter to lowercase.

```
uint32_t
unicode_tolower(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

The conversion to lowercase if the entry is a capital letter. Otherwise, the same `codepoint`.

Remarks:

Only consider US-ASCII characters.

unicode_toupper

Convert a letter to uppercase.

```
uint32_t
unicode_toupper(const uint32_t codepoint);
```

codepoint The Unicode character code.

Return:

The conversion to upper case if the entry is a lowercase letter. Otherwise, the same `codepoint`.

Remarks:

Only consider US-ASCII characters.

bmath_cos

Get the cosine of an angle.

```
real32_t
bmath_cosf(const real32_t angle);

real64_t
bmath_cosd(const real64_t angle);

real
BMath::cos(const real angle);
```

angle Angle in radians.

Return:

The cosine of the angle.

bmath_sin

Get the sine of an angle.

```
real32_t
bmath_sinf(const real32_t angle);

real64_t
bmath_sind(const real64_t angle);

real
BMath::sin(const real angle);
```

angle Angle in radians.

Return:

The sine of the angle.

bmath_tan

Get the tangent of an angle.

```
real32_t
bmath_tanf(const real32_t angle);

real64_t
bmath_tand(const real64_t angle);

real
BMath::tan(const real angle);
```

angle Angle in radians.

Return:

The angle tangent.

bmath_acos

Get the cosine arc, or inverse cosine, which is the angle whose cosine is the value.

```
real32_t
bmath_acosf(const real32_t cos);

real64_t
bmath_acosd(const real64_t cos);

real
BMath::acos(const real cos);
```

cos Cosine (-1, 1).

Return:

The angle (0, Pi).

bmath_asin

Get the sine arc, or inverse sine, which is the angle whose sine is the value.

```
real32_t
bmath_asinf(const real32_t sin);

real64_t
bmath_asind(const real64_t sin);

real
BMath::asin(const real sin);
```

sin Sine (-1, 1).

Return:

The angle (0, Pi).

bmath_atan2

Get the tangent arc, or inverse tangent. Es is the angle measured from the X axis to the line containing the origin (0, 0) and the point with the coordinates (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 Y coordinate.

x Coordinate X.

Return:

The angle (-Pi, Pi).

bmath_norm_angle

Normalizes an angle, that is, it returns the same angle expressed in the range (-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 The angle in radians.

Return:

The angle (-Pi, Pi).

bmath_sqrt

Get the square root of a number.

```
real32_t
bmath_sqrtf(const real32_t value);

real64_t
bmath_sqrtd(const real64_t value);

real
BMath::sqrt(const real value);
```

value The number.

Return:

The square root.

bmath_isqrt

Get the inverse square root of a number (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 The number.

Return:

The inverse square root.

bmath_log

Get the natural logarithm (base e) of a number.

```
real32_t
bmath_logf(const real32_t value);

real64_t
bmath_logd(const real64_t value);

real
BMath::log(const real value);
```

value The number.

Return:

The logarithm.

bmath_log10

Get the logarithm in base 10 of a number.

```
real32_t
bmath_log10f(const real32_t value);

real64_t
bmath_log10d(const real64_t value);

real
BMath::log10(const real value);
```

value The number.

Return:

The logarithm.

bmath_exp

Get the number of Euler e (2.7182818) raised to a power.

```
real32_t
bmath_expf(const real32_t value);

real64_t
bmath_expd(const real64_t value);

real
BMath::exp(const real value);
```

value The exponent.

Return:

The exponential.

bmath_pow

Calculate a power, `base` raised to `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 Exponent.

Return:

The result of the power.

bmath_abs

Get the absolute value of a number.

```
real32_t
bmath_absf(const real32_t value);

real64_t
bmath_absd(const real64_t value);

real
BMath::abs(const real value);
```

value The number.

Return:

The absolute value.

bmath_max

Get the maximum of two values.

```
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 First number.

value2 Second number.

Return:

The maximum value.

bmath_min

Get the minimum of two values.

```
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 First number.

value2 Second number.

Return:

The minimum value.

bmath_clamp

Restrict a value to a certain range.

```
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 The number.

min Minimum value of the range.

max Maximum value of the range.

Return:

The limited value.

bmath_mod

Get the module of divide `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 Numerator.

den Denominator.

Return:

The module.

bmath_modf

Get the integer and fraction part of a real number.

```
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 The number.

intpart Get the integer part.

Return:

The fractional part `[0,1)`.

bmath_prec

Get the number of decimals (precision) of a real number.

```
uint32_t
bmath_precf(const real32_t value);

uint32_t
bmath_precd(const real64_t value);

uint32_t
BMath::prec(const real value);
```

value The number.

Return:

The number of decimal places.

bmath_round

Rounds a number to the nearest integer (above or below).

```
real32_t
bmath_roundf(const real32_t value);

real64_t
bmath_roundd(const real64_t value);

real
BMath::round(const real value);
```

value The number.

Return:

The nearest whole.

bmath_round_step

Round a number to the nearest fraction.

```
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 The number.

step The fraction.

Return:

The nearest number.

bmath_floor

Rounds a number to the integer below.

```
real32_t
bmath_floorf(const real32_t value);

real64_t
bmath_floord(const real64_t value);

real
BMath::floor(const real value);
```

value The number.

Return:

The largest integer number, less than or equal to the number.

bmath_ceil

Round a number to the integer above.

```
real32_t
bmath_ceilf(const real32_t value);

real64_t
bmath_ceild(const real64_t value);

real
BMath::ceil(const real value);
```

value The number.

Return:

The smallest integer number, greater than or equal to the number.

bmath_rand_seed

Establish a new seed of random numbers.

```
void
bmath_rand_seed(const uint32_t seed);
```

seed The new seed.

Remarks:

Each time the seed changes, a new sequence of random numbers begins. For the same seed, we will get the same sequence, so they are pseudo-random numbers. Similar seeds (eg. `4, 5`) produce radically different sequences. Use `bmath_rand_env` in multi-threaded applications.

bmath_rand

Gets a random real number, within an interval.

```
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 The lower limit of the interval.

to The upper limit of the interval.

Return:

The random number.

bmath_randi

Gets a random number, within an interval.

```
uint32_t
bmath_randi(const uint32_t from,
            const uint32_t to);
```

from The lower limit of the interval.

to The upper limit of the interval.

Return:

The random number.

bmath_rand_env

Create *thread-safe* environment for random numbers.

```
REnv*
bmath_rand_env(const uint32_t seed);
```

seed The seed.

Return:

The environment.

bmath_rand_destroy

Destroy an environment of random numbers.

```
void
bmath_rand_destroy(REnv **env);
```

env The environment. Will be set to `NULL` after destruction.

bmath_rand_mt

Gets a random real number, within an interval.

```
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 The random number environment.

from The lower limit of the interval.

to The upper limit of the interval.

Return:

The random number.

bmath_rand_mti

Gets a random number, within an interval.

```
uint32_t
bmath_rand_mti(REnv *env,
               const uint32_t from,
               const uint32_t to);
```

env The random number environment.

from The lower limit of the interval.

to The upper limit of the interval.

Return:

The random number.

blib_strlen

Returns the length in bytes of a text string.

```
uint32_t
blib_strlen(const char_t *str);
```

str String terminated with null character '\0'.

Return:

String length not including the null character.

Remarks:

See *"Unicode"* (page 155), the number of bytes is not equivalent to the number of characters.

blib_strstr

Find a substring within a longer string.

```
const char_t*
blib_strstr(const char_t *str,
            const char_t *substr);
```

str String terminated with null character '\0'.

substr Substring to search ending in null character '\0'.

Return:

Pointer to the start of the first substring found or `NULL` if none exists.

blib_strcpy

Copy the content of one string to another.

```
void
blib_strcpy(char_t *dest,
            const uint32_t size,
            const char_t *src);
```

dest Destiny buffer.

size Destination buffer size in bytes.

src String to copy ending in null character '\0'.

Remarks:

Only the first `size-1` bytes will be copied, in case `src` is longer than the capacity of `dest`.

blib_strncpy

Copy the first `n` bytes of one string to another.

```
void
blib_strncpy(const char_t *dest,
             const uint32_t size,
             const char_t *src,
             const uint32_t n);
```

dest Destiny buffer.

size Destination buffer size in bytes.

src String to copy ending in null character '\0'.

n Number of bytes to copy.

Remarks:

Only the first `size-1` bytes will be copied, in case `n` is greater than `size`.

blib_strcat

Concatenation of strings.

```
void
blib_strcat(char_t *dest,
            const uint32_t size,
            const char_t *src);
```

dest Source and destination buffer.

size Destination buffer size in bytes.

src String to add to `dest`, terminated with null character '\0'.

Remarks:

The `size-1` bytes in `dest` will not be exceeded, so the concatenation will be truncated if necessary.

blib_strcmp

Compare two strings.

```
int
blib_strcmp(const char_t *str1,
            const char_t *str2);
```

str1 First string to compare, terminated with null character '\0'.

str2 Second string to compare, terminated with null character '\0'.

Return:

Comparison Result.

blib_strncmp

Compare the first `n` bytes of two strings.

```
int
blib_strncmp(const char_t *str1,
             const char_t *str2,
             const uint32_t n);
```

str1 First string to compare, terminated with null character '\0'.

str2 Second string to compare, terminated with null character '\0'.

n Maximum number of bytes to compare.

Return:

Comparison Result.

blib_strtol

Convert a text string to an integer.

```
int64_t
blib_strtol(const char_t *str,
            char_t **endptr,
            uint32_t base,
            bool_t *err);
```

str — String starting with an integer.

endptr — Pointer whose value will be the first character after the number. Can be NULL.

base — Number base: 2, 8, 10, 16.

err — Value TRUE is assigned if there is an error in the parsing of the string. Can be NULL.

Return:

String parsing result number.

blib_strtoul

Convert a text string to an unsigned integer.

```
uint64_t
blib_strtoul(const char_t *str,
             char_t **endptr,
             uint32_t base,
             bool_t *err);
```

str — String starting with an integer.

endptr — Pointer whose value will be the first character after the number. Can be NULL.

base — Number base: 2, 8, 10, 16.

err — Value TRUE is assigned if there is an error in the parsing of the string. Can be NULL.

Return:

String parsing result number.

blib_strtof

Convert a text string to a 32-bit real number.

```
real32_t
blib_strtof(const char_t *str,
            char_t **endptr,
            bool_t *err);
```

str String starting with an real number.

endptr Pointer whose value will be the first character after the number. Can be NULL.

err Value TRUE is assigned if there is an error in the parsing of the string. Can be NULL.

Return:

String parsing result number.

blib_strtod

Convert a text string to a 32-bit real number.

```
real64_t
blib_strtod(const char_t *str,
            char_t **endptr,
            bool_t *err);
```

str String starting with an real number.

endptr Pointer whose value will be the first character after the number. Can be NULL.

err Value TRUE is assigned if there is an error in the parsing of the string. Can be NULL.

Return:

String parsing result number.

blib_qsort

Sorts a vector of elements using the *QuickSort* algorithm.

```
void
blib_qsort(byte_t *array,
           const uint32_t nelems,
           const uint32_t size,
           FPtr_compare func_compare);
```

array	Vector of elements.
nelems	Number of elements.
size	Size of each element.
func_compare	Comparison function.

blib_qsort_ex

Sorts a vector of elements using the *QuickSort* algorithm.

```
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 of elements.
nelems	Number of elements.
size	Size of each element.
func_compare	Compare function that accepts extra data.
data	Extra data that will be passed in each comparison.

blib_bsearch

Search for an element in an ordered vector.

```
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 of elements.
key	Search key.
nelems	Number of elements.
size	Size of each element.
func_compare	Comparison function.
pos	Position of the found element. It can be NULL.

Return:

TRUE if the element was found.

blib_bsearch_ex

Search for an element in an ordered vector.

```
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 of elements.
key	Search key.
nelems	Number of elements.
size	Size of each element.
func_compare	Compare function that accepts extra data.
data	Extra data that will be passed in each comparison.
pos	Position of the found element. It can be NULL.

Return:

TRUE if the element was found.

blib_atexit

Add a function that will be called when the program ends.

```
void
```

```
blib_atexit(void()(void) *func);
```

func Function.

blib_abort

The execution of the program ends abruptly.

```
void
blib_abort(void);
```

Remarks:

No resources are released or a controlled shutdown is performed. The only case where its use is justified is to exit the program after detecting an unrecoverable error (eg `NULL` pointer).

blib_debug_break

Stops program execution at the point where the function is located and returns debugger control so we can inspect the stack, variables, etc.

```
void
blib_debug_break(void);
```

bstd_sprintf

Write a string with the `printf` format in a memory buffer.

```
uint32_t
bstd_sprintf(char_t *str,
             const uint32_t size,
             const char_t *format,
             ...);
```

str Pointer to the buffer where the result will be written. It will end in a null character `'\0'`.

size Size of `str` in bytes.

format String with the printf-like format with a variable number of parameters.

... Arguments or variables of printf.

Return:

The number of bytes written, not including the null character `'\0'`.

Remarks:

It is a safe function and will not write more than `size` bytes. To obtain the necessary size of `str`, call this function with `str=NULL` and `size=0`.

bstd_vsprintf

Like `bstd_sprintf` but with the list of arguments already resolved.

```
uint32_t
bstd_vsprintf(char_t *str,
              const uint32_t size,
              const char_t *format,
              va_list args);
```

str	Pointer to the buffer where the result will be written. It will end in a null character `'\0'`.
size	Size of `str` in bytes.
format	String with the printf-like format with a variable number of parameters.
args	Arguments.

Return:

The number of bytes written, not including the null character `'\0'`.

Remarks:

It is a safe function and will not write more than `size` bytes.

bstd_printf

Writes a formatted string in the standard output `(stdout)`. It is equivalent to the function `printf` from the standard library.

```
uint32_t
bstd_printf(const char_t *format,
            ...);
```

format	String with the printf-like format with a variable number of parameters.
...	Arguments or variables of printf.

Return:

The number of bytes written in `stdout`.

bstd_eprintf

Writes a formatted string in the error output `(stderr)`.

```
uint32_t
bstd_eprintf(const char_t *format,
             ...);
```

format String with the printf-like format with a variable number of parameters.

... Arguments or variables of printf.

Return:

The number of bytes written in `stderr`.

bstd_writef

Write a string C UTF8 in the standard output `(stdout)`.

```
uint32_t
bstd_writef(const char_t *str);
```

str String C UTF8 ending in null character `'\0'`.

Return:

The number of bytes written in `stdout`.

bstd_ewritef

Write a string C UTF8 on the error output `(stderr)`.

```
uint32_t
bstd_ewritef(const char_t *str);
```

str String C UTF8 ending in null character `'\0'`.

Return:

The number of bytes written in `stderr`.

bstd_read

Read data from standard input `stdin`.

```
bool_t
bstd_read(byte_t *data,
          const uint32_t size,
          uint32_t *rsize);
```

data Buffer where the read data will be written.

size The number of maximum bytes to read (buffer size).

rsize Receive the number of bytes actually read. Can be NULL.

Return:

TRUE if data has been read. FALSE if any error has occurred.

Remarks:

"Standard streamStandard stream" (page 196) implements high-level functions for reading/writing on standard channels.

bstd_write

Write data in the standard output stdout.

```
bool_t
bstd_write(const byte_t *data,
           const uint32_t size,
           uint32_t *wsize);
```

data Buffer that contains the data to write.

size The number of bytes to write.

wsize It receives the number of bytes actually written. Can be NULL.

Return:

TRUE if data has been written. FALSE if any error has occurred.

Remarks:

"Standard streamStandard stream" (page 196) implements high-level functions for reading/writing on standard channels.

bstd_ewrite

Write data in the error output stderr.

```
bool_t
bstd_ewrite(const byte_t *data,
            const uint32_t size,
            uint32_t *wsize);
```

data Buffer that contains the data to write.

size The number of bytes to write.

wsize It receives the number of bytes actually written. Can be `NULL`.

Return:

`TRUE` if data has been written. `FALSE` if any error has occurred.

Remarks:

"Standard streamStandard stream" (page 196) implements high-level functions for reading/writing on standard channels.

bmem_malloc

Reserve a memory block with the default alignment `sizeof(void*)`.

```
byte_t*
bmem_malloc(const uint32_t size);
```

size Size in bytes of the block.

Return:

Pointer to the new block. Must be released with `bmem_free` when it is no longer necessary.

Remarks:

Use *"Heap - Memory manager"* (page 188) for more efficient and secure allocations.

bmem_realloc

Reallocs an existing memory block due to the expansion or reduction of it. Guarantees that the previous content of the block is preserved `min(size, new_size)`. Try to do it without moving memory (in situ), but if it is not possible look for a new zone. It also guarantees the default alignment `sizeof(void*)` if has to reserve a new block.

```
byte_t*
bmem_realloc(byte_t *mem,
             const uint32_t size,
             const uint32_t new_size);
```

mem Pointer to the original block to relocate.

size Size in bytes of the original block `mem`.

new_size New required size, in bytes.

Return:

Pointer to the relocated block. It will be the same as the original pointer `mem` if the relocation "in-situ" has been successful. Must be released with `bmem_free` when it is no longer necessary.

Remarks:

Use *"Heap - Memory manager"* (page 188) for more efficient and secure allocations.

bmem_aligned_malloc

Reserve a memory block with alignment.

```
byte_t*
bmem_aligned_malloc(const uint32_t size,
                    const uint32_t align);
```

size	Size in bytes of the block.
align	Alignment. It must be power of 2.

Return:

Pointer to the new block. Must be released with `bmem_free` when it is no longer necessary.

Remarks:

Use *"Heap - Memory manager"* (page 188) for more efficient and secure allocations.

bmem_aligned_realloc

Like `bmem_realloc`, but it guarantees a specific alignment.

```
byte_t*
bmem_aligned_realloc(byte_t *mem,
                     const uint32_t size,
                     const uint32_t new_size,
                     const uint32_t align);
```

mem	Pointer to the original block to relocate.
size	Size in bytes of the original block `mem`.
new_size	New required size, in bytes.
align	Alignment. It must be power of 2.

Return:

Pointer to the relocated block.

Remarks:

Use *"Heap - Memory manager"* (page 188) for more efficient and secure allocations.

bmem_free

Free memory pointed by `mem`, previously reserved by `bmem_malloc`, `bmem_realloc` or its equivalents with alignment.

```
void
bmem_free(byte_t *mem);
```

mem Pointer to the memory block to be released.

Remarks:

Use *"Heap - Memory manager"* (page 188) for more efficient and secure allocations.

bmem_set1

Fill a block of memory with the same 1-byte mask.

```
void
bmem_set1(byte_t *dest,
          const uint32_t size,
          const byte_t mask);
```

dest Pointer to the memory block.

size Size in bytes of the block `dest`.

mask Mask.

bmem_set4

Fill a block of memory with the same 4-byte mask.

```
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 Pointer to the memory block.

size Size in bytes of the block `dest`. It is not necessary to be a multiple of 4.

mask 4-byte mask.

bmem_set8

Fill a block of memory with the same 8-byte mask.

```
void
bmem_set8(byte_t *dest,
          const uint32_t size,
          const byte_t *mask);
```

dest Pointer to the memory block.

size Size in bytes of the block `dest`. It is not necessary to be a multiple of 8.

mask 8-byte mask.

bmem_set16

Fill a block of memory with the same 16-byte mask.

```
void
bmem_set16(byte_t *dest,
           const uint32_t size,
           const byte_t *mask);
```

dest Pointer to the memory block.

size Size in bytes of the block `dest`. It is not necessary to be a multiple of 16.

mask 16-byte mask.

bmem_set_u32

Fill an array of type `uint32_t` with the same value.

```
void
bmem_set_u32(uint32_t *dest,
             const uint32_t n,
             const uint32_t value);
```

dest Pointer to the array.

n Array size (number of elements).

value Filling value.

bmem_set_r32

Fills an array of type `real32_t` with the same value.

```
void
bmem_set_r32(real32_t *dest,
             const uint32_t n,
             const real32_t value);
```

dest Pointer to the array.

n Array size (number of elements).

value Filling value.

bmem_cmp

Compare two generic memory blocks.

```
int
bmem_cmp(const byte_t *mem1,
         const byte_t *mem2,
         const uint32_t size);
```

mem1 Pointer to the first block of memory.

mem2 Pointer to the second block of memory.

size Number of bytes to compare.

Return:

Comparison result.

bmem_is_zero

Check if a memory block is completely filled with 0s.

```
bool_t
bmem_is_zero(const byte_t *mem,
             const uint32_t size);
```

mem Pointer to the memory block.

size Size in bytes of the block `mem`.

Return:

TRUE if all positions are 0, otherwise FALSE.

bmem_set_zero

Fill a memory block with 0s.

```
void
bmem_set_zero(byte_t *dest,
              const uint32_t size);
```

dest	Pointer to the memory block that must be filled.
size	Size in bytes of the block `dest`.

bmem_zero

Initialize an object with 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	Pointer to the object.
type	Object type.

bmem_zero_n

Initialize an array of objects with 0s.

```
void
bmem_zero_n(type *dest,
            const uint32_t n,
            type);
```

dest Object array.

n Array size.

type Object type.

bmem_copy

Copy the contents of one block in another. The blocks must not be overlapping.

```
void
bmem_copy(byte_t *dest,
          const byte_t *src,
          const uint32_t size);
```

dest Pointer to the destination block.

src Pointer to the source block.

size Number of bytes to copy.

bmem_copy_n

Copy an array of objects to another location.

```
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 Pointer to the destination array.

src Pointer to the source array.

n Array size (number of elements, not bytes).

type Object type.

bmem_move

Like `bmem_copy`, but the blocks can overlap.

```
void
bmem_move(byte_t *dest,
          const byte_t *src,
          const uint32_t size);
```

dest Pointer to the destination block.

src Pointer to the source block.

size Number of bytes to copy.

Remarks:

If we have the certainty that both blocks do not overlap, `bmem_copy` is much more efficient.

bmem_overlaps

Check if two memory blocks overlap.

```
bool_t
bmem_overlaps(byte_t *mem1,
              byte_t *mem2,
              const uint32_t size1,
              const uint32_t size2);
```

mem1 Pointer to the first block.

mem2 Pointer to the second block.

size1 Size of the first block (in bytes).

size2 Size of the second block (in bytes).

Return:

`TRUE` if there is overlap.

bmem_rev

Reverts a memory block `m[i] = m[ni-1]`.

```
void
bmem_rev(byte_t *mem,
         const uint32_t size);
```

mem Pointer to the memory block.

size Block size in bytes.

bmem_rev2

Reverts a 2-byte memory block.

```
void
bmem_rev2(byte_t *mem);
```

mem Pointer to the memory block.

bmem_rev4

Reverts a 4-byte memory block.

```
void
bmem_rev4(byte_t *mem);
```

mem Pointer to the memory block.

bmem_rev8

Reverts an 8-byte memory block.

```
void
bmem_rev8(byte_t *mem);
```

mem Pointer to the memory block.

bmem_revcopy

Make a reverse copy of a memory block.

```
void
bmem_revcopy(byte_t *dest,
             const byte_t *src,
             const uint32_t size);
```

dest Pointer to the destination block.

src Pointer to the source block.

size Number of bytes to copy.

bmem_rev_elems

Reverts the elements inside an array.

```
void
bmem_rev_elems(type*,
               const uint32_t num_elems,
               type);
```

type* Pointer to the beginning of the array.

num_elems Number of elements of the array.

type Object type.

bmem_swap

Exchanges the contents of two memory blocks (not overlapping). At end, `mem1[i] = mem2[i]` and `mem2[i] = mem1[i]`.

```
void
bmem_swap(byte_t *mem1,
          byte_t *mem2,
          const uint32_t size);
```

mem1 Pointer to the first block.

mem2 Pointer to the second block.

size Number of bytes to be exchanged.

bmem_swap_type

Exchange the contents of two objects.

```
void
bmem_swap_type(type *obj1,
               type *obj2,
               type);
```

obj1 First object.

obj2 Second object.

type Object type.

bmem_shuffle

Randomly shuffles a memory block.

```
void
bmem_shuffle(byte_t *mem,
             const uint32_t size,
             const uint32_t esize);
```

mem Pointer to the memory block.

size Block size (number of elements).

esize Size of each element.

Remarks:

This function is based on a pseudo-random number generator. Use `bmath_rand_seed` to change the sequence.

bmem_shuffle_n

Randomly shuffle an object array.

```
void
bmem_shuffle_n(type *array,
               const uint32_t size,
               type);
```

array	Elements array.
size	Number of elements.
type	Object type.

Remarks:

This function is based on a pseudo-random number generator. Use `bmath_rand_seed` to change the sequence.

Chapter 36

Osbs library

36.1. Types and Constants

enum platform_t

Operating systems supported by NAppGUI.

ekWINDOWS	Microsoft Windows.
ekMACOS	Apple macOS.
ekLINUX	GNU/Linux.
ekIOS	Apple iOS.

enum device_t

Device type.

ekDESKTOP	Desktop or laptop computer.
ekPHONE	Phone.
ekTABLET	Tablet.

enum win_t

Microsoft Windows versions.

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	The system is not Windows.

enum endian_t

Represents the *"Byte orderByte order"* (page 206), or how multi-byte data is stored in memory.

ekLITEND	*Little endian*. The lowest byte first.
ekBIGEND	*Big endian*. The highest byte first.

enum week_day_t

Weekday.

ekSUNDAY	Sunday.
ekMONDAY	Monday.
ekTUESDAY	Tuesday.
ekWEDNESDAY	Wednesday.
ekTHURSDAY	Thursday.
ekFRIDAY	Friday.
ekSATURDAY	Saturday.

enum month_t

Month.

ekJANUARY	January.
ekFEBRUARY	February.
ekMARCH	March.
ekAPRIL	April.
ekMAY	May.
ekJUNE	June.
ekJULY	July.
ekAUGUST	August.
ekSEPTEMBER	September.
ekOCTOBER	October.
ekNOVEMBER	November.
ekDECEMBER	December.

enum file_type_t

File type.

ekARCHIVE	Ordinary file.
ekDIRECTORY	Directory.
ekOTHERFILE	Another type of file reserved for the operating system (devices, pipes, etc.)

enum file_mode_t

Different ways to open a file.

ekREAD	Read only.
ekWRITE	Read and write.
ekAPPEND	Writing at the end of the file.

enum file_seek_t

Initial position of the pointer in `bfile_seek`.

ekSEEKSET	Start of file.
ekSEEKCUR	Current pointer position.

`ekSEEKEND` End of file.

enum ferror_t

Error codes manipulating files.

`ekFEXISTS` The file already exists.

`ekFNOPATH` The directory does not exist.

`ekFNOFILE` The file does not exists.

`ekFBIGNAME` The name of the file exceeds the capacity of the buffer to store it.

`ekFNOFILES` There are no more files when we travel through a directory. `bfile_dir_get`.

`ekFNOEMPTY` You are trying to delete a non-empty directory. `hfile_dir_destroy`.

`ekFNOACCESS` The file can not be accessed (possibly due to lack of permissions).

`ekFLOCK` The file is being used by another process.

`ekFBIG` The file is so big. It may appear in functions that can not handle files larger than 4Gb.

`ekFSEEKNEG` Negative position within a file. See `bfile_seek`.

`ekFUNDEF` There is no more information about the error.

`ekFOK` There is no error.

enum perror_t

Error codes working with processes.

`ekPPIPE` Error in the standard I/O channel.

`ekPEXEC` Error when launching the process. Surely the command is invalid.

`ekPOK` There is no error.

enum serror_t

Error code in network communications.

ekSNONET	There is no Internet connection on the device.
ekSNOHOST	Unable to connect to the remote server.
ekSTIMEOUT	The maximum wait time for the connection has been exceeded.
ekSSTREAM	Error in the I/O channel when reading or writing.
ekSUNDEF	There is no more information about the error.
ekSOK	There is no error.

struct Date

Public structure that contains the fields of a time stamp (date + time) for direct access.

```
struct Date
{
    int16_t year;
    uint8_t month;
    uint8_t wday;
    uint8_t mday;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
};
```

year	Year.
month	The month (1-12). month_t.
wday	The day of the week (0-6). week_day_t.
mday	The day of the month (1-31).
hour	The hour (0-23).
minute	The minute (0-59).
second	The second (0-59).

struct Dir

Represents an open directory, by which you can browse. bfile_dir_open.

```
struct Dir;
```

struct File

File handler on disk. bfile_open.

```
struct File;
```

struct Mutex

Mutual exclusion mechanism (**mutex**) used to control concurrent access to a resource. *"LocksLocks"* (page 175).

```
struct Mutex;
```

struct Proc

Represents a running process, with which the main program can communicate using the standard I/O channels. `bproc_exec`.

```
struct Proc;
```

struct DLib

Represents a dynamically loaded library in the process. `dlib_open`.

```
struct DLib;
```

struct Thread

Represents a thread of execution, launched from the main process. `bthread_create`.

```
struct Thread;
```

struct Socket

Handler of network connection. `bsocket_connect`.

```
struct Socket;
```

36.2. Functions

FPtr_thread_main

Prototype of a thread start function *(thread main)*. `bthread_create`.

```
uint32_t
(*FPtr_thread_main)(type *data);
```

data Data passed to the thread *main* function.

Return:

The thread return value.

osbs_start

Start *osbs* library, reserving space for global internal structures.

```
void
osbs_start(void);
```

osbs_finish

Ends *osbs* library, freeing space from global internal structures.

```
void
osbs_finish(void);
```

osbs_platform

Get the operating system in which the application is running.

```
platform_t
osbs_platform(void);
```

Return:

The platform.

osbs_windows

Get the Windows version.

```
win_t
osbs_windows(void);
```

Return:

The Microsoft Windows version.

osbs_endian

Get the *"Byte orderByte order"* (page 206) of the running platform.

```
endian_t
osbs_endian(void);
```

Return:

The byte order of multi-byte data types.

bproc_exec

Launch a new process.

```
Proc*
bproc_exec(const char_t *command,
           perror_t *error);
```

command The command to execute (path and arguments). Eg. `"ls -lh"` or `"C:\Programs\imgresize background.png -w640 -h480"`.

error Error code if the function fails. Can be `NULL`.

Return:

Child process handler that we can use to communicate with him. If the function fails, return `NULL`.

Remarks:

"Multi-processing examplesMulti-processing examples" (page 168).

bproc_close

Close communication with the child process and free resources.

```
void
bproc_close(Proc **proc);
```

proc Process handler. It will be set to `NULL` after closing.

Remarks:

If the process is still running, this function does not finish it. It only closes the communication channel between the parent and child that will continue to run independently. Like any other object, a process must always be closed, even if it has already finished its execution. *"Multi-processing examplesMulti-processing examples"* (page 168).

bproc_cancel

Force the finalization of the process.

```
bool_t
bproc_cancel(Proc *proc);
```

proc Process handler.

Return:

TRUE if the process is finish. FALSE otherwise.

bproc_wait

Wait until the child process finishes.

```
uint32_t
bproc_wait(Proc *proc);
```

proc Process handler.

Return:

The return value of the child process or UINT32_MAX if there is any error.

bproc_finish

Check if the child process is still running.

```
bool_t
bproc_finish(Proc *proc,
             uint32_t *code);
```

proc Process handler.

code The output value of the process (if it has finished). Can be NULL.

Return:

TRUE if the child process has finish, FALSE if not.

Remarks:

This function returns immediately. It does not block the process that calls it.

bproc_read

Read data from the process standard output (stdout).

```
bool_t
bproc_read(Proc *proc,
           byte_t *data,
           const uint32_t size,
           uint32_t *rsize,
           perror_t *error);
```

proc	Process handler.
data	Buffer where the read data will be written.
size	The maximum bytes to read (buffer size).
rsize	Receive the number of bytes actually read. Can be NULL.
error	Error code if the function fails. Can be NULL.

Return:

TRUE if data has been read. FALSE if any error has occurred.

Remarks:

This function will block the parent process until the child writes in its stdout. If there is no data in the channel and the child ends, will return FALSE with rsize = 0 and error = ekPROC_SUCCESS. *"Multi-processing examplesMulti-processing examples"* (page 168).

bproc_eread

Read data from the process error output (stderr).

```
bool_t
bproc_eread(Proc *proc,
            byte_t *data,
            const uint32_t size,
            uint32_t *rsize,
            perror_t *error);
```

proc	Process handler.
data	Buffer where the read data will be written.
size	The maximum bytes to read (buffer size).
rsize	Receive the number of bytes actually read. Can be NULL.
error	Error code if the function fails. Can be NULL.

Return:

`TRUE` if data has been read. `FALSE` if any error has occurred.

Remarks:

This function will block the parent process until the child writes in its `stdout`. If there is no data in the channel and the child ends, will return `FALSE` with `rsize = 0` and `error = ekPROC_SUCCESS`. *"Multi-processing examplesMulti-processing examples"* (page 168).

bproc_write

Write data in the process input channel (`stdin`).

```
bool_t
bproc_write(Proc *proc,
            const byte_t *data,
            const uint32_t size,
            uint32_t *wsize,
            perror_t *error);
```

proc	Process handler.
data	Buffer that contains the data to write.
size	The number of bytes to write.
wsize	It receives the number of bytes actually written. Can be `NULL`.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if data has been written. `FALSE` if any error has occurred.

Remarks:

This function will block the parent process if there is no space in the buffer to complete the write. When the child process reads `stdin` and free space, the writing will be completed and the parent process will continue its execution. *"Multi-processing examplesMulti-processing examples"* (page 168).

bproc_read_close

Close the `stdout` channel of child process.

```
bool_t
bproc_read_close(Proc *proc);
```

proc Process handler.

Return:

TRUE if the channel has been closed. FALSE if it was already closed.

Remarks:

This function allows ignoring the output of the child process, preventing blockages due to channel saturation. *"Launching processesLaunching processes"* (page 167).

bproc_eread_close

Close the stderr channel of child process.

```
bool_t
bproc_eread_close(Proc *proc);
```

proc Process handler.

Return:

TRUE if the channel has been closed. FALSE if it was already closed.

Remarks:

This function allows ignoring the error output of the child process, preventing blockages due to channel saturation. *"Launching processesLaunching processes"* (page 167).

bproc_write_close

Close the stdin channel of child process.

```
bool_t
bproc_write_close(Proc *proc);
```

proc Process handler.

Return:

TRUE if the channel has been closed. FALSE if it was already closed.

Remarks:

Some processes need to read all the stdin content before starting work. When closing the channel, the child process receives the signal EOF *End-Of-File* in stdin. *"Launching processesLaunching processes"* (page 167).

bproc_exit

End the current process (the caller) and all its execution children.

```
void
bproc_exit(const uint32_t code);
```

code The exit code of the process.

bthread_create

Create a new execution thread, which starts in `thmain`.

```
Thread*
bthread_create(FPtr_thread_main thmain,
               type *data,
               type);
```

thmain The thread start function *thread_main*. Shared data can be passed through the *data* pointer.

data Data passed as a parameter to `thmain`.

type Type of `data`.

Return:

Thread handle. If the function fails, return `NULL`.

Remarks:

The thread will run in parallel until `thmain` return or call `bthread_cancel`. *"Throwing threadsThrowing threads"* (page 171).

bthread_current_id

Returns the identifier of the current thread, that is, the one that is running when this function is called.

```
int
bthread_current_id(void);
```

Return:

Thread identifier.

bthread_close

Close the thread handler and free resources.

```
void
bthread_close(Thread **thread);
```

thread Thread handle. It will be put to `NULL` after closing.

Remarks:

If the thread is still running, this function does not finish it. Like any other object, a thread must always be closed, even if it has already finished its execution. *"Throwing threadsThrowing threads"* (page 171).

bthread_cancel

Force a thread termination.

```
void
bthread_cancel(Thread *thread);
```

thread Thread handler.

Remarks:

It is not recommended to call this function. There will be no "clean" exit of the thread. If it is within a critical section, it will not be released. Neither will it release the dynamic memory reserved privately by the thread. The correct way to end a thread of execution is to return `thmain`. Shared variables can be used (*"Mutual exclusion"* (page 175)) to indicate to a thread that it should end cleanly.

bthread_wait

Stops the thread that calls this function until `thread` finishes its execution.

```
uint32_t
bthread_wait(Thread *thread);
```

thread Thread handle to which we must wait.

Return:

The thread return value. If an error occurs, return `UINT32_MAX`.

bthread_finish

Check if the thread is still running.

```
bool_t
bthread_finish(Thread *thread,
               uint32_t *code);
```

thread Thread handler.

code The return value of the *thmain* function (if it has ended). Can be NULL.

Return:

TRUE if the thread has finished, FALSE otherwise.

Remarks:

This function returns immediately.

bthread_sleep

Suspends the execution of the current thread (the one that calls this function) for a certain number of milliseconds.

```
void
bthread_sleep(const uint32_t milliseconds);
```

milliseconds Time interval (in milliseconds) that the suspension will last.

Remarks:

Performs a "passive" suspension, where no "empty loop" will be executed. The thread is dropped by the *scheduler* and reactivated later.

bmutex_create

Creates a mutual exclusion object that allows multiple threads to share the same resource, such as a memory or file area on disk, preventing them from accessing at the same time.

```
Mutex*
bmutex_create(void);
```

Return:

The mutual exclusion handler.

Remarks:

"Threads" (page 170), *"Multi-thread exampleMulti-thread example"* (page 172).

bmutex_close

Close the mutual exclusion object and free memory.

```
void
bmutex_close(Mutex **mutex);
```

mutex The mutual exclusion handler. It will be set to NULL after closing.

Remarks:

"Threads" (page 170), *"Multi-thread exampleMulti-thread example"* (page 172).

bmutex_lock

Marks the start of a critical section, blocking access to a shared resource. If another thread tries to block, it will be stopped until the current thread calls bmutex_unlock.

```
void
bmutex_lock(Mutex *mutex);
```

mutex The mutual exclusion handler.

Remarks:

"Threads" (page 170), *"Multi-thread exampleMulti-thread example"* (page 172).

bmutex_unlock

Mark the end of a critical section, unlocking access to a shared resource. If another thread is waiting, access will be allowed to its critical section and, therefore, to the shared resource.

```
void
bmutex_unlock(Mutex *mutex);
```

mutex The mutual exclusion handler.

Remarks:

To avoid unnecessary delays, the time between bmutex_lock and bmutex_unlock should be as short as possible. Any calculation that the thread can make in its private memory space must precede the call to bmutex_lock. *"Threads"* (page 170), *"Multi-thread exampleMulti-thread example"* (page 172).

dlib_open

Load a dynamic library at runtime.

```
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 Directory where the library is located. Can be `NULL`.

libname Library name. It must be the "plain" name without prefixes, suffixes or extensions specific to each operating system.

Return:

Pointer to library or `NULL` if failed to load.

Remarks:

If `path` is `NULL`, the library search strategy of each operating system will be followed. See *"Library search pathsLibrary search paths"* (page 176).

dlib_close

Close a previously opened library with `dlib_open`.

```
void
dlib_close(DLib **dlib);
```

dlib Pointer to the library. Will be set to `NULL` upon destruction.

dlib_proc

Get a pointer to a library method.

```
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 Library.

procname Method name.

type Method type. Needed to convert from a generic pointer.

Return:

Pointer to method.

dlib_var

Get a pointer to a library variable.

```
type*
dlib_var(DLib *lib,
         const char_t *varname,
         type);
```

```
const V2Df *vzero = dlib_var(lib, "kV2D_ZEROf", V2Df);
```

lib Library.

varname Variable name.

type Variable type.

Return:

Pointer to variable.

bfile_dir_work

Gets the current working directory of the process. It is the directory from which the relative *pathnames* will be interpreted.

```
uint32_t
bfile_dir_work(char_t *pathname,
               const uint32_t size);
```

pathname Buffer where the directory will be written.

size Size in bytes of the buffer `pathname`.

Return:

The number of bytes written in `pathname`, including the null character `'\0'`.

Remarks:

"Filename and pathnameFilename and pathname" (page 178)

bfile_dir_set_work

Change the current working directory of the application. The relative *pathnames* will be interpreted from here.

```
bool_t
bfile_dir_set_work(const char_t *pathname,
                   ferror_t *error);
```

pathname The name of the directory.

error Error code if the function fails. Can be NULL.

Return:

TRUE if the working directory has changed, FALSE if there have been any errors.

Remarks:

"Filename and pathnameFilename and pathname" (page 178)

bfile_dir_home

Get the home directory of the current user.

```
uint32_t
bfile_dir_home(char_t *pathname,
               const uint32_t size);
```

pathname Buffer where the directory will be written.

size Size in bytes of the buffer pathname.

Return:

The number of bytes written in pathname, including the null character '\0'.

Remarks:

"Filename and pathnameFilename and pathname" (page 178)

bfile_dir_data

Gets the *AppData* directory where application configuration data can be saved.

```
uint32_t
bfile_dir_data(char_t *pathname,
               const uint32_t size);
```

pathname Buffer where the directory will be written.

size Size in bytes of the buffer `pathname`.

Return:

The number of bytes written in `pathname`, including the null character `'\0'`.

Remarks:

"Home and AppDataHome and AppData" (page 179)

bfile_dir_exec

Gets the absolute *pathname* of the current executable.

```
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 Buffer where the directory will be written.

size Size in bytes of the buffer `pathname`.

Return:

The number of bytes written in `pathname`, including the null character `'\0'`.

bfile_dir_create

Create a new directory. It will fail if any intermediate directory of `pathname` does not exist.

```
bool_t
bfile_dir_create(const char_t *pathname,
                 ferror_t *error);
```

pathname Name of the directory to be created, ending in a null character `'\0'`.

error Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if the directory has been created, `FALSE` if there have been any errors.

Remarks:

`hfile_dir_create` create all intermediate directories at once.

bfile_dir_open

Open a directory to browse its contents. Then you have to use `bfile_dir_get` to iterate. The *filename* is not ordered under any criteria. At the end, you should call `bfile_dir_close`.

```
Dir*
bfile_dir_open(const char_t *pathname,
               ferror_t *error);
```

pathname Name of the directory, ending in a null character `'\0'`.

error Error code if the function fails. Can be `NULL`.

Return:

The directory handler or `NULL` if there has been an error.

bfile_dir_close

Close a previously open directory with `bfile_dir_open`.

```
void
bfile_dir_close(Dir **dir);
```

dir The directory handler. It will be set to `NULL` after the closing.

bfile_dir_get

Gets the attributes of the current file when we go through a directory. Previously we have to open the directory with `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	Open directory handler.
filename	Here will write the name of the file or sub-directory, ending in a null character `'\0'` and without including any path. Can be `NULL`.
size	Size in bytes of the `name` buffer.
type	Get the file type. Can be `NULL`.
fsize	Gets the file size in bytes. Can be `NULL`.
updated	Gets the date of the last update of the file. Can be `NULL`.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if the file attributes have been read correctly. When there are no more files to go, it returns `FALSE` with `error=ekFNOFILES`.

Remarks:

This function will advance to the next file within the open directory after obtaining the current item's data. If there is not enough space in `name`, will return `FALSE` with `error =ekFBIGNAME` and will not advance to the next file. Use `hfile_dir_loop` to browse the contents of a directory more comfortably.

bfile_dir_delete

Delete a directory. It will fail if the directory is not completely empty. Use `hfile_dir_destroy` to completely and recursively erase a directory that may have content.

```
bool_t
bfile_dir_delete(const char_t *pathname,
                 ferror_t *error);
```

pathname	Name of the directory, ending in a null character `'\0'`.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE`if the directory has been deleted, `FALSE` otherwise.

bfile_create

Create a new file. If previously it already exists its content will be erased. The new file will be opened for writing.

```
File*
bfile_create(const char_t *pathname,
             ferror_t *error);
```

pathname File name including its absolute or relative path.

error Error code if the function fails. Can be NULL.

Return:

The file handler or NULL if there has been an error.

bfile_open

Open an existing file. Do not create it, if file does not exist this function will fail.

```
File*
bfile_open(const char_t *pathname,
           const file_mode_t mode,
           ferror_t *error);
```

pathname File name including its absolute or relative path.

mode Opening mode.

error Error code if the function fails. Can be NULL.

Return:

The file handler or NULL if there has been an error.

bfile_close

Close a file previously opened with bfile_create or bfile_open.

```
void
bfile_close(File **file);
```

file File handler. It will be set to NULL after closing.

bfile_lstat

Get the attributes of a file through its *pathname*.

```
bool_t
bfile_lstat(const char_t *pathname,
            file_type_t *type,
            uint64_t *fsize,
            Date *updated,
```

```
            ferror_t *error);
```

pathname File name including its absolute or relative path.

type Get the file type. Can be NULL.

fsize Gets the file size in bytes. Can be NULL.

updated Gets the date of the last update of the file. Can be NULL.

error Error code if the function fails. Can be NULL.

Return:

TRUEif it worked correctly, or FALSE otherwise.

bfile_fstat

Get the attributes of a file through its handler.

```
bool_t
bfile_fstat(File *file,
            file_type_t *type,
            uint64_t *fsize,
            Date *updated,
            ferror_t *error);
```

file File manager.

type Get the file type. Can be NULL.

fsize Gets the file size in bytes. Can be NULL.

updated Gets the date of the last update of the file. Can be NULL.

error Error code if the function fails. Can be NULL.

Return:

TRUEif it worked correctly, or FALSE otherwise.

bfile_read

Read data from an open file.

```
bool_t
bfile_read(File *file,
           byte_t *data,
           const uint32_t size,
           uint32_t *rsize,
           ferror_t *error);
```

file	File handler.
data	Buffer where the read data will be written.
size	The number of maximum bytes to read.
rsize	Receive the number of bytes actually read. Can be NULL.
error	Error code if the function fails. Can be NULL.

Return:

TRUE if the data has been read correctly. If there is no more data (end of the file) it returns FALSE with `rsize = 0` and `error=ekFOK`.

Remarks:

"File streamFile stream" (page 194) implements high-level functions for reading/writing files.

bfile_write

Write data in an open file.

```
bool_t
bfile_write(File *file,
            const byte_t *data,
            const uint32_t size,
            uint32_t *wsize,
            ferror_t *error);
```

file	File handler.
data	Buffer that contains the data to write.
size	The number of bytes to write.
wsize	It receives the number of bytes actually written. Can be NULL.
error	Error code if the function fails. Can be NULL.

Return:

TRUE if the data has been written, or FALSE if there have been any errors.

Remarks:

"File streamFile stream" (page 194) implements high-level functions for reading/writing files.

bfile_seek

Move a file pointer to a new location.

```
bool_t
bfile_seek(File *file,
           const int64_t offset,
           const file_seek_t whence,
           ferror_t *error);
```

file	File handler.
offset	Number of bytes to move the pointer. Can be negative.
whence	Pointer position from which `offset` will be added.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if it worked correctly, `FALSE` if not.

Remarks:

It will return `FALSE` and `error` `ekFSEEKNEG` if the final pointer position is negative. It is not an error to set a pointer to a position beyond the end of the file. The file size does not increase until it is written to. A write operation increases the size of the file to the pointer position plus the size of the write buffer. Intermediate bytes would be left undetermined.

bfile_pos

Return the current position of the file pointer.

```
uint64_t
bfile_pos(const File *file);
```

file	File handler.

Return:

Position from start of file.

bfile_delete

Delete a file from the file system.

```
bool_t
bfile_delete(const char_t *pathname,
             ferror_t *error);
```

pathname	File name including its absolute or relative path.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if the file has been deleted, or `FALSE` if any error has occurred.

bsocket_connect

Create a client socket and try to establish a connection to a remote server.

```
Socket*
bsocket_connect(const uint32_t ip,
                const uint16_t port,
                const uint32_t timeout_ms,
                serror_t *error);
```

ip	The 32-bit IPv4 address of the remote host. `bsocket_str_ip`.
port	The connection port.
timeout_ms	Maximum number of milliseconds to wait to establish connection. If it is `0` it will wait indefinitely.
error	Error code if the function fails. Can be `NULL`.

Return:

Socket handle, or `NULL` if the function fails.

Remarks:

The process will be blocked until a response is obtained from the server or the `timeout` is fulfilled. See *"Client/Server exampleClient/Server example"* (page 180).

bsocket_server

Create a server socket.

```
Socket*
bsocket_server(const uint16_t port,
               const uint32_t max_connect,
               serror_t *error);
```

port	The port where the server will "listen".
max_connect	The maximum number of connections can queue.
error	Error code if the function fails. It can be `NULL`.

Return:

Socket handle, or NULL if the function fails.

Remarks:

Client requests will be stored in a queue until a call to bsocket_accept is received. See *"Client/Server exampleClient/Server example"* (page 180).

bsocket_accept

Accepts a connection to the server created with bsocket_server and starts the conversation with the client.

```
Socket*
bsocket_accept(Socket *socket,
               const uint32_t timeout_ms,
               serror_t *error);
```

socket	Handler returned by bsocket_server.
timeout_ms	Maximum number of milliseconds to wait to receive the request. If it is 0 it will wait indefinitely.
error	Error code if the function fails. It can be NULL.

Return:

Socket handle, or NULL if the function fails.

Remarks:

The process will be blocked until a request is obtained from a client or the timeout is fulfilled. See *"Client/Server exampleClient/Server example"* (page 180).

bsocket_close

Close a previously created socket with bsocket_connect, bsocket_server or bsocket_accept .

```
void
bsocket_close(Socket **socket);
```

socket	The socket handler. It will be set to NULL after closing.

bsocket_local_ip

Get the local ip address and port associated with the socket.

```
void
bsocket_local_ip(Socket *socket,
                 uint32_t *ip,
                 uint16_t *port);
```

socket	Socket handle.
ip	Local IP address.
port	Local IP port.

bsocket_remote_ip

Get the IP address and the remote port associated with the other interlocutor of the connection.

```
void
bsocket_remote_ip(Socket *socket,
                  uint32_t *ip,
                  uint16_t *port);
```

socket	Socket handle.
ip	Remote IP address.
port	Remote IP port.

bsocket_read_timeout

Sets the maximum time to wait for the function `bsocket_read`.

```
void
bsocket_read_timeout(Socket *socket,
                     const uint32_t timeout_ms);
```

socket	Socket handle.
timeout_ms	Maximum number of milliseconds to wait for the caller to write data to the channel. If it is `0` it will wait indefinitely.

bsocket_write_timeout

Sets the maximum time to wait for the function `bsocket_write`.

```
void
bsocket_write_timeout(Socket *socket,
                      const uint32_t timeout_ms);
```

socket	Socket handle.
timeout_ms	Maximum number of milliseconds that will wait until the caller reads the data and unblocked on the channel. If it is `0` it will wait indefinitely.

bsocket_read

Read data from the socket.

```
bool_t
bsocket_read(Socket *socket,
             byte_t *data,
             const uint32_t size,
             uint32_t *rsize,
             serror_t *error);
```

socket	Socket handle.
data	Buffer where the read data will be written.
size	The number of maximum bytes to read (buffer size).
rsize	Receive the number of bytes actually read. Can be `NULL`.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if data has been read. `FALSE` if any error has occurred.

Remarks:

The process will be blocked until the interlocutor writes data to the channel or the timeout expires. See `bsocket_read_timeout`.

bsocket_write

Write data in the socket.

```
bool_t
bsocket_write(Socket *socket,
              const byte_t *data,
              const uint32_t size,
              uint32_t *wsize,
              serror_t *error);
```

socket	Socket handle.
data	Buffer that contains the data to write.
size	The number of bytes to write.
wsize	It receives the number of bytes actually written. Can be `NULL`.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if data has been written. `FALSE` if any error has occurred.

Remarks:

The process will be blocked if the channel is full until the interlocutor reads the data and unblocks or expires the timeout. See `bsocket_write_timeout`.

bsocket_url_ip

Get the IPv4 address of a host from its 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	The host url, eg. `www.google.com`.
error	Error code if the function fails. Can be `NULL`.

Return:

Value of the host's IPv4 address or `0` if there has been an error.

bsocket_str_ip

Get the IPv4 address from a string of type `"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 The string with the IP.

Return:

Value of the IPv4 address in 32-bit binary format.

bsocket_host_name

Gets the name of the host.

```
const char_t*
bsocket_host_name(char_t *buffer,
                  const uint32_t size);
```

buffer Buffer to store the name.

size Size of `buffer`.

Return:

Pointer to the string `buffer`.

bsocket_host_name_ip

Gets the host name from its IP.

```
const char_t*
bsocket_host_name_ip(uint32_t ip,
                     char_t *buffer,
                     const uint32_t size);
```

ip Value of the IPv4 address in 32-bit binary format.

buffer Buffer to store the name.

size Size of `buffer`.

Return:

Pointer to the string `buffer`.

bsocket_ip_str

Gets the IP address in text string format.

```
const char_t*
bsocket_ip_str(uint32_t ip,
               const char_t *ip);
```

ip Value of the IPv4 address in 32-bit binary format.

ip The string with the IP.

Return:

String of type “192.168.1.1”.

Remarks:

The string is returned in an internal buffer that will be overwritten on the next call. Make a copy of the string if we need it to be persistent.

bsocket_hton2

Change the “endianness” of a 16bit value prior to being sent through the 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 Destination buffer (at least 2 bytes).

src Buffer (variable).

bsocket_hton4

Same as `bsocket_hton2`, for 4-byte values.

```
void
bsocket_hton4(byte_t *dest,
              const byte_t *src);
```

dest Destination buffer (at least 4 bytes).

src Buffer (variable).

bsocket_hton8

Same as bsocket_hton2, for 8-byte values.

```
void
bsocket_hton8(byte_t *dest,
              const byte_t *src);
```

dest Destination buffer (at least 8 bytes).

src Buffer (variable).

bsocket_ntoh2

Change the "endianness" of a 16bit value after being received by the 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 16-bit destination buffer (variable).

src Buffer received by socket.

bsocket_ntoh4

Same as bsocket_ntoh2, for 4-byte values.

```
void
bsocket_ntoh4(byte_t *dest,
              const byte_t *src);
```

dest Buffer (variable) destination 32bits.

src Buffer received by socket.

bsocket_ntoh8

Same as bsocket_ntoh2, for 8-byte values.

```
void
bsocket_ntoh8(byte_t *dest,
              const byte_t *src);
```

dest Buffer (variable) destination 64bits.

src Buffer received by socket.

btime_now

Gets the number of micro-seconds elapsed since January 1, 1970 until this precise moment. Use the difference between instants to know the time consumed by a process.

```
uint64_t
btime_now(void);
```

Return:

The number of micro-seconds elapsed, that is, the number of intervals of 1/1000000 seconds.

Remarks:

The initial instant is January 1, 1970 in Unix/Linux systems and January 1, 1601 in Windows since it is the first year of the Gregorian cycle in which Windows NT was activated. This function equates both starts, always returning the Unix time.

btime_date

Gets the current system date.

```
void
btime_date(Date *date);
```

date Current date.

btime_to_micro

Convert a date to Unix Time.

```
uint64_t
btime_to_micro(const Date *date);
```

date The date to convert.

Return:

The number of micro-seconds since January 1, 1970 UTC.

btime_to_date

Transform Unix Time into a date.

```
void
btime_to_date(const uint64_t micro,
              Date *date);
```

micro Number of micro-seconds since January 1, 1970 UTC.

date Result date.

log_printf

Write a message in the *log*, with the `printf` format.

```
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 String with the printf-like format with a variable number of parameters.

... Arguments or variables of printf.

Return:

The number of bytes written.

log_output

It establishes whether the content of the *log* will be redirected or not to the standard output.

```
void
log_output(const bool_t std,
           const bool_t err);
```

std If `TRUE` the lines will be sent to the standard output `stdout`. Default, `TRUE`.

err If `TRUE` the lines will be sent to the error output `stderr`. Default, `FALSE`.

log_file

Set a destination file, where the *log* lines will be written.

```
void
log_file(const char_t *pathname);
```

pathname File name including its absolute or relative path. If the file does not exist it will be created and if it already exists, future lines will be added at the end of it. If `NULL` writing to *log* file will be disabled.

log_get_file

Gets the current file associated with the *log*.

```
const char_t*
log_get_file(void);
```

Return:

The absolute *pathname* of the file.

Chapter 37

Core library

37.1. Types and Constants

DeclSt

Given a `struct`, enable macros for compile-time type checking in *"Arrays"* (page 208) and *"Binary search trees"* (page 217). Usage: `DeclSt(Product)` immediately after the `struct` definition. See *"Registers or pointersRegisters or pointers"* (page 209).

DeclPt

Same as `DeclSt` for pointer containers.

kSTDIN

Stream connected to the standard input `stdin`.

```
Stream* kSTDIN;
```

kSTDOUT

Stream connected to standard output `stdout`.

```
Stream* kSTDOUT;
```

kSTDERR

Stream connected to error output `stderr`.

```
Stream* kSTDERR;
```

kDEVNULL

Null write stream. All content sent through this channel will be ignored.

```
Stream* kDEVNULL;
```

kDATE_NULL

Represents an invalid date.

```
Date kDATE_NULL;
```

enum core_event_t

Event types in *core* library.

ekEASSERT	Redirection of *"Asserts"* (page 153).
ekEFILE	A file detected while browsing a directory. hfile_dir_loop .
ekEENTRY	Entry in a sub-directory while we go through a directory. hfile_dir_loop.
ekEEXIT	Exit of a sub-directory.

enum sstate_t

"Streams" (page 193) state.

ekSTOK	All ok, no errors.
ekSTEND	No more data on the channel.
ekSTCORRUPT	The data in the channel is invalid or has not been read correctly.
ekSTBROKEN	Error in the communication channel.

enum vkey_t

Keyboard codes. See *"Using the keyboardUsing the keyboard"* (page 319).

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

Modifier keys.

ekMKEY_NONE
ekMKEY_SHIFT
ekMKEY_CONTROL
ekMKEY_ALT
ekMKEY_COMMAND

enum token_t

Token types on `stm_read_token`.

ekTSLCOM	One-line comment, which begins with `//`.
ekTMLCOM	Multi-line commentary, enclosed between `/*` and `*/`.
ekTSPACE	Represents a series of blanks (`' '`, `'\t'`, `'\v'`, `'\f'`, `'\r'`).
ekTEOL	Represents the new line character (`'\n'`).
ekTLESS	Less than sign `'<'`.
ekTGREAT	Greater than sign `'>'`.
ekTCOMMA	Comma sign `','`.
ekTPERIOD	Point sign `'.'`.
ekTSCOLON	Semicolon sign `';'`.
ekTCOLON	Colon sign `':'`.
ekTOPENPAR	Opening parenthesis `'('`.
ekTCLOSPAR	Closing parenthesis `')'`.
ekTOPENBRAC	Opening bracket `'['`.
ekTCLOSBRAC	Closing bracket `']'`.
ekTOPENCURL	Opening curly bracket `'{'`.
ekTCLOSCURL	Closing curly bracket `'}'`.
ekTPLUS	Plus sign `'+'`.
ekTMINUS	Minus sign `'-'`.
ekTASTERK	Asterisk sign `'*'`.
ekTEQUALS	Equal sign `'='`.
ekTDOLLAR	Dollar sign.

ekTPERCEN	Percentage sign `'%'`.
ekTPOUND	Pound sign `'#'`.
ekTAMPER	Ampersand sign `'&'`.
ekTAPOST	Apostrophe sign `'''`.
ekTQUOTE	Quotation sign `'"'`.
ekTCIRCUM	Circumflex accent sign `'^'`.
ekTTILDE	Tilde sign `'~'`.
ekTEXCLA	Exclamation sign `'!'`.
ekTQUEST	Question mark `'?'`.
ekTVLINE	Vertical bar sign `'|'`.
ekTSLASH	Slash bar sign `'/'`.
ekTBSLASH	Backslash sign `'\'`.
ekTAT	At sign `'@'`.
ekTINTEGER	Integer number. *"NumbersNumbers"* (page 202).
ekTOCTAL	Octal number. *"NumbersNumbers"* (page 202).
ekTHEX	Hexadecimal number. *"NumbersNumbers"* (page 202).
ekTREAL	Real number. *"NumbersNumbers"* (page 202).
ekTSTRING	Unicode character string, enclosed in quotation marks. *"StringsStrings"* (page 201).
ekTIDENT	Identifier. *"IdentifiersIdentifiers"* (page 200).
ekTUNDEF	Unknown token.
ekTCORRUP	Error in the input *"Streams"* (page 193) or data.
ekTEOF	End of the *"Streams"* (page 193) or data. No more tokens.
ekTRESERVED	Keywords. Being of general purpose, the analyzer does not label any identifier as a reserved word. It must be done in phases after the analysis.

struct Buffer

Block of memory of general purpose, reserved dynamically. Once created, you can no longer resize. *"Buffers"* (page 192).

```
struct Buffer;
```

struct String

UTF8 character string reserved dynamically. They are "partially mutable" objects. The reserved memory can not grow, but characters can be substituted as long as the buffer's initial capacity does not overflow. *"Strings"* (page 192).

```
struct String;
```

struct Stream

Generic input/output channel, where it is possible to read and write formatted data. *"Streams"* (page 193).

```
struct Stream;
```

struct ArrSt

Array of records. The type of object is indicated in parentheses. *"Arrays"* (page 208).

```
struct ArrSt;
```

struct ArrPt

Pointers array. The type of object is indicated in parentheses. *"Arrays (pointers)"* (page 217).

```
struct ArrPt;
```

struct SetSt

Set of records. The type of object is indicated in parentheses. *"Binary search trees"* (page 217).

```
struct SetSt;
```

struct SetPt

Pointers set. The type of object is indicated in parentheses. *"Binary search trees (pointers)"* (page 222).

```
struct SetPt;
```

struct RegEx

Regular expresion. *"Regular expressions"* (page 222).

```
struct RegEx;
```

struct Event

Contains information regarding an event. *"Events"* (page 230).

```
struct Event;
```

struct KeyBuf

Keyboard buffer with the state of each key (pressed/released). *"Keyboard buffer"* (page 231).

```
struct KeyBuf;
```

struct Listener

Link to the generator and receiver of an event through a *callback* function *"Events"* (page 230).

```
struct Listener;
```

struct IListener

C++ interface for use class members as event handlers. *"Use of C++"* (page 45).

```
struct IListener;
```

struct DirEntry

Directory element, obtained by `hfile_dir_list`.

```
struct DirEntry
{
    String* name;
    file_type_t type;
    uint64_t size;
    Date date;
};
```

name	File or subdirectory name, without path.
type	Item type.
size	Size in bytes.
date	Date of last modification.

struct EvFileDir

Parameters of the event ekEFILE and ekEENTRY during automatic directory browsing. hfile_dir_loop.

```
struct EvFileDir
{
    const char_t* pathname;
    uint32_t level;
};
```

pathname	The partial path from the parameter pathname of hfile_dir_loop.
level	The depth of the directory from pathname.

struct ResPack

Package of resources that will be loaded together. Use ResId to access a specific resource. *"Resources"* (page 129).

```
struct ResPack;
```

struct ResId

Identifier of a resource. They are generated automatically by *nrc NAppGUI Resource Compiler*. *"Resources"* (page 129).

```
struct ResId;
```

struct Clock

It measures the time elapsed between two instants within the application, with micro-seconds precision. It is also useful for launching events at regular intervals of time.

```
struct Clock;
```

37.2. Functions

FPtr_remove

Releases the memory of an object's fields, but not the object itself. *"Registers or pointer-sRegisters or pointers"* (page 209).

```
void
(*FPtr_remove)(type *obj);
```

obj Pointer to the object whose fields must be released.

FPtr_event_handler

Event handler. They are *callback* functions that will be called by the generator of an event when it happens. *"Events"* (page 230).

```
void
(*FPtr_event_handler)(type *obj,
                      Event *event);
```

obj General data passed as the first parameter of the function.

event The event.

FPtr_read

Create an object from data read from a *"Streams"* (page 193). *"SerializationSerialization"* (page 213).

```
type*
(*FPtr_read)(Stream *stream);
```

stream The I/O channel where the object is serialized.

Return:

The created object, deserializing the stream data.

FPtr_read_init

Similar to `FPtr_read` where the memory of the object has already been reserved, but not initialized. *"SerializationSerialization"* (page 213).

```
void
(*FPtr_read_init)(Stream *stream,
                  type *obj);
```

stream The I/O channel where the object is serialized.

obj The object whose fields must be deserialized.

FPtr_write

Write an object in a *"Streams"* (page 193). *"SerializationSerialization"* (page 213).

```
void
(*FPtr_write)(Stream *stream,
              const type *obj);
```

stream The I/O channel where serialize the object.

obj The object to write.

core_start

Start the *core* library, reserving space for the global internal structures. Internally calls `osbs_start`.

```
void
core_start(void);
```

core_finish

Ends the *core* library, freeing the space of the global internal structures. Internally calls `osbs_finish`.

```
void
core_finish(void);
```

heap_start_mt

Start a multi-threaded section.

```
void
heap_start_mt(void);
```

Remarks:

See *"Multi-thread memoryMulti-thread memory"* (page 189).

heap_end_mt

End a multi-thread section.

```
void
heap_end_mt(void);
```

Remarks:

See *"Multi-thread memoryMulti-thread memory"* (page 189).

heap_verbose

Enable/disable memory auditor 'verbose' mode.

```
void
heap_verbose(bool_t verbose);
```

verbose TRUE to activate.

Remarks:

By default FALSE.

heap_stats

Enable/disable memory auditor statistics.

```
void
heap_stats(bool_t stats);
```

stats TRUE to activate.

Remarks:

By default TRUE.

heap_leaks

Returns TRUE if there are memory leaks at the end of execution.

```
bool_t
heap_leaks(void);
```

Return:

TRUE if leaks exist.

heap_malloc

Reserve a memory block with the default alignment `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 Size in bytes of the block.

name Reference text for the auditor.

Return:

Pointer to the new block. Must be released with `heap_free` when it is no longer necessary.

Remarks:

Use this function for generic blocks. For types use `heap_new`.

heap_calloc

Like `heap_malloc`, but initializing the block with 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 Size in bytes of the block.

name Reference text for the auditor.

Return:

Pointer to the new block. Must be released with `heap_free` when it is no longer necessary.

Remarks:

Use this function for generic blocks. For types use `heap_new`.

heap_realloc

Reallocs an existing memory block due to the expansion or reduction of it. Guarantees that the previous content of the block is preserved `min(size, new_size)`. Try to do it without moving memory (in situ), but if it is not possible look for a new zone. It also guarantees the default alignment `sizeof(void*)` if you have to reserve a new block.

```
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	Pointer to the original block to relocate.
size	Size in bytes of the original block `mem`.
new_size	New required size, in bytes.
name	Reference text for the auditor. It must be the same as the one used in `heap_malloc`.

Return:

Pointer to the relocated block. It will be the same as the original pointer `mem` if the relocation "in-situ" has been successful. Must be released with `heap_free` when it is no longer necessary.

Remarks:

Use this function for generic blocks. For types use `heap_realloc_n`.

heap_aligned_malloc

Reserve a memory block with alignment.

```
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 Size in bytes of the block.

align Alignment. It must be power of 2.

name Reference text for the auditor.

Return:

Pointer to the new block. Must be released with `heap_free` when it is no longer necessary.

heap_aligned_calloc

Like `heap_aligned_malloc` , but initializing the block with 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 Size in bytes of the block.

align Alignment. It must be power of 2.

name Reference text for the auditor.

Return:

Pointer to the new block. Must be released with `heap_free` when it is no longer necessary.

heap_aligned_realloc

Like `heap_realloc`, but guaranteeing memory alignment.

```
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	Pointer to the original block to relocate.
size	Size in bytes of the original block `mem`.
new_size	New required size, in bytes.
align	Alignment. It must be power of 2.
name	Texto reference for the auditor. It must be the same as the one used in `heap_aligned_malloc`.

Return:

Pointer to the relocated block. Must be released with `heap_free` when it is no longer necessary.

heap_free

Free memory pointed by `mem`, previously reserved by `heap_malloc`, `heap_realloc` or its equivalents with alignment.

```
void
heap_free(byte_t **mem,
          const uint32_t size,
          const char_t *name);
```

mem	Double pointer to the block to be released. It will be set to `NULL` after the release.
size	Memory block size.
name	Reference text for the auditor, must be the same as that used in `heap_malloc`.

Remarks:

Use this function for generic memory blocks. For types it uses `heap_delete`.

heap_new

Reserve memory for an object. The return pointer is converted to `type`.

```
type*
heap_new(type);
```

```
MyAppCtrl *ctrl = heap_new(MyAppCtrl);
...
heap_delete(&ctrl, MyAppCtrl);
```

type Object type.

Return:

Pointer to the created object. It must be destroyed by `heap_delete` when it is no longer necessary.

heap_new0

Like `heap_new`, but initializing the object with 0s.

```
type*
heap_new0(type);
```

```
MyAppModel *model = heap_new0(MyAppModel);
/* model = {0} */
...
heap_delete(&model, MyAppModel);
```

type Object type.

Return:

Pointer to the created object. It must be destroyed by `heap_delete` when it is no longer necessary.

heap_new_n

Reserve memory for `n` objects. The return pointer is converted to `type`.

```
type*
heap_new_n(const uint32_t n,
           type);
```

```
Car *cars = heap_new_n(10, Car);
...
heap_delete_n(&cars, 10, Car);
```

n Number of objects to create.

type Object type.

Return:

Pointer to the newly created array. It must be destroyed by `heap_delete_n` when it is no longer necessary.

heap_new_n0

Like `heap_new_n`, but initializing the array with 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	Number of objects to create.
type	Object type.

Return:

Pointer to the newly created array. It must be destroyed by `heap_delete_n` when it is no longer necessary.

heap_realloc_n

Reallocs an array of objects created dynamically with `heap_new_n` or `heap_new_n0`. Guarantees that the previous objects remain unchanged `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 Pointer to the array to relocate.

size Number of elements of the original array `mem`.

new_size New required size (in elements).

type Object type.

Return:

Pointer to the relocated array. It must be destroyed by `heap_delete_n` when it is no longer necessary.

heap_delete

Releases the object targeted by `obj`, previously reserved by `heap_new` or `heap_new0`.

```
void
heap_delete(type **obj,
            type);
```

obj Double pointer to the object to be released. It will be set to `NULL` after the release.

type Object type.

heap_delete_n

Free `n` objects targeted by `obj`, previously booked by `heap_new_n`, `heap_new_n0`.

```
void
heap_delete_n(type **obj,
              const uint32_t n,
              type);
```

obj Double pointer to the array to be released. It will be set to `NULL` after the release.

n Number of objects to be released, the same as in the reservation.

type Object type.

heap_auditor_add

Add an opaque object to the memory auditor.

```
void
heap_auditor_add(const char_t *name);
```

name Name of the object to add.

heap_auditor_delete

Releases an opaque object from the memory auditor.

```
void
heap_auditor_delete(const char_t *name);
```

name Name of the object to release.

buffer_create

Create a new buffer.

```
Buffer*
buffer_create(const uint32_t size);
```

size Buffer size in bytes.

Return:

The new buffer.

buffer_with_data

Create a new buffer and initialize it.

```
Buffer*
buffer_with_data(const byte_t *data,
                 const uint32_t size);
```

data Data to initialize the buffer.

size Buffer size in bytes.

Return:

The new buffer.

buffer_destroy

Destroy the buffer.

```
void
buffer_destroy(Buffer **buffer);
```

buffer The buffer. It will be set to `NULL` after the destruction.

buffer_size

Gets the size of the buffer.

```
uint32_t
buffer_size(const Buffer *buffer);
```

buffer Buffer.

Return:

The size of the buffer in bytes.

buffer_data

Gets a pointer to the contents of the buffer.

```
byte_t*
buffer_data(Buffer *buffer);
```

buffer Buffer.

Return:

Pointer to the contents of the buffer that can be used to read or write.

buffer_const

Get a *const* pointer to the contents of the buffer.

```
const byte_t*
buffer_const(const Buffer *buffer);
```

buffer Buffer.

Return:

Pointer to the content of the buffer that can be used for reading only.

tc

Returns the inner C string in format *"UTF-8UTF-8"* (page 158) contained in the String.

```
const char_t*
tc(const String *str);
```

str String object.

Return:

Pointer to the C-string.

tcc

Returns the inner C (non-const) string in *"UTF-8UTF-8"* (page 158) format contained in String.

```
char_t*
tcc(String *str);
```

str String object.

Return:

Pointer to the C-string.

str_c

Create a String from a *"UTF-8UTF-8"* (page 158)-encoded C string.

```
String*
str_c(const char_t *str);
```

str C UTF8 string ending in null character `'\0'`.

Return:

The String object.

str_cn

Create a String by copying the first `n` bytes of a C string.

```
String*
str_cn(const char_t *str,
       const uint32_t n);
```

str UTF8 C String.

n The number of bytes to copy.

Return:

The String object.

Remarks:

In *"UTF-8UTF-8"* (page 158) strings, the number of bytes does not correspond to the number of characters.

str_trim

Create a String from a C string by cutting the blanks, both at the beginning and at the end.

```
String*
str_trim(const char_t *str);
```

str C UTF8 string ending in null character `'\0'`.

Return:

The String object.

str_trim_n

Create a String from the first `n` bytes of a C string cutting the blanks, both at the beginning and at the end.

```
String*
str_trim_n(const char_t *str,
           const uint32_t n);
```

str UTF8 C string.

n The number of bytes to consider from the original string. The copy can contain 'n' or fewer bytes, depending on the number of blanks.

Return:

The String object.

str_copy

Create an exact copy of the String.

```
String*
str_copy(const String *str);
```

str The original String object.

Return:

The copy of String object.

Remarks:

Strings are a special type of mutable object. Copy involves creating a new object and not increasing a reference counter.

str_printf

Compose a String from several fields, using the the `printf` format.

```
String*
str_printf(const char_t *format,
           ...);
```

format String with the printf-like format with a variable number of parameters.

... Arguments or variables of the printf.

Return:

The String object.

Remarks:

The use of this function prevents **buffer overflow** vulnerabilities, associated with the classic C functions such as `strcpy`.

str_path

Like `str_printf`, but consider the string to be a *pathname* and therefore use the convenient separator according `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 Platform for which the *pathname* is created.

format String with the printf-like format with a variable number of parameters.

... Arguments or variables of the printf.

Return:

The String object.

str_cpath

Like `str_path`, but considering the platform where the program is running.

```
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 String with the printf-like format with a variable number of parameters.

... Arguments or variables of the printf.

Return:

The String object.

str_relpath

Calculate the relative path to `path1` to get to `path2`.

```
String*
str_relpath(const platform_t platform,
            const char_t *path1,
            const char_t *path2);
```

platform Platform for which the path is calculated (directory separator).

path1 The origin path.

path2 The destination path.

Return:

The string object that contains the relative path.

str_crelpath

Calculate the relative path to `path1` to get to `path2`.

```
String*
str_crelpath(const char_t *path1,
             const char_t *path2);
```

path1 The origin path.

path2 The destination path.

Return:

The string object that contains the relative path.

Remarks:

Same as `str_relpath`, but using the directory separator of the platform where the program is running.

str_repl

Create a String by replacing an undetermined number of sub-strings. The first parameter is the original string. The following pairs indicate the sub-string to be searched and the sub-string that should replace it. The last parameter must be 0.

```
String*
str_repl(const char_t *str,
         ...);
```

```
String *str = str_repl("const Product **pr;", "const", "", "*", "", " ", "", 0)
    ↪ ;
str = "Productpr;"
```

str Original C UTF8 string terminated in null character `'\0'`.

... Variable number of parameters, in pairs. The first element of the pair indicates the sub-string to look for in `str`. The second element replaces it.

Return:

The String object.

str_reserve

Create a String with n+1 bytes, but without assigning any content.

```
String*
str_reserve(const uint32_t n);
```

n Number of bytes. Reserve space for one more (the `'\n'`).

Return:

The String object. Its content will be indeterminate (garbage). It must be written later.

str_fill

Create a String by repeating `n` times the same character.

```
String*
str_fill(const uint32_t n,
         const char_t c);
```

n Number of characters.

c Pattern character.

Return:

The String object.

str_read

Create a String by reading its contents from a `Stream` (de-serialization). String must have been previously written by `str_write`.

```
String*
str_read(Stream *stream);
```

stream A read *stream*.

Return:

The String object.

Remarks:

It is a **binary** operation. String size is deserialized first.

str_write

Write a string in a *"Streams"* (page 193) (serialization).

```
void
str_write(Stream *stream,
          String *str);
```

stream A write *stream*.

str The String object.

Remarks:

It is a **binary** operation. The string size is serialized first. Use `str_writef` to write only the text.

str_writef

Write in a *"Streams"* (page 193) the C string contained in the string.

```
void
str_writef(Stream *stream,
           String *str);
```

stream A write *stream*.

str The String object.

Remarks:

Write only the *string* text, **without the null final character `'0'`**. It is equivalent to `stm_writef(stream, tc(str));` but more efficient, since you don't have to calculate the size of `str`.

str_copy_c

Copy the C string `src` in the buffer pointed by `dest`, including the null character `'\0'`.

```
void
str_copy_c(char_t *dest,
           const uint32_t size,
           const char_t *str);
```

dest Destination Buffer.

size Size in bytes of `dest`.

str UTF8 C string terminated in null character `'\0'`.

Remarks:

It is a safe operation. They will not be written in `dest` more of `size` bytes and a character will never be truncated. `dest` it will always end the null character `'\0'`.

str_copy_cn

Copy in `dest` a maximum of `n` bytes of the C UTF8 string pointed by `src`, including the null character `'\0'`.

```
void
str_copy_cn(char_t *dest,
            const uint32_t size,
            const char_t *str,
            const uint32_t n);
```

dest Destination Buffer.

size Size in bytes of `dest`.

str UTF8 C string.

n Maximum number of bytes to copy in `dest`.

Remarks:

It is a safe operation. They will not be written in `dest` more of `n` bytes and a character will never be truncated. `dest` it will always end the null character `'\0'`.

str_cat

Dynamically concatenates the content of `src` in `dest`.

```
void
str_cat(String **dest,
        const char_t *src);
```

**dest *String* object of origin and destination.

src UTF8 C string to concatenate.

Remarks:

This operation involves reallocating dynamic memory. To compose long texts it is more efficient to use `Stream`.

str_cat_c

Concatenate the content of `src` in `dest`. The null character in `dest` will be overwritten by the first character of `src`.

```
void
str_cat_c(char_t *dest,
          const uint32_t size,
          const char_t *src);
```

dest UTF8 C string origin and destination.

size Size in bytes of `dest`.

src UTF8 C string to concatenate.

Remarks:

It is a safe operation. They will not be written in `dest` more of `size` bytes and a character will never be truncated. `dest` it will always end the null character `'\0'`.

str_upd

Change the content of a *string* to another.

```
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	Destination *string* object. The original content will be deleted.
new_str	UTF8 C string that will replace the original.

str_destroy

Destroy a string object.

```
void
str_destroy(String **str);
```

str	The string object. Will be set to `NULL` after destruction.

str_destopt

Destroy a string object if its content is not `NULL` (optional destroyer).

```
void
str_destopt(String **str);
```

str	The string object. Will be set to `NULL` after destruction.

str_len

Returns the size in bytes of a string.

```
uint32_t
str_len(const String *str);
```

str	The String object.

Return:

The number of bytes, not including the null character '\0'.

Remarks:

In *"UTF-8UTF-8"* (page 158) strings the number of bytes is not the same as the characters. str_nchars.

str_len_c

Returns the size in bytes of a UTF8 C string.

```
uint32_t
str_len_c(const char_t *str);
```

str UTF8 C string terminated in null character '\0'.

Return:

The number of bytes, not including the null character '\0'.

Remarks:

In *"UTF-8UTF-8"* (page 158) strings the number of bytes is not the same as the characters. str_nchars.

str_nchars

Returns the number of characters of a string object.

```
uint32_t
str_nchars(const String *str);
```

str The String object.

Return:

The number of characters, not including the null character '\0'.

Remarks:

In *"UTF-8UTF-8"* (page 158) strings the number of bytes is not the same as the characters.

str_prefix

Locate the common begin of two strings.

```
uint32_t
str_prefix(const char_t *str1,
           const char_t *str2);
```

str1 First UTF8 C string terminated in null character `'\0'`.

str2 Second UTF8 C string terminated in null character `'\0'`.

Return:

The number of bytes that are identical at the beginning of both strings.

str_is_prefix

Check if one string is prefix of another.

```
bool_t
str_is_prefix(const char_t *str,
              const char_t *prefix);
```

str UTF8 C string terminated in null character `'\0'`.

prefix Prefix of `str` terminated in null character `'\0'`.

Return:

`TRUE` if `prefix` is prefix of `str`.

str_is_sufix

Check if one string is a suffix of another.

```
bool_t
str_is_sufix(const char_t *str,
             const char_t *sufix);
```

str Null-terminated UTF8 C string `'\0'`.

sufix Suffix of `str` terminated in null character `'\0'`.

Return:

`TRUE` si `sufix` is sufix of `str`.

str_scmp

Compare two strings alphabetically.

```
int
str_scmp(const String *str1,
         const String *str2);
```

str1 First string.

str2 Second string.

Return:

Comparison result.

str_cmp

Compare alphabetically a string with a UTF8 C string.

```
int
str_cmp(const String *str1,
        const char_t *str2);
```

str1 String object.

str2 C UTF8 string terminated in null character `'\0'`.

Return:

Comparison result.

str_cmp_c

Compare alphabetically two UTF8 C strings terminated in a null character `'\0'`.

```
int
str_cmp_c(const char_t *str1,
          const char_t *str2);
```

str1 First UTF8 C string.

str2 Second UTF8 C string.

Return:

Comparison result.

str_cmp_cn

Compare alphabetically the first `n` bytes of two UTF8 C strings terminated in a null character `'\0'`.

```
int
str_cmp_cn(const char_t *str1,
           const char_t *str2,
           const uint32_t n);
```

str1 First UTF8 C string.

str2 Second UTF8 C string.

n Maximum number of bytes to compare.

Return:

Comparison result.

Remarks:

It is a safe operation. If either of the two chains reaches the end before reaching `n` bytes, the comparison ends.

str_empty

Check if a string is empty (`str->data[0] == '\0'`).

```
bool_t
str_empty(const String *str);
```

str The String object.

Return:

`TRUE` if it is empty or is `NULL`.

str_empty_c

Check if a UTF8 C string is empty (`str[0] == '\0'`).

```
bool_t
str_empty_c(const char_t *str);
```

str UTF8 C string.

Return:

`TRUE` if it is empty or is `NULL`.

str_equ

Check if the content of a string is equal to a C string.

```
bool_t
str_equ(const String *str1,
        const char_t *str2);
```

str1 String object.

str2 UTF8 C string terminated in null character '\0'.

Return:

TRUE if they are equals.

str_equ_c

Check if two UTF8 C strings are equal.

```
bool_t
str_equ_c(const char_t *str1,
          const char_t *str2);
```

str1 First UTF8 C string terminated in null character '\0'.

str2 Second UTF8 C string terminated in null character '\0'.

Return:

TRUE if they are equals.

str_equ_cn

Check if the first bytes of two UTF8 C strings are equal.

```
bool_t
str_equ_cn(const char_t *str1,
           const char_t *str2,
           const uint32_t n);
```

str1 First UTF8 C string terminated in null character '\0'.

str2 Second UTF8 C string terminated in null character '\0'.

n First 'n' bytes to compare.

Return:

TRUE if they are equals.

Remarks:

If '\0' is reached in either of the two strings, `TRUE` will be returned.

str_equ_nocase

Check if two UTF8 C strings are equal, ignoring upper or lower case.

```
bool_t
str_equ_nocase(const char_t *str1,
               const char_t *str2);
```

str1 First UTF8 C string terminated in null character `'\0'`.

str2 Second UTF8 C string terminated in null character `'\0'`.

Return:

`TRUE` if they are equals.

Remarks:

Only US-ASCII characters are considered (0-127).

str_equ_end

Check the termination of a string.

```
bool_t
str_equ_end(const char_t *str,
            const char_t *end);
```

str UTF8 C string terminated in null character `'\0'`.

end UTF8 C string with termination.

Return:

`TRUE` if `str` ends in `end`.

str_upper

Change lowercase letters to uppercase.

```
void
str_upper(String *str);
```

str The String object.

Remarks:

Only US-ASCII characters (0-127) are considered. The original string will change, but not the memory requirements.

str_lower

Change uppercase letters to lowercase letters.

```
void
str_lower(String *str);
```

str The String object.

Remarks:

Only US-ASCII characters (0-127) are considered. The original string will change, but not the memory requirements.

str_upper_c

Convert a string to uppercase.

```
void
str_upper_c(char_t *dest,
            const uint32_t size,
            const char_t *str);
```

dest Destination buffer.

size Size in bytes of the destination buffer.

str String C UTF8 terminated in null character '\0'.

Remarks:

Only US-ASCII characters are considered (0-127).

str_lower_c

Convert a string to lowercase.

```
void
str_lower_c(char_t *dest,
            const uint32_t size,
            const char_t *str);
```

dest	Destination buffer.
size	Size in bytes of the destination buffer.
str	String C UTF8 terminated in null character `'\0'`.

Remarks:

Only US-ASCII characters are considered (0-127).

str_subs

Change all instances of one character to another.

```
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	The String object.
replace	Character to replace.
with	Replacement character.

Remarks:

Only US-ASCII characters (0-127) are considered. The original string will change, but not the memory requirements.

str_repl_c

Change all instances of one substring to another.

```
void
str_repl_c(String *str,
           const char_t *replace,
           const char_t *with);
```

str	The String object.
replace	Substring to replace.
with	Replacement substring.

Remarks:

The substrings `replace` and `with` they must be the same size, otherwise a *"Asserts"* (page 153) will be triggered. Use `str_repl` for the general case.

str_str

Search for a substring within a larger one.

```
const char_t*
str_str(const char_t *str,
        const char_t *substr);
```

str	UTF8 C strings terminated in null character `'\0'`.
substr	Substring to search terminated in null character `'\0'`.

Return:

Pointer to the first occurrence of `substr` in `str` or `NULL` if there is none.

str_split

Divide a string into two, using the first occurrence of a substring.

```
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	UTF8 C string terminated in null character `'\0'`.
substr	Substring to search.
left	Left substring. It will be equal to `str` if `substr` does not exist. The parameter can be `NULL` if not necessary.
right	Right substring. It will be equal to `""` if `substr` does not exist. The parameter can be `NULL` if not necessary.

Return:

`TRUE` if `substr` exists in `str`.

str_split_trim

Like `str_split` but removing all the blanks at the beginning and end of `left` and `right`.

```
bool_t
str_split_trim(const char_t *str,
               const char_t *substr,
               String **left,
               String **right);
```

str	UTF8 C string terminated in null character `'\0'`.
substr	Substring to search.
left	Left substring.
right	Right substring.

Return:

`TRUE` if `substr` exists in `str`.

str_splits

Splits a string into several, using a substring as a separator.

```
ArrPt(String)*
str_splits(const char_t *str,
           const char_t *substr,
           const bool_t trim);
```

str	UTF8 C string terminated in null character `'\0'`.
substr	Substring to search (separator).
trim	If `TRUE`, substrings will remove leading and trailing whitespace.

Return:

Array with the substrings found. It must be destroyed with `arrpt_destroy(&array, str_destroy, String)`.

Remarks:

Same as `str_split` or `str_split_trim`, but considering more than one substring.

str_split_pathname

Divide a *pathname* into `path` and `file` *"Filename and pathnameFilename and pathname"* (page 178).

```
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 Input pathname.

path Directory path. The parameter can be `NULL` if not necessary.

file File name or final directory. The parameter can be `NULL` if not necessary.

str_split_pathext

Like `str_split_pathname` but also extracting the file extension.

```
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 Input pathname.

path `Path` part.

file `File` part.

ext File extension.

str_filename

Returns the final part of a *pathname*. *"Filename and pathnameFilename and pathname"* (page 178).

```
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 Input pathname.

Return:

The last part of a directory path.

str_filext

Returns the file extension, from a *pathname*. *"Filename and pathnameFilename and pathname"* (page 178).

```
const char_t*
str_filext(const char_t *pathname);
```

```
const char_t *ext = str_fileext("C:\\Users\\john\\Desktop\\image.png");
ext = "png"
```

pathname Input pathname.

Return:

The file extension.

str_find

Search for a string in an array.

```
uint32_t
str_find(const ArrPt(String) *array,
         const char_t *str);
```

array Array.

str The string to find.

Return:

The position of the string or `UINT32_MAX` if it does not exist.

str_to_i8

Converts a text string into an integer.

```
int8_t
str_to_i8(const char_t *str,
          const uint32_t base,
          bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_i16

Converts a text string into an integer.

```
int16_t
str_to_i16(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_i32

Converts a text string into an integer.

```
int32_t
str_to_i32(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_i64

Converts a text string into an integer.

```
int64_t
str_to_i64(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_u8

Converts a text string into an integer.

```
uint8_t
str_to_u8(const char_t *str,
          const uint32_t base,
          bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_u16

Converts a text string into an integer.

```
uint16_t
str_to_u16(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_u32

Converts a text string into an integer.

```
uint32_t
str_to_u32(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_u64

Converts a text string into an integer.

```
uint64_t
str_to_u64(const char_t *str,
           const uint32_t base,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
base	Numeric base: 8 (octal), 10 (decimal), 16 (hexadecimal).
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0 with `error=TRUE`.

str_to_r32

Convert a string of text into a real.

```
real32_t
str_to_r32(const char_t *str,
           bool_t *error);
```

str	Text string, ending in null character `'\0'`.
error	Gets `TRUE` if there is an error in the conversion. Can be `NULL`.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0.0 with `error=TRUE`.

str_to_r64

Convert a string of text into a real.

```
real64_t
str_to_r64(const char_t *str,
           bool_t *error);
```

str Text string, ending in null character '\0'.

error Gets TRUE if there is an error in the conversion. Can be NULL.

Return:

The numerical value.

Remarks:

If the string is wrong or the value is out of range, return 0.0 with error=TRUE.

stm_from_block

Create a read stream from an existing memory block.

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

data Pointer to the memory block.

size Size in bytes of the memory block.

Return:

The stream.

Remarks:

The original block will not be modified (read only). When the end of the block is reached stm_state will return ekSTEND. *"Block streamBlock stream"* (page 195).

stm_memory

Create a read/write memory stream.

```
Stream*
stm_memory(const uint32_t size);
```

size Initial buffer size (in bytes). It will grow if necessary.

Return:

The stream.

Remarks:

It can be used as an internal pipeline for the information exchange between functions or threads. It behaves like a FIFO *(First In Fist Out)* buffer. For multi-threaded access you must be protected with a `Mutex`. *"Memory streamMemory stream"* (page 195).

stm_from_file

Create a stream to read from a file on disk.

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

pathname File *pathname*. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

The stream or `NULL` if the file opening fails.

Remarks:

"File streamFile stream" (page 194).

stm_to_file

Create a stream to write data to a file on disk.

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

pathname File *pathname*. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

The stream or `NULL` if file creation fails.

Remarks:

If the file already exists it will be overwritten. *"File streamFile stream"* (page 194).

stm_append_file

Create a stream to write data to the end of an existing file.

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

pathname File *pathname*. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

The stream or `NULL` if the file opening fails.

Remarks:

It will fail if the file does not exist (do not create it). *"File streamFile stream"* (page 194).

stm_socket

Create a stream from a socket.

```
Stream*
stm_socket(Socket *socket);
```

socket Client or server socket.

Return:

The stream.

Remarks:

It allows to use the streams functionality to read or write in a remote process. The socket must have been previously created with `bsocket_connect` (client) or `bsocket_accept` (server). See *"Socket streamSocket stream"* (page 194).

stm_close

Close the stream. All resources such as file descriptors or *sockets* will be released. Before to closing, the data will be written to the channel `stm_flush`.

```
void
stm_close(Stream **stm);
```

stm The stream. Will be set to NULL after closing.

stm_get_write_endian

Get the current byte order when writing to the stream.

```
endian_t
stm_get_write_endian(const Stream *stm);
```

stm The stream.

Return:

The *"Byte orderByte order"* (page 206).

stm_get_read_endian

Get the current byte order when reading from the stream.

```
endian_t
stm_get_read_endian(const Stream *stm);
```

stm The stream.

Return:

The *"Byte orderByte order"* (page 206).

stm_set_write_endian

Set the order of bytes when writing to the stream, from now on.

```
void
stm_set_write_endian(Stream *stm,
                     const endian_t endian);
```

stm The stream.

endian The *"Byte orderByte order"* (page 206).

Remarks:

Default is ekLITEND, except in sockets that will be ekBIGEND.

stm_set_read_endian

Set the order of bytes when reading from the stream, from now on.

```
void
stm_set_read_endian(Stream *stm,
                    const endian_t endian);
```

stm The stream.

endian The *"Byte orderByte order"* (page 206).

Remarks:

Default is ekLITEND, except in sockets that will be ekBIGEND.

stm_get_write_utf

Gets the UTF encoding with which the texts are being written in the stream.

```
unicode_t
stm_get_write_utf(const Stream *stm);
```

stm The stream.

Return:

"UTF encodingsUTF encodings" (page 157).

Remarks:

See *"Text streamText stream"* (page 198).

stm_get_read_utf

Get the UTF encoding with which the texts are being read in the stream.

```
unicode_t
stm_get_read_utf(const Stream *stm);
```

stm The stream.

Return:

"UTF encodingsUTF encodings" (page 157).

Remarks:

See *"Text streamText stream"* (page 198).

stm_set_write_utf

Set the UTF encoding when writing texts in the stream, from now on.

```
void
stm_set_write_utf(Stream *stm,
                  const unicode_t format);
```

stm The stream.

format *"UTF encodingsUTF encodings"* (page 157).

Remarks:

See *"Text streamText stream"* (page 198).

stm_set_read_utf

Set the UTF encoding when reading texts in the stream, from now on.

```
void
stm_set_read_utf(Stream *stm,
                 const unicode_t format);
```

stm The stream.

format *"UTF encodingsUTF encodings"* (page 157).

Remarks:

See *"Text streamText stream"* (page 198).

stm_is_memory

Gets if it is a memory-resident stream.

```
bool_t
stm_is_memory(const Stream *stm);
```

stm The stream.

Return:

`TRUE` if it was created by `stm_from_block` or `stm_memory`.

stm_bytes_written

Gets the total bytes written in the stream since its creation.

```
uint64_t
stm_bytes_written(const Stream *stm);
```

stm The stream.

Return:

The total number of bytes written.

stm_bytes_readed

Get the total bytes read from the stream since its creation.

```
uint64_t
stm_bytes_readed(const Stream *stm);
```

stm The stream.

Return:

The total number of bytes readed.

stm_col

Get the column in text streams.

```
uint32_t
stm_col(const Stream *stm);
```

stm The stream.

Return:

Column number.

Remarks:

When we read characters in text streams with `stm_read_char` or derivatives, the columns and rows are counted in a similar way as text editors do. This information can be useful when displaying warnings or error messages. In mixed streams (binary + text), the count stops when reading binary data and continues when reading the text is resumed. View *"Text streamText stream"* (page 198).

stm_row

Get row in text streams.

```
uint32_t
stm_row(const Stream *stm);
```

stm The stream.

Return:

Row number.

Remarks:

See `stm_col`.

stm_token_col

Gets the column of the last token read.

```
uint32_t
stm_token_col(const Stream *stm);
```

stm The stream.

Return:

Column number.

Remarks:

It only takes effect after calling `stm_read_token` or derivatives. See `stm_col` and *"TokensTokens"* (page 199).

stm_token_row

Gets the row of the last token read.

```
uint32_t
stm_token_row(const Stream *stm);
```

stm The stream.

Return:

Row number.

Remarks:

It only takes effect after calling `stm_read_token` or derivatives. See `stm_col` and *"TokensTokens"* (page 199).

stm_token_lexeme

Gets the lexeme of the last token read.

```
const char_t*
stm_token_lexeme(const Stream *stm);
```

stm The stream.

Return:

The lexeme. It is stored in a temporary buffer and will be lost when reading the next token. If you need it, make a copy with `str_c`.

Remarks:

It only takes effect after calling `stm_read_token` or derivatives. See `stm_col` and *"TokensTokens"* (page 199).

stm_token_escapes

Escape sequences option when reading tokens.

```
void
stm_token_escapes(const Stream *stm,
                  const bool_t active_escapes);
```

stm The stream.

active_escapes `TRUE` the escape sequences will be processed when reading `ekTSTRING` tokens. For example, the sequence `"\n"` will become the character `0x0A (10)`. `FALSE` will ignore escape sequences, reading strings literally. By default `FALSE`.

Remarks:

It will take effect on the next call to `stm_read_token`. See *"TokensTokens"* (page 199).

stm_token_spaces

Blanks option when reading tokens.

```
void
stm_token_spaces(const Stream *stm,
                 const bool_t active_spaces);
```

stm The stream.

active_spaces `TRUE` `ekTSPACE` tokens will be returned when finding sequences of whitespace. `FALSE` will ignore whitespace. By default `FALSE`.

Remarks:

It will take effect on the next call to `stm_read_token`. See *"TokensTokens"* (page 199).

stm_token_comments

Comments option when reading tokens.

```
void
stm_token_comments(const Stream *stm,
                   const bool_t active_comments);
```

stm The stream.

active_comments `TRUE` an `ekTMLCOM` token will be returned every time it encounters C comments `/ * Comment */` and `ekTSLCOM` for comments C++ `// Comment`. `FALSE` comments will be ignored. By default `FALSE`.

Remarks:

It will take effect on the next call to `stm_read_token`. See *"TokensTokens"* (page 199).

stm_state

Get the current state of the stream.

```
sstate_t
stm_state(const Stream *stm);
```

stm The stream.

Return:

The *"Stream stateStream state"* (page 207).

stm_file_err

Get additional information about the error, in disk streams.

```
ferror_t
stm_file_err(const Stream *stm);
```

stm The stream.

Return:

File error.

Remarks:

It is only relevant in *"File streamFile stream"* (page 194) with the state ekSTBROKEN.

stm_sock_err

Get additional information about the error, in network streams.

```
serror_t
stm_sock_err(const Stream *stm);
```

stm The stream.

Return:

Socket error.

Remarks:

It is only relevant in *"Socket streamSocket stream"* (page 194) with the state ekSTBROKEN
.

stm_corrupt

Set the stream status to ekSTCORRUPT.

```
void
stm_corrupt(Stream *stm);
```

stm The stream.

Remarks:

Sometimes, it is the application that detects that the data is corrupted since the data semantics wasn't expected.

stm_str

Create a string with the current content of the internal buffer. It is only valid for stream in memory. stm_memory.

```
String*
stm_str(const Stream *stm);
```

stm The stream.

Return:

The string with the buffer content.

stm_buffer

Gets a pointer to the current content of the internal buffer. Only valid for stream in memory. `stm_memory`.

```
const byte_t*
stm_buffer(const Stream *stm);
```

stm The stream.

Return:

Internal buffer pointer.

Remarks:

This pointer is read only. Writing here will have unexpected consequences.

stm_buffer_size

Get the current size of the internal buffer. Only valid for stream in memory. `stm_memory`.

```
uint32_t
stm_buffer_size(const Stream *stm);
```

stm The stream.

Return:

The size of the internal buffer (in bytes).

stm_write

Write bytes in the stream.

```
void
stm_write(Stream *stm,
          const byte_t *data,
          const uint32_t size);
```

stm	The stream.
data	Pointer to the data block to write.
size	Number of bytes to write.

Remarks:

The block is written as is, regardless of the *"Byte orderByte order"* (page 206) neither the *"UTF encodingsUTF encodings"* (page 157).

stm_write_char

Write a Unicode character in the stream.

```
void
stm_write_char(Stream *stm,
               const uint32_t codepoint);
```

stm	The stream.
codepoint	The *"Unicode"* (page 155) value of character.

Remarks:

The encoding can be changed with `stm_set_write_utf`.

stm_printf

Write text in the stream, using the `printf` format .

```
uint32_t
stm_printf(Stream *stm,
           const char_t *format,
           ...);
```

```
stm_printf(stream, Code: %-10s Price %5.2f\n", code, price);
```

stm	The stream.
format	String with the printf-like format with a variable number of parameters.
...	Arguments or variables of the printf.

Return:

The number of bytes written.

Remarks:

The final null character (`'\0'`) will not be written. The encoding can be changed with `stm_set_write_utf`.

stm_writef

Writes a UTF8 C string in the stream.

```
uint32_t
stm_writef(Stream *stm,
           const char_t *str);
```

stm The stream.

str C UTF8 string terminated in null character `'\0'`.

Return:

The number of bytes written.

Remarks:

The final null character (`'\0'`) will not be written. This function is faster than `stm_printf` when the string is constant and does not need formatting. For `String` objects use `str_writef`. The encoding can be changed with `stm_set_write_utf`.

stm_write_bool

Write a `bool_t` variable in the stream.

```
void
stm_write_bool(Stream *stm,
               const bool_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams.

stm_write_i8

Write a `int8_t` variable in the stream.

```
void
stm_write_i8(Stream *stm,
             const int8_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in “pure” text streams.

stm_write_i16

Write a `int16_t` variable in the stream.

```
void
stm_write_i16(Stream *stm,
              const int16_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in “pure” text streams. *“Byte orderByte order”* (page 206).

stm_write_i32

Write a `int32_t` variable in the stream.

```
void
stm_write_i32(Stream *stm,
              const int32_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in “pure” text streams. *“Byte orderByte order”* (page 206).

stm_write_i64

Write a `int64_t` variable in the stream.

```
void
stm_write_i64(Stream *stm,
              const int64_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_u8

Write a `uint8_t` variable in the stream.

```
void
stm_write_u8(Stream *stm,
             const uint8_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams.

stm_write_u16

Write a `uint16_t` variable in the stream.

```
void
stm_write_u16(Stream *stm,
              const uint16_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_u32

Write a `uint32_t` variable in the stream.

```
void
stm_write_u32(Stream *stm,
              const uint32_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_u64

Write a `uint64_t` variable in the stream.

```
void
stm_write_u64(Stream *stm,
              const uint64_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_r32

Write a `real32_t` variable in the stream.

```
void
stm_write_r32(Stream *stm,
              const real32_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_r64

Write a `real64_t` variable in the stream.

```
void
stm_write_r64(Stream *stm,
              const real64_t value);
```

stm The stream.

value Value to write.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_write_enum

Write a `enum` variable in the stream.

```
void
stm_write_enum(Stream *stm,
               const type value,
               type);
```

stm The stream.

value Value to write.

type The `enum` type.

Remarks:

It is a binary write. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read

Read bytes from the stream.

```
uint32_t
stm_read(Stream *stm,
         byte_t *data,
         const uint32_t size);
```

stm The stream.

data Pointer to the buffer where the read data will be written.

size The number of bytes to read (buffer size).

Return:

The number of bytes actually read.

stm_read_char

Read a text character from the stream.

```
uint32_t
stm_read_char(Stream *stm);
```

stm The stream.

Return:

The Unicode character code.

Remarks:

The encoding of the input text can be changed with `stm_set_read_utf`.

stm_read_chars

Read several characters from the stream.

```
const char_t*
stm_read_chars(Stream *stm,
               const uint32_t n);
```

stm The stream.

n The number of characters to read.

Return:

Pointer to the UTF8 C string read. It will end with the null character `'\0'`.

Remarks:

The returned pointer is temporary and will be overwritten in the next reading. If necessary, make a copy with `str_c`. The encoding of the input text can be changed with `stm_set_read_utf`.

stm_read_line

Read stream characters until an end of line is reached `'\n'`.

```
const char_t*
stm_read_line(Stream *stm);
```

stm The stream.

Return:

Pointer to the UTF8 C string, terminated with the null character `'\0'`. The characters `'\n'` or `'\r\n'` will not be included in the result. `NULL` will be returned when the end of the stream is reached.

Remarks:

The returned pointer is temporary and will be overwritten in the next reading. If necessary, make a copy with `str_c`. The encoding of the input text can be changed with `stm_set_read_utf`.

stm_read_trim

Read the following sequence of characters removing the blank spaces.

```
const char_t*
stm_read_trim(Stream *stm);
```

stm The stream.

Return:

Pointer to the C UTF8 string read. It will end with the null character `'\0'`.

Remarks:

Useful for reading strings from text streams. It will ignore all leading blanks and read characters until the first blank is found (`' '`, `'\\t'`, `'\\n'`, `'\\v'`, `'\\f'`, `'\\r'`). If you need more control over *tokens* use `stm_read_token`. The pointer returned is temporary and will be overwritten on the next read. If necessary, make a copy with `str_c`. The input text encoding can be adjusted with `stm_set_read_utf`. It will update the row and column counter. See `stm_col`.

stm_read_token

Get the following token in *"Text streamText stream"* (page 198).

```
token_t
stm_read_token(Stream *stm);
```

stm The stream.

Return:

The type of token obtained.

Remarks:

To get the text string associated with the token, use stm_token_lexeme. See *"TokensTokens"* (page 199).

stm_read_i8_tok

Read the following token with stm_read_token and, if it is an integer, convert it to int8_t.

```
int8_t
stm_read_i8_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

In case a token of type ekTINTEGER cannot be read (with or without ekTMINUS) or the numeric value is out of range, 0 will be returned and the stream will be marked as corrupt with stm_corrupt.

stm_read_i16_tok

Read the next token and convert it to int16_t.

```
int16_t
stm_read_i16_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See stm_read_i8_tok.

stm_read_i32_tok

Read the next token and convert it to int32_t.

```
int32_t
stm_read_i32_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See `stm_read_i8_tok`.

stm_read_i64_tok

Read the next token and convert it to `int64_t`.

```
int64_t
stm_read_i64_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See `stm_read_i8_tok`.

stm_read_u8_tok

Read the following token with `stm_read_token` and, if it is an integer, convert it to `uint8_t`.

```
uint8_t
stm_read_u8_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

In case a token of type `ekTINTEGER` cannot be read or the numeric value is out of range, 0 will be returned and the stream will be marked as corrupt with `stm_corrupt`.

stm_read_u16_tok

Read the next token and convert it to uint16_t.

```
uint16_t
stm_read_u16_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See stm_read_u8_tok.

stm_read_u32_tok

Read the next token and convert it to uint32_t.

```
uint32_t
stm_read_u32_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See stm_read_u8_tok.

stm_read_u64_tok

Read the next token and convert it to uint64_t.

```
uint64_t
stm_read_u64_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See stm_read_u8_tok.

stm_read_r32_tok

Read the following token with `stm_read_token` and, if it is a real number, convert it to `real32_t`.

```
real32_t
stm_read_r32_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

In case a token of type `ekTINTEGER` or `ekTREAL` cannot be read (with or without `ekTMINUS`), 0 will be returned and the stream will be marked as corrupt with `stm_corrupt`.

stm_read_r64_tok

Read the next token and convert it to `real64_t`.

```
real64_t
stm_read_r64_tok(Stream *stm);
```

stm The stream.

Return:

The numeric value of the token.

Remarks:

See `stm_read_r32_tok`.

stm_read_bool

Read a `bool_t` value from the stream.

```
bool_t
stm_read_bool(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in “pure” text streams.

stm_read_i8

Read a `int8_t` value from the stream.

```
int8_t
stm_read_i8(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in “pure” text streams.

stm_read_i16

Read a `int16_t` value from the stream.

```
int16_t
stm_read_i16(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in “pure” text streams. *“Byte orderByte order”* (page 206).

stm_read_i32

Read a `int32_t` value from the stream.

```
int32_t
stm_read_i32(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_i64

Read a `int64_t` value from the stream.

```
int64_t
stm_read_i64(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_u8

Read a `uint8_t` value from the stream.

```
uint8_t
stm_read_u8(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams.

stm_read_u16

Read a `uint16_t` value from the stream.

```
uint16_t
stm_read_u16(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_u32

Read a `uint32_t` value from the stream.

```
uint32_t
stm_read_u32(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_u64

Read a `uint64_t` value from the stream.

```
uint64_t
stm_read_u64(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_r32

Read a `real32_t` value from the stream.

```
real32_t
stm_read_r32(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_r64

Read a `real64_t` value from the stream.

```
real64_t
stm_read_r64(Stream *stm);
```

stm The stream.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_read_enum

Read a `enum` value from the stream.

```
type
stm_read_enum(Stream *stm,
              type);
```

stm The stream.

type The `enum` type.

Return:

Value read.

Remarks:

It is a binary reading. Do not use in "pure" text streams. *"Byte orderByte order"* (page 206).

stm_skip

Skip and ignore the next bytes of the stream.

```
void
stm_skip(Stream *stm,
         const uint32_t size);
```

stm The stream.

size The number of bytes to skip.

stm_skip_bom

Skip the possible *Byte Order Mark* sequence `"ï»?`"` found at the beginning of some text streams.

```
void
stm_skip_bom(Stream *stm);
```

stm The stream.

Remarks:

This function will have no effect if there is no such sequence at the beginning of the stream. The BOM is common in streams coming from some web servers.

stm_skip_token

Skip the next token in the stream. If the token does not correspond to the one indicated, the stream will be marked as corrupt.

```
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 The stream.

token Expected token.

stm_flush

Write in the channel the existing information in the cache.

```
void
stm_flush(Stream *stm);
```

stm The stream.

Remarks:

To improve performance, write operations on disk streams or standard I/O are stored in an internal cache. This function forces writing on the channel and cleans the buffer. It will be useful with *full-duplex* protocols where the receiver awaits reply to continue.

stm_pipe

Connect two streams, reading data from one and writing it to another.

```
void
stm_pipe(Stream *from,
         Stream *to,
         const uint32_t n);
```

from The input stream (to read).

to The output stream (to write).

n The number of bytes to be transferred.

Remarks:

The transfer will be made on raw data, regardless of *"Byte orderByte order"* (page 206) or *"UTF encodingsUTF encodings"* (page 157). If you are clear that this does not affect, it is much faster than using atomic read/write operations.

stm_lines

Iterate over all lines in a *"Text streamText stream"* (page 198). You should use `stm_next` to close the loop.

```
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 Name of the variable that will temporarily host the line. Use an internal stream cache, so you should make a copy with `str_c` if you need to keep it.

stm The stream.

stm_next

Close a loop open by `stm_lines`.

```
void
stm_next(const char_t *line,
         Stream *stm);
```

line Name of the line variable.

stm The stream.

arrst_create

Create an empty array.

```
ArrSt(type)*
arrst_create(type);
```

type Object type.

Return:

The new array.

arrst_copy

Create a copy of an array.

```
ArrSt(type)*
arrst_copy(const ArrSt(type) *array,
           FPtr_scopy func_copy,
           type);
```

array	The original array.
func_copy	Function that must copy the fields of each object.
type	Object type.

Return:

The copy of the original array.

Remarks:

The copy function must allocate memory to the fields that require it, but NOT to the object itself. If we pass `NULL`, a byte-by-byte copy of the original object will be made, which may pose an integrity risk if the array elements contain `String` or other objects that need dynamic memory.

arrst_read

Create an array by reading its contents from a *"Streams"* (page 193) (de-serialization).

```
ArrSt(type)*
arrst_read(Stream *stream,
           FPtr_read_init func_read,
           type);
```

stream	A read *stream*.
func_read	Function to initialize an object from the data obtained from a stream. This function should not reserve memory for the object itself (the container already does). *"SerializationSerialization"* (page 213).
type	Object type.

Return:

The array readed.

arrst_destroy

Destroy an array and all its elements.

```
void
arrst_destroy(ArrSt(type) **array,
              FPtr_remove func_remove,
              type);
```

array The array. It will be set to NULL after destruction.

func_remove Function that must free the memory associated with the object's fields, but not the object itself *"DestructorsDestructors"* (page 214). If NULL only the array will be destroyed and not the internal content of the elements.

type Object type.

arrst_destopt

Destroy an array and all its elements, as long as the array object is not NULL.

```
void
arrst_destopt(ArrSt(type) **array,
              FPtr_remove func_remove,
              type);
```

array The array.

func_remove See arrst_destroy.

type Object type.

arrst_clear

Delete the contents of the array, without destroying the container that will be left with zero elements.

```
void
arrst_clear(ArrSt(type) *array,
            FPtr_remove func_remove,
            type);
```

array The array.

func_remove Remove function. See arrst_destroy.

type Object type.

arrst_write

Write an array in a *"Streams"* (page 193) (serialization).

```
void
arrst_write(Stream *stream,
            const ArrSt(type) *array,
            FPtr_write func_write,
            type);
```

stream A write *stream*.

array The array.

func_write Function that writes the content of an element in a stream *"SerializationSerialization"* (page 213).

type Object type.

arrst_size

Get the number of elements in an array.

```
uint32_t
arrst_size(const ArrSt(type) *array,
           type);
```

array The array.

type Object type.

Return:

Number of elements.

arrst_get

Get a pointer to the item in `pos` position.

```
type*
arrst_get(ArrSt(type) *array,
          const uint32_t pos,
          type);
```

array The array.

pos Item position or index.

type Object type.

Return:

Item Pointer.

arrst_get_const

Get a **const** pointer to the item in `pos` position.

```
const type*
arrst_get_const(const ArrSt(type) *array,
                const uint32_t pos,
                type);
```

array The array.

pos Item position or index.

type Object type.

Return:

Item Pointer.

arrst_first

Gets a pointer to the first element of the array.

```
type*
arrst_first(ArrSt(type) *array,
            type);
```

array The array.

type Object type.

Return:

Item pointer.

arrst_first_const

Gets a **const** pointer to the first element of the array.

```
const type*
arrst_first_const(const ArrSt(type) *array,
                  type);
```

array The array.

type Object type.

Return:

Item pointer.

arrst_last

Get a pointer to the last element of the array.

```
type*
arrst_last(ArrSt(type) *array,
           type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrst_last_const

Get a **const** pointer to the last element of the array.

```
const type*
arrst_last_const(const ArrSt(type) *array,
                 type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrst_all

Get a pointer to the internal memory of the array, which gives direct access to all the elements.

```
type*
arrst_all(ArrSt(type) *array,
          type);
```

array The array.

type Object type.

Return:

Base pointer. Increasing it one by one we will iterate over the elements.

Remarks:

Use `arrst_foreach` to iterate over all elements in a more secure and elegant way.

arrst_all_const

Get a **const** pointer to the internal memory of the array, which gives direct access to all the elements.

```
const type*
arrst_all_const(const ArrSt(type) *array,
                type);
```

array The array.

type Object type.

Return:

Base pointer. Increasing it one by one we will iterate over the elements.

Remarks:

Use `arrst_foreach_const` to iterate over all elements in a more secure and elegant way.

arrst_grow

Add `n` elements, not initialized, at the end of the array.

```
void
arrst_grow(ArrSt(type) *array,
           const uint32_t n,
           type);
```

array The array.

n Number of items to add.

type Object type.

arrst_new

Reserve space for an element at the end of the 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 The array.

type Object type.

Return:

Pointer to added element.

Remarks:

It is slightly faster than `arrst_append`, especially in large structures, since it avoids copying the contents of the object. Initial memory content is indeterminate.

arrst_new0

Reserve space for an element at the end of the array and initialize it to 0.

```
type*
arrst_new0(ArrSt(type) *array,
           type);
```

array The array.

type Object type.

Return:

Pointer to added element.

Remarks:

Same as `arrst_new` but initializing all memory to 0.

arrst_new_n

Reserve space for multiple elements at the end of the array.

```
type*
arrst_new_n(ArrSt(type) *array,
            const uint32_t n,
```

```
            type);
```

array The array.

n Number of elements to add.

type Object type.

Return:

Pointer to the first element added.

Remarks:

Same as `arrst_new` but reserving multiple elements in the same call. Initial memory content is indeterminate.

arrst_new_n0

Reserve space for several elements at the end of the array and initialize them to 0.

```
type*
arrst_new_n0(ArrSt(type) *array,
             const uint32_t n,
             type);
```

array The array.

n Number of elements to add.

type Object type.

Return:

Pointer to the first element added.

Remarks:

Same as `arrst_new_n` but initializing all memory to 0.

arrst_prepend_n

Reserve space for several elements at the beginning of the array. The rest of the elements will be shifted to the right.

```
type*
arrst_prepend_n(ArrSt(type) *array,
                const uint32_t n,
                type);
```

array The array.

n Number of elements to insert.

type Object type.

Return:

Pointer to the first inserted element.

Remarks:

Initial memory content is indeterminate.

arrst_insert_n

Reserve space for several elements in an arbitrary position of the array.

```
type*
arrst_insert_n(ArrSt(type) *array,
               const uint32_t pos,
               const uint32_t n,
               type);
```

array The array.

pos Position where it will be inserted. The current element in `pos` and following will be shifted to the right.

n Number of elements to insert.

type Object type.

Return:

Pointer to the first inserted element.

Remarks:

Initial memory content is indeterminate.

arrst_append

Append an element to the end of the array.

```
void
arrst_append(ArrSt(type) *array,
             type value,
             type);
```

array The array.

value Item to add.

type Object type.

arrst_prepend

Insert an element at the beginning of the array. The rest of the elements will be shifted to the right.

```
void
arrst_prepend(ArrSt(type) *array,
              type value,
              type);
```

array The array.

value Item to insert.

type Object type.

arrst_insert

Insert an element in an arbitrary array position.

```
void
arrst_insert(ArrSt(type) *array,
             const uint32_t pos,
             type value,
             type);
```

array The array.

pos Position where it will be inserted. The current item in `pos` and following will be shifted to the right.

value Item to insert.

type Object type.

arrst_join

Join two vectors. Add all the elements of `src` to the end of `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	The destination array.
src	The array whose elements will be added to `dest`.
func_copy	Object copy function.
type	Object type.

Remarks:

The copy function must create dynamic memory for the fields that require it, but NOT for the object itself. See `arrst_copy`. If it is `NULL`, a byte-by-byte copy of the element will be made.

arrst_delete

Remove an element from the array.

```
void
arrst_delete(ArrSt(type) *array,
             const uint32_t pos,
             FPtr_remove func_remove,
             type);
```

array	The array.
pos	Position of the item to be deleted. The current item in `pos+1` and following will be shifted to the left.
func_remove	'Remove' function. See `arrst_destroy`.
type	Object type.

arrst_pop

Remove the last element from the array.

```
void
arrst_pop(ArrSt(type) *array,
          FPtr_remove func_remove,
          type);
```

array	The array.
func_remove	'Remove' function. See `arrst_destroy`.
type	Object type.

arrst_sort

Sort array elements using Quicksort.

```
void
arrst_sort(ArrSt(type) *array,
           FPtr_compare func_compare,
           type);
```

array	The array.
func_compare	Function to compare two elements. *"Sort and searchSort and search"* (page 216).
type	Object type.

arrst_sort_ex

Sort array elements using Quicksort and additional data.

```
void
arrst_sort_ex(ArrSt(type) *array,
              FPtr_compare_ex func_compare,
              type,
              dtype);
```

array	The array.
func_compare	Function to compare two elements using an additional data.
type	Object type.
dtype	Type of data in the comparison function.

arrst_search

Search for an element in the array linearly `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	The array.
func_compare	Comparison function. The first parameter is the element, the second the search key. *"Sort and searchSort and search"* (page 216).
key	Search key. Pointer to a data type that may be different from the type of array element.
pos	Position of the element in the array (if it exists), or `UINT32_MAX` if it does not exist. Can be `NULL`.
type	Object type.
ktype	Key type.

Return:

Pointer to the first element that matches the search criteria or `NULL` if none exists.

arrst_search_const

Const version of `arrst_search`.

```
const type*
arrst_search_const(const ArrSt(type) *array,
                   FPtr_compare func_compare,
                   const ktype *key,
                   uint32_t *pos,
                   type,
                   ktype);
```

array	The array.
func_compare	Comparison function.
key	Search key.
pos	Position of the element in the array.
type	Object type.
ktype	Key type.

Return:

Pointer to element.

arrst_bsearch

Search for an element in the array logarithmically `O(logn)`.

```
type*
arrst_bsearch(ArrSt(type) *array,
              FPtr_compare func_compare,
              const ktype *key,
              uint32_t *pos,
              type,
              ktype);
```

array	The array.
func_compare	Comparison function. The first parameter is the element, the second the search key. *"Sort and searchSort and search"* (page 216).
key	Search key. Pointer to a data type that may be different from the type of array element.
pos	Position of the element in the array (if it exists), or `UINT32_MAX` if it does not exist. Can be `NULL`.
type	Object type.
ktype	Key type.

Return:

Pointer to the first element that matches the search criteria or `NULL` if none exists.

Remarks:

The array must be sorted according to the same criteria as the search. If not, the result is unpredictable.

arrst_bsearch_const

`Const` version of `arrst_bsearch`.

```
const type*
arrst_bsearch_const(const ArrSt(type) *array,
                    FPtr_compare func_compare,
                    const ktype *key,
                    uint32_t *pos,
                    type,
                    ktype);
```

array	The array.
func_compare	Comparison function.
key	Seach key.
pos	Element position in array.
type	Object type.
ktype	Key type.

Return:

Pointer to element.

arrst_foreach

Iterate on all array elements. Uses `arrst_end` to close the loop.

```
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	Name of the 'element' variable within the loop. Adding the suffix `'_i'` we get the index.
array	The array.
type	Object type.

arrst_foreach_const

Const version of `arrst_foreach`.

```
void
arrst_foreach_const(const type *elem,
                    const ArrSt(type) *array,
                    type);
```

elem Element.

array The array.

type Object type.

arrst_forback

Iterate on all array elements backward, from the last to the first. Uses `arrst_end` to close the loop.

```
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 Name of the 'element' variable within the loop. Adding the suffix `'_i'` we get the index.

array The array.

type Object type.

arrst_forback_const

Const version of `arrst_forback`.

```
void
arrst_forback_const(const type *elem,
                    const ArrSt(type) *array,
                    type);
```

elem Element.

array The array.

type Object type.

arrst_end

Close the loop opened by `arrst_foreach`, `arrst_foreach_const`, `arrst_forback` or `arrst_forback_const`.

```
void
arrst_end(void);
```

arrpt_create

Create an empty array of pointers.

```
ArrPt(type)*
arrpt_create(type);
```

type Object type.

Return:

The new array.

arrpt_copy

Create a copy of an array of pointers.

```
ArrPt(type)*
arrpt_copy(const ArrPt(type) *array,
           FPtr_copy func_copy,
           type);
```

array The original array.

func_copy Object copy function.

type Object type.

Return:

The copy of the original array.

Remarks:

The copy function must create a dynamic object and allocate memory for internal fields that require it. If we pass `NULL`, a copy of the original pointers will be made, which can pose an integrity risk since the same object can be destroyed twice if we are not careful. See *"Copy objectsCopy objects"* (page 213).

arrpt_read

Create an array by reading its contents from a *"Streams"* (page 193) (de-serialization).

```
ArrPt(type)*
arrpt_read(Stream *stream,
           FPtr_read func_read,
           type);
```

stream — A read *stream*.

func_read — Constructor to create an object from the data obtained from a stream. *"SerializationSerialization"* (page 213).

type — Object type.

Return:

The array readed.

arrpt_destroy

Destroy an array and all its elements.

```
void
arrpt_destroy(ArrPt(type) **array,
              FPtr_destroy func_destroy,
              type);
```

array — The array. It will be set to `NULL` after destruction.

func_destroy — Function to destroy an element *"DestructorsDestructors"* (page 214). If `NULL` only the array will be destroyed, but not its elements.

type — Object type.

arrpt_destopt

Destroy an array and all its elements, as long as the array object is not `NULL`.

```
void
arrpt_destopt(ArrSt(type) **array,
              FPtr_destroy func_destroy,
              type);
```

array — The array.

func_destroy — See `arrpt_destroy`.

type — Object type.

arrpt_clear

Delete the contents of the array, without destroying the container that will be left with zero elements.

```
void
arrpt_clear(ArrPt(type) *array,
            FPtr_destroy func_destroy,
            type);
```

array	The array.
func_destroy	Destructor function. See `arrpt_destroy`.
type	Object type.

arrpt_write

Write an array in a *"Streams"* (page 193) (serialization).

```
void
arrpt_write(Stream *stream,
            const ArrPt(type) *array,
            FPtr_write func_write,
            type);
```

stream	A write *stream*.
array	The array.
func_write	Function that writes the content of an element in a stream *"SerializationSerialization"* (page 213).
type	Object type.

arrpt_size

Get the number of elements in an array.

```
uint32_t
arrpt_size(const ArrPt(type) *array,
           type);
```

array	The array.
type	Object type.

Return:

Number of elements.

arrpt_get

Get a pointer to the item in `pos` position.

```
type*
arrpt_get(ArrPt(type) *array,
          const uint32_t pos,
          type);
```

array The array.

pos Item position or index.

type Object type.

Return:

Item Pointer.

arrpt_get_const

Get a **const** pointer to the item in `pos` position.

```
const type*
arrpt_get_const(const ArrPt(type) *array,
                const uint32_t pos,
                type);
```

array The array.

pos Item position or index.

type Object type.

Return:

Item Pointer.

arrpt_first

Get a pointer to the first element of the array.

```
type*
arrpt_first(ArrPt(type) *array,
            type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrpt_first_const

Get a **const** pointer to the first element of the array.

```
const type*
arrpt_first_const(const ArrPt(type) *array,
                  type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrpt_last

Get a pointer to the last element of the array.

```
type*
arrpt_last(ArrPt(type) *array,
           type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrpt_last_const

Get a **const** pointer to the last element of the array.

```
const type*
arrpt_last_const(const ArrPt(type) *array,
                 type);
```

array The array.

type Object type.

Return:

Item Pointer.

arrpt_all

Get a pointer to the internal memory of the array, which gives access to all the elements.

```
type**
arrpt_all(ArrPt(type) *array,
          type);
```

array The array.

type Object type.

Return:

Base pointer. Increasing it one by one we will iterate over the elements.

Remarks:

Use `arrpt_foreach` to iterate over all elements in a more secure and elegant way.

arrpt_all_const

Get a **const** pointer to the internal memory of the array, which gives access to all the elements.

```
const type**
arrpt_all_const(const ArrPt(type) *array,
                type);
```

array The array.

type Object type.

Return:

Base pointer. Increasing it one by one we will iterate over the elements.

Remarks:

Use `arrpt_foreach_const` to iterate over all elements in a more secure and elegant way.

arrpt_grow

Append `n` elements, not initialized, at the end of the array.

```
type**
arrpt_grow(ArrPt(type) *array,
           const uint32_t n,
           type);
```

array The array.

n Number of items to add.

type Object type.

Return:

Pointer to the first item added.

arrpt_append

Adds a pointer to the end of the array.

```
void
arrpt_append(ArrPt(type) *array,
             type *value,
             type);
```

array The array.

value Pointer to the item to append.

type Object type.

arrpt_prepend

Insert a pointer at the beginning of the array. The rest of the elements will be shifted to the right.

```
void
arrpt_prepend(ArrPt(type) *array,
              type *value,
              type);
```

array The array.

value Pointer to the element to insert.

type Object type.

arrpt_insert

Insert a pointer in an arbitrary array position.

```
void
arrpt_insert(ArrPt(type) *array,
             const uint32_t pos,
             type *value,
             type);
```

array	The array.
pos	Position where it will be inserted. The current item in `pos` and following will be shifted to the right.
value	Pointer to the element to insert.
type	Object type.

arrpt_join

Join two vectors. Add all the elements of `src` to the end of `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	The destination array.
src	The array whose elements will be added to `dest`.
func_copy	Object copy function.
type	Object type.

Remarks:

The copy function must create dynamic memory for both the object and the fields that require it. If it is `NULL` it will only add a copy of the original pointer to `dest`.

arrpt_delete

Remove a pointer from the array.

```
void
arrpt_delete(ArrPt(type) *array,
             const uint32_t pos,
             FPtr_destroy func_destroy,
             type);
```

array	The array.
pos	Position of the item to be deleted. The current item in `pos+1` and following will be shifted to the left.
func_destroy	Element destructor. See `arrpt_destroy`.
type	Object type.

arrpt_pop

Remove the last pointer from the array.

```
void
arrpt_pop(ArrPt(type) *array,
          FPtr_destroy func_destroy,
          type);
```

array	The array.
func_destroy	Element destructor. See `arrpt_destroy`.
type	Object type.

arrpt_sort

Sort the array elements using Quicksort.

```
void
arrpt_sort(ArrPt(type) *array,
           FPtr_compare func_compare,
           type);
```

array	The array.
func_compare	Function to compare two elements. *"Sort and searchSort and search"* (page 216).
type	Object type.

arrpt_sort_ex

Sort array elements using Quicksort and additional data.

```
void
arrpt_sort_ex(ArrPt(type) *array,
              FPtr_compare_ex func_compare,
              type,
              dtype);
```

array	The array.
func_compare	Function to compare two elements using an additional data.
type	Object type.
dtype	Type of data in the comparison function.

arrpt_find

Search for a specific pointer in the array.

```
uint32_t
arrpt_find(const ArrPt(type) *array,
           type *elem,
           type);
```

array	The array.
elem	Pointer to find.
type	Object type.

Return:

The position of the pointer if it exists, or `UINT32_MAX` if not.

arrpt_search

Search for an element in the array linearly `O(n)`.

```
type*
arrpt_search(ArrPt(type) *array,
             FPtr_compare func_compare,
             ktype key,
             uint32_t *pos,
             type,
             ktype);
```

array	The array.
func_compare	Comparison function. The first parameter is the element, the second the search key. *"Sort and searchSort and search"* (page 216).
key	Search key. Pointer to a data type that may be different from the type of array element.
pos	Position of the element in the array (if it exists), or `UINT32_MAX` if it does not exist. Can be `NULL`.
type	Object type.
ktype	Key type.

Return:

Pointer to the first element that matches the search criteria or `NULL` if none exists.

arrpt_search_const

Const version of `arrpt_search`.

```
const type*
arrpt_search_const(const ArrPt(type) *array,
                   FPtr_compare func_compare,
                   const ktype *key,
                   uint32_t *pos,
                   type,
                   ktype);
```

array	The array.
func_compare	Comparison function.
key	Search key.
pos	Position of the element in the array.
type	Object type.
ktype	Key type.

Return:

Element.

arrpt_bsearch

Search for an element in the array logarithmically `O(logn)`.

```
type*
arrpt_bsearch(ArrPt(type) *array,
              FPtr_compare func_compare,
              ktype key,
              uint32_t *pos,
              type,
              ktype);
```

array	The array.
func_compare	Comparison function. The first parameter is the element, the second the search key. *"Sort and searchSort and search"* (page 216).
key	Key to search. Pointer to a data type that can be different from the element type of the array.
pos	Position of the element in the array (if it exists), or `UINT32_MAX` if it does not exist. Can be `NULL`.
type	Object type.
ktype	Key type.

Return:

Pointer to the first element that matches the search criteria or `NULL` if none exists.

Remarks:

The array must be sorted according to the same criteria as the search. If not, the result is unpredictable.

arrpt_bsearch_const

Const version of `arrpt_bsearch`.

```
const type*
arrpt_bsearch_const(const ArrPt(type) *array,
                    FPtr_compare func_compare,
                    const ktype *key,
                    uint32_t *pos,
                    type,
                    ktype);
```

array	The array.
func_compare	Comparison function.
key	Search key.
pos	Position of the element in the array.
type	Object type.
ktype	Key type.

Return:

Element.

arrpt_foreach

Iterate on all array elements. Uses `arrpt_end` to close the loop.

```
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	Name of the 'element' variable within the loop. Adding the suffix `'_i'` we get the index.
array	The array.
type	Object type.

arrpt_foreach_const

Const version of `arrpt_foreach`.

```
void
arrpt_foreach_const(const type *elem,
                    const ArrPt(type) *array,
                    type);
```

elem	Element.
array	The array.
type	Object type.

arrpt_forback

Iterate on all array elements backward, from the last to the first. Uses arrpt_end to close the loop.

```
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 Name of the 'element' variable within the loop. Adding the suffix '_i' we get the index.

array The array.

type Object type.

arrpt_forback_const

Const version of arrpt_forback.

```
void
arrpt_forback_const(const type *elem,
                    const ArrPt(type) *array,
                    type);
```

elem Element.

array The array.

type Object type.

arrpt_end

Close the loop opened by arrpt_foreach, arrpt_foreach_const, arrpt_forback or arrpt_forback_const.

```
void
arrpt_end(void);
```

setst_create

Create an empty set of registers.

```
SetSt(type)*
setst_create(FPtr_compare func_compare,
             type);
```

func_compare Function to compare two elements. *"Sort and searchSort and search"* (page 216).

type Object type.

Return:

The new set.

setst_destroy

Destroy a set and all its elements.

```
void
setst_destroy(SetSt(type) **set,
              FPtr_remove func_remove,
              type);
```

set The set. Will be set to `NULL` after destruction.

func_remove Function that must free the memory associated with the object's fields, but not the object itself *"DestructorsDestructors"* (page 214). If it is `NULL` only the set will be released and not the internal content of the elements.

type Object type.

setst_size

Get the number of set elements.

```
uint32_t
setst_size(const SetSt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Number of items.

setst_get

Search for an item in O(logn). It is equivalent to arrst_bsearch. If exists, the internal structure iterator will be fixed in it.

```
type*
setst_get(SetSt(type) *set,
          const type *key,
          type);
```

```
Product key;
Product *pr;
key.id = 453;
pr = setst_get(setst, &key, Product);
```

set The set.

key Search key. It is a pointer to an object where only the relevant search fields must be initialized.

type Object type.

Return:

Pointer to the item if it exists, or NULL if not.

Remarks:

"IteratorsIterators" (page 220).

setst_get_const

Const version of setst_get.

```
const type*
setst_get_const(const SetSt(type) *set,
                const type *key,
                type);
```

set The set.

key Search key.

type Object type.

Return:

Element.

setst_insert

Insert a new item in the 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 The set.

key Key to insert. It is a pointer to an object where only the relevant search fields must be initialized.

type Object type.

Return:

Pointer to the inserted element, which should be used to initialize the object. If an item with the same key already exists, it returns `NULL`.

Remarks:

Inserting or deleting elements invalidates the internal set iterator *"IteratorsIterators" (page 220)*. You must re-initialize it with `setst_first`.

setst_delete

Remove an item from the 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	The set.
key	Key to delete. It is a pointer to an object where only the relevant search fields must be initialized.
func_remove	Remove function. See `setst_destroy`.
type	Object type.

Return:

`TRUE` if the item has been deleted, or `FALSE` if there is no item with that key.

Remarks:

Inserting or deleting elements invalidates the internal set iterator *"IteratorsIterators"* (page 220). You must re-initialize it with `setst_first`.

setst_first

Get the first set element and initialize the internal iterator.

```
type*
setst_first(SetSt(type) *set,
            type);
```

set	The set.
type	Object type.

Return:

Pointer to the first element or `NULL` if the set is empty.

Remarks:

"IteratorsIterators" (page 220).

setst_first_const

Const version of `setst_first`.

```
const type*
setst_first_const(const SetSt(type) *set,
                  type);
```

set	The set.
type	Object type.

Return:

Element.

setst_last

Get the last element of the set and initialize the internal iterator.

```
type*
setst_last(SetSt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the last item or `NULL` if the set is empty.

Remarks:

"IteratorsIterators" (page 220).

setst_last_const

Const version of `setst_last`.

```
const type*
setst_last_const(const SetSt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setst_next

Get the next set item, after increasing the internal iterator.

```
type*
setst_next(SetSt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the next item or NULL if the iterator has reached the last.

Remarks:

Use setst_first to initialize the internal iterator *"IteratorsIterators"* (page 220).

setst_next_const

Const version of setst_next.

```
const type*
setst_next_const(const SetSt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setst_prev

Gets the previous element of the set, after decrementing the internal iterator.

```
type*
setst_prev(SetSt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the previous item or NULL if the iterator has reached the first.

Remarks:

Use setst_last to initialize the internal iterator on reversed loops *"IteratorsIterators"* (page 220).

setst_prev_const

Const version of `setst_prev`.

```
const type*
setst_prev_const(const SetSt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setst_foreach

Go through all the elements of the set. Use `setst_fornext` to close the loop.

```
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 Name of the variable 'element' within the loop. Adding the suffix `'_i'` we get the index.

set The set.

type Object type.

setst_foreach_const

Const version of `setst_foreach`.

```
void
setst_foreach_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setst_fornext

Close the loop opened by setst_foreach, increasing the internal iterator.

```
void
setst_fornext(type *elem,
              SetSt(type) *set,
              type);
```

elem Name of the variable 'element'. It must be the same as setst_foreach.

set The set.

type Object type.

setst_fornext_const

Const version of setst_fornext.

```
void
setst_fornext_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setst_forback

Go through all the elements of the set in reverse order. Use setst_forprev to close the loop.

```
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 Name of the variable 'element' within the loop. Adding the suffix '_i' we get the index.

set The set.

type Object type.

setst_forback_const

Const version of `setst_forback`.

```
void
setst_forback_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setst_forprev

Close the loop opened by `setst_forback`, decreasing the internal iterator.

```
void
setst_forprev(type *elem,
              SetSt(type) *set,
              type);
```

elem Name of the variable 'element'. It must be the same as `setst_foreach_rev`.

set The set.

type Object type.

setst_forprev_const

Const version of `setst_forprev`.

```
void
setst_forprev_const(const type *elem,
                    const SetSt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setpt_create

Create an empty pointer set.

```
SetPt(type)*
setpt_create(FPtr_compare func_compare,
             type);
```

func_compare Function to compare two elements. *“Sort and searchSort and search”* (page 216).

type Object type.

Return:

The new set.

setpt_destroy

Destroy a set and all its elements.

```
void
setpt_destroy(SetPt(type) **set,
              FPtr_destroy func_destroy,
              type);
```

set The set. Will be set to `NULL` after destruction.

func_destroy Function to destroy an element of the set *“DestructorsDestructors”* (page 214). If it is `NULL` only the set will be destroyed, but not its elements.

type Object type.

setpt_size

Get the number of set elements.

```
uint32_t
setpt_size(const SetPt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Number of items.

setpt_get

Search for an item in `O(logn)`. It is equivalent to `arrpt_bsearch`. The internal set iterator will be fixed in it.

```
type*
setpt_get(SetPt(type) *set,
          type *key,
          type);
```

```
Product key;
Product *pr;
key.id = 453;
pr = setpt_get(setpt, &key, Product);
```

set The set.

key Search key. It is a pointer to an object where only the relevant fields of the search must be initialized.

type Object type.

Return:

Pointer to the searched item if it exists, or `NULL` if not.

Remarks:

"IteratorsIterators" (page 220).

setpt_get_const

Const version of `setpt_get`.

```
const type*
setpt_get_const(const SetPt(type) *set,
                const type *key,
                type);
```

set The set.

key Search key.

type Object type.

Return:

Element.

setpt_insert

Insert a new item in the 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	The set.
value	Pointer to the element to insert.
type	Object type.

Return:

TRUE if the item has been inserted. FALSE if another element with the same key already exists.

Remarks:

Inserting or deleting elements invalidates the internal set iterator *"IteratorsIterators" (page 220)*. You must initialize it with setpt_first.

setpt_delete

Remove an item from the 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	The set.
key	Key to delete. It is a pointer to an object where only the relevant fields of the search must be initialized.
func_destroy	Element destructor. Can be `NULL`. See `setpt_destroy`.
type	Object type.

Return:

`TRUE` if the item has been deleted, or `FALSE` if there is no item with that key.

Remarks:

Inserting or deleting elements invalidates the internal set iterator *"IteratorsIterators"* (page 220). You must initialize it with `setpt_first`.

setpt_first

Get the first element of the set and initialize the internal iterator.

```
type*
setpt_first(SetPt(type) *set,
            type);
```

set	The set.
type	Object type.

Return:

Pointer to the first element or `NULL` if the set is empty.

Remarks:

"IteratorsIterators" (page 220).

setpt_first_const

Const version of `setpt_first`.

```
const type*
setpt_first_const(const SetPt(type) *set,
                  type);
```

set	The set.
type	Object type.

Return:

Element.

setpt_last

Get the last element of the set and initialize the internal iterator.

```
type*
setpt_last(SetPt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the last item or NULL if the set is empty.

Remarks:

"IteratorsIterators" (page 220).

setpt_last_const

Const version of setpt_last.

```
const type*
setpt_last_const(const SetPt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setpt_next

Get the next set item, after increasing the internal iterator.

```
type*
setpt_next(SetPt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the next item or `NULL` if the iterator has reached the last.

Remarks:

Use `setpt_first` to initialize the internal iterator *"IteratorsIterators"* (page 220).

setpt_next_const

Const version of `setpt_next`.

```
const type*
setpt_next_const(const SetPt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setpt_prev

Gets the previous element of the set, after decrementing the internal iterator.

```
type*
setpt_prev(SetPt(type) *set,
           type);
```

set The set.

type Object type.

Return:

Pointer to the previous item or `NULL` if the iterator has reached the first.

Remarks:

Use `setpt_last` to initialize the internal iterator on reversed loops *"IteratorsIterators"* (page 220).

setpt_prev_const

Const version of `setpt_prev`.

```
const type*
setpt_prev_const(const SetPt(type) *set,
                 type);
```

set The set.

type Object type.

Return:

Element.

setpt_foreach

Loop over all the elements of the set. Use `setpt_fornext` to close the loop.

```
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 Name of the variable 'element' within the loop. Adding the suffix `'_i'` we get the index.

set The set.

type Object type.

setpt_foreach_const

Const version of `setpt_foreach`.

```
void
setpt_foreach_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setpt_fornext

Close the loop opened by `setpt_foreach`, increasing the internal iterator.

```
void
setpt_fornext(type *elem,
              SetPt(type) *set,
              type);
```

elem Name of the variable 'element'. It must be the same as `setpt_foreach`.

set The set.

type Object type.

setpt_fornext_const

Const version of `setpt_fornext`.

```
void
setpt_fornext_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setpt_forback

Loop over all the elements of the set in reverse order. Use `setpt_forprev` to close the loop.

```
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 Name of the variable 'element' within the loop. Adding the suffix `'_i'` we get the index.

set The set.

type Object type.

setpt_forback_const

Const version of `setpt_forback`.

```
void
setpt_forback_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

setpt_forprev

Close the loop opened by `setpt_forback`, decreasing the internal iterator.

```
void
setpt_forprev(type *elem,
              SetPt(type) *set,
              type);
```

elem Name of the variable 'element'. It must be the same as `setpt_foreach_rev`.

set The set.

type Object type.

setpt_forprev_const

Const version of `setpt_forprev`.

```
void
setpt_forprev_const(const type *elem,
                    const SetPt(type) *set,
                    type);
```

elem Element.

set The set.

type Object type.

regex_create

Create a regular expression from a pattern.

```
RegEx*
regex_create(const char_t *pattern);
```

pattern Search pattern.

Return:

Regular expression (automata).

Remarks:

See *"Define patternsDefine patterns"* (page 223).

regex_destroy

Destroy a regular expression.

```
void
regex_destroy(RegEx **regex);
```

regex Regular expresion. Will be set to `NULL` after destruction.

regex_match

Check if a string matches the search pattern.

```
bool_t
regex_match(const RegEx *regex,
            const char_t *str);
```

regex Regular expresion.

str String to evaluate.

Return:

`TRUE` if the string is accepted by the regular expression.

dbind

Adds a structure/class field to its internal table within dbind.

```
void
dbind(type,
      mtype,
      name);
```

type	Type of structure or class.
mtype	Type of field to register.
name	Name of the field within the structure.

Remarks:

Errors will be generated at compile time if the indicated field does not belong to the structure. The method also works for classes in C++.

dbind_enum

Register an `enum` type value.

```
void
dbind_enum(type,
           value,
           const char_t *alias);
```

type	Enum type.
value	Value.
alias	Alias para el valor.

Remarks:

`dbind_enum(mode_t, ekIMAGE_ANALISYS, "Image Analisys");` it will use the string "Image Analisys" instead of "ekIMAGE_ANALISYS" for those I/O or interface operations that require displaying the literals of the enumeration. For example, to populate the fields of a `PopUp` linked with a data field.

dbind_create

Create an object of registered type, initializing its fields with the default values.

```
type*
dbind_create(type);
```

type	Object type.

Return:

Newly created object or `NULL` if *dbind* does not recognize the data type.

dbind_init

Initializes the fields of an object of a registered type with the default values.

```
void
dbind_init(type *obj,
           type);
```

obj Object whose memory has been reserved, but not initialized.

type Object type.

dbind_remove

Destroys the memory reserved by the fields of an object of registered type, but does not destroy the object itself.

```
void
dbind_remove(type *obj,
             type);
```

obj Object.

type Object type.

dbind_destroy

Destroy an object of registered type. The memory allocated to the fields and sub-objects will also be released recursively.

```
void
dbind_destroy(type **obj,
              type);
```

obj Object. Will be set to `NULL` after destruction.

type Object type.

dbind_destopt

Destructor optional. Like `dbind_destroy`, but accepting `NULL` values for the object.

```
void
dbind_destopt(type **obj,
              type);
```

obj Object to destroy.

type Object type.

dbind_read

Creates an object of a registered type from the data read from a stream.

```
type*
dbind_read(Stream *stm,
           type);
```

stm Reading stream.

type Object type to read.

Return:

Newly created object or NULL if there has been an error.

dbind_write

Write the content of an object of registered type in a write stream.

```
void
dbind_write(Stream *stm,
            const type *data,
            type);
```

stm Writing stream.

data Object to write.

type Type of object to write.

dbind_default

Set the default value of a field.

```
void
dbind_default(type,
              mtype,
              name,
              mtype value);
```

type Type of structure or class.

mtype Field type.

name Name of the field within the structure.

value Default value as of now.

dbind_range

Set the maximum and minimum value in numeric fields.

```
void
dbind_range(type,
            mtype,
            name,
            mtype min,
            mtype max);
```

type	Type of structure or class.
mtype	Field type.
name	Name of the field within the structure.
min	Minimum value.
max	Maximum value.

Remarks:

It will fail if used in non-numeric fields.

dbind_precision

Set the jump between two consecutive numerical values.

```
void
dbind_precision(type,
                mtype,
                name,
                mtype prec);
```

type	Type of structure or class.
mtype	Field type.
name	Name of the field within the structure.
prec	Accuracy (eg `.05f` in `real32_t` values).

Remarks:

It will fail if used in non-numeric fields.

dbind_increment

Sets the increment of a numerical value when clicking on a *"UpDown"* (page 314) control.

```
void
dbind_increment(type,
                mtype,
                name,
                mtype incr);
```

type	Type of structure or class.
mtype	Field type.
name	Name of the field within the structure.
incr	Increase.

Remarks:

It will fail if used in non-numeric fields.

dbind_suffix

Set a suffix that will be added to the numerical value when converted to text.

```
void
dbind_suffix(type,
             mtype,
             name,
             const char_t *suffix);
```

type	Type of structure or class.
mtype	Field type.
name	Name of the field within the structure.
suffix	Suffix.

Remarks:

It will fail if used in non-numeric fields.

listener

Create a listener. This function will link an event sender with the receiver, usually the application controller. The sender object is responsible for destroying the listener.

```
Listener*
listener(type *obj,
         FPtr_event_handler func_event_handler,
         type);
```

obj — Receiver object that will be passed as the first parameter to `func_event_handler`.

func_event_handler — *Callback* function that will be called when the event occurs. Also known as *event handler*.

type — The type of receiver object.

Return:

Listener object.

listen

Like `listener`, but used in C++ to define class *callbacks*. *"Use of C++"* (page 45).

```
void
listen(void);
```

listener_destroy

Destroy a listener.

```
void
listener_destroy(Listener **listener);
```

listener — Listener. Will be set to `NULL` after destruction.

Remarks:

The sender is responsible for destroying the listener.

listener_update

Update the receiver and event handler. It is equivalent to destroying it, and creating it again.

```
void
listener_update(Listener **listener,
                Listener *new_listener);
```

listener — The current listener.

new_listener — The new listener.

Remarks:

This method must be used within the sender.

listener_event

Launches an event from the sender to the receiver.

```
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	List through which the event will be sent.
type	Event code.
sender	Event sender.
params	Event parameters, or `NULL` if it doesn't have.
result	Event result, or `NULL` if not expected.
sender_type	Type of sender object.
params_type	Type of params object, or `void` if it does not have.
result_type	Type of result object, or `void` if it does not have.

Remarks:

This method must be invoked within the event sender.

listener_pass_event

Pass the received event to another object, changing only the sender. Useful for not generating a new `Event` object.

```
void
listener_pass_event(Listener *list,
                    Event *event,
                    sender_type *sender,
                    sender_type);
```

list	List through which the event will be resent.
event	Incoming event.
sender	The new event sender.
sender_type	Sender object type.

Remarks:

This method must be invoked within the event sender.

event_type

Get the event type.

```
uint32_t
event_type(const Event *event);
```

event Event.

Return:

The event type. Normally associated with a `enum`. Examples in `core_event_t`, `gui_event_t`.

event_sender

Get the event sender.

```
type*
event_sender(Event *event,
             type);
```

event Event.

type Sender type.

Return:

Sender.

event_params

Get the event parameters, encapsulated in a structure, which will be different depending on the event type.

```
type*
event_params(Event *event,
             type);
```

event Event.

type Parameters type.

Return:

Event parameters.

event_result

Gets an object to write the results of the event. Some events require the return of data by the receiver. The type of result object will depend on the type of event.

```
type*
event_result(Event *event,
             type);
```

event Event.

type Result type.

Return:

Event results.

keybuf_create

Create a buffer with keyboard status.

```
KeyBuf*
keybuf_create(void);
```

Return:

The buffer.

keybuf_destroy

Destroy the buffer.

```
void
keybuf_destroy(KeyBuf **bufer);
```

bufer The buffer. It will be set to NULL after the destruction.

keybuf_OnUp

Set the state of a key as released.

```
void
keybuf_OnUp(KeyBuf *bufer,
            const vkey_t key);
```

bufer The buffer.

key The key code.

Remarks:

Normally it will not be necessary to call this function. It will be done by `View` or the module that captures keyboard events.

keybuf_OnDown

Sets the state of a key as pressed.

```
void
keybuf_OnDown(KeyBuf *bufer,
              const vkey_t key);
```

bufer The buffer.

key The key code.

Remarks:

Normally it will not be necessary to call this function. It will be done by `View` or the module that captures keyboard events.

keybuf_clear

Clear the buffer. Set all keys as released.

```
void
keybuf_clear(KeyBuf *bufer);
```

bufer The buffer.

Remarks:

Normally it will not be necessary to call this function. It will be done by `View` or the module that captures keyboard events.

keybuf_pressed

Returns the state of a key.

```
bool_t
keybuf_pressed(const KeyBuf *bufer,
               const vkey_t key);
```

bufer The buffer.

key The key code.

Return:

Pulsed (TRUE) or released (FALSE).

keybuf_str

Returns a text string associated with a key.

```
void
keybuf_str(const vkey_t key);
```

key The key code.

keybuf_dump

Dump the buffer status into the *"Log"* (page 184).

```
void
keybuf_dump(const KeyBuf *bufer);
```

bufer The buffer.

hfile_dir

Check if the path is a directory.

```
bool_t
hfile_dir(const char_t *pathname);
```

pathname Name of the path to check. *"Filename and pathnameFilename and pathname"* (page 178).

Return:

TRUE if pathname is a directory. If it does not exist or is a file FALSE.

hfile_dir_create

Create all intermediate subdirectories of a path.

```
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 Name of the path to create. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if the entire path has been created, otherwise `FALSE`.

hfile_dir_destroy

Recursive destroy a directory and all its contents.

```
bool_t
hfile_dir_destroy(const char_t *pathname,
                  ferror_t *error);
```

pathname Directory path to destroy. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if the directory has been destroyed, or `FALSE` if there has been an error.

hfile_dir_list

Get a list of the contents of a directory.

```
ArrSt(DirEntry)*
hfile_dir_list(const char_t *pathname,
               ferror_t *error);
```

pathname Directory path to list. *"Filename and pathnameFilename and pathname"* (page 178).

error Error code if the function fails. Can be `NULL`.

Return:

Array of `DirEntry` with the content. It must be destroyed with `arrst_destroy(&array, hfile_dir_entry_remove, DirEntry)` when it is no longer necessary.

hfile_dir_entry_remove

Free the memory of an item in the directory listing.

```
void
hfile_dir_entry_remove(DirEntry *entry);
```

entry Element.

Remarks:

See hfile_dir_list.

hfile_date

Gets the most recent modification date of a file or directory.

```
Date
hfile_date(const char_t *pathname,
           const bool_t recursive);
```

pathname Path to file or directory. *"Filename and pathnameFilename and pathname"* (page 178).

recursive If pathname is a directory, it indicates whether to do a deep scan through subdirectories.

Return:

The modification date. If pathname does not exist kDATE_NULL.

Remarks:

If pathname is a directory, the modification dates of the files will be considered as well, not just the directory itself.

hfile_dir_sync

Synchronize the contents of two directories.

```
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	Source directory.
dest	Destination directory.
recursive	If `TRUE` recursive process the subdirectories.
remove_in_dest	If `TRUE` removes in `dest` those files/directories that are not in `src`.
except	List of file/directory names that will remain intact in `dest`.
except_size	Array `except` size.
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if everything went well, `FALSE` if there has been an error.

Remarks:

If a file is in `src` and not in `dest`, is copied to `dest`. If a file is newer in `src` it is also copied in `dest`. If a file exists in `dest` but not in `src` and `remove_in_dest` is `TRUE`, will be removed from `dest`. If the file exists in `except` array it will not be taken into account to copy or delete. If `recursive` is `TRUE` subdirectories will be processed in this way: If both subdirs exist in `src` and `dest` the same logic described here will be executed in both subdirs. If the subdir exists in `src` but not in `dest`, will be copied in its entirety to `dest`. If it exists in `dest` and not in `src` and `remove_in_dest` is `TRUE` will be completely removed from `dest`.

hfile_exists

Check if `pathname` exists in the file system.

```
bool_t
hfile_exists(const char_t *pathname,
             file_type_t *file_type);
```

pathname	Path of the directory or file to check. *"Filename and pathnameFilename and pathname"* (page 178).
file_type	Type of file. It can be `NULL`.

Return:

`TRUE` if `pathname` exists, `FALSE` if not.

hfile_is_uptodate

Check if a file is up to date. Consider that `dest` is a copy or depends on `src`.

```
bool_t
hfile_is_uptodate(const char_t *src,
                  const char_t *dest);
```

src Source file pathname.

dest Destiny file pathname.

Return:

`TRUE` if `dest` exists and is more recent than `src`, otherwise `FALSE`.

hfile_copy

Copy a file from one location to another.

```
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 of the file to copy. *"Filename and pathnameFilename and pathname"* (page 178).

dest Copy destination. If it is a directory it will have the same *filename* as the source. Otherwise, the copy will be made with another file name.

error Error code if the function fails. It can be `NULL`.

Return:

`TRUE` if the copy was successful. Otherwise `FALSE`.

hfile_buffer

Create a buffer with the contents of a file on disk.

```
Buffer*
hfile_buffer(const char_t *pathname,
             ferror_t *error);
```

pathname File path to load.

error Error code if the function fails. It can be `NULL`.

Return:

The buffer with the file data or `NULL` if the function fails.

Remarks:

It does not work with files larger than 4Gb (32-bit).

hfile_string

Create a string with the contents of a file on disk.

```
String*
hfile_string(const char_t *pathname,
             ferror_t *error);
```

pathname File path to load.

error Error code if the function fails. It can be `NULL`.

Return:

The string object with the text file data or `NULL` if the function fails.

Remarks:

It does not work with files larger than 4Gb (32-bit).

hfile_stream

Create a *"Memory streamMemory stream"* (page 195) and initializes it with the contents of a file.

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

pathname File path to load.

error Error code if the function fails. It can be `NULL`.

Return:

The stream initialized with the file data or `NULL` if the function fails.

Remarks:

It does not work with files larger than 4Gb (32-bit).

hfile_from_string

Create a file on disk with the contents of a *"Strings"* (page 192).

```
bool_t
hfile_from_string(const char_t *pathname,
                  const String *str,
                  ferror_t *error);
```

pathname File path to save.

str String to save to file.

error Error code if the function fails. It can be NULL.

Return:

TRUE if the file has been created successfully. Otherwise FALSE.

hfile_from_data

Create a file on disk with the contents of a generic block of memory.

```
bool_t
hfile_from_data(const char_t *pathname,
                const byte_t *data,
                const uint32_t size,
                ferror_t *error);
```

pathname File path to save.

data Block to save in the file.

size Block size in bytes.

error Error code if the function fails. It can be NULL.

Return:

TRUE if the file has been created successfully. Otherwise FALSE.

hfile_dir_loop

Browse all the files in a directory.

```
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 — Directory Path. *"Filename and pathnameFilename and pathname" (page 178).*

listener — *Callback* function to be called for each directory file.

subdirs — If TRUE the loop will process the subdirectories.

hiddens — If TRUE hidden files will be processed.

error — Error code if the function fails. It can be NULL.

Return:

TRUE tf the loop has been successfully completed. FALSE if an error has occurred.

Remarks:

For each file, an event will be sent to listener. Will be of type ekEFILE for regular files, ekEENTRY when enters a subdirectory and ekEEXIT when leaves it. The file attributes are sent in the event parameter as a EvFileDir object. The tour will continue until all files/subdirectories have been processed or returned FALSE in event_result . This controlled output will not be considered an error and this function will return TRUE.

hfile_appdata

Get the full path of a data file or application settings.

```
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 Relative file path.

Return:

The full path to the configuration file.

Remarks:

In many cases, applications need to create configuration files to remember user preferences or other data between sessions *"Home and AppDataHome and AppData"* (page 179). This function adds a relative path and file name and ensures that all intermediate directories will exist.

hfile_home_dir

Get the full path to a file in the user's **(home)** directory.

```
String*
hfile_home_dir(const char_t *path);
```

path Relative path from the **home** directory.

Return:

Absolute file path.

respack_destroy

Destroy a resource package.

```
void
respack_destroy(ResPack **pack);
```

pack Resource Package. Will be set to `NULL` after destruction.

respack_text

Get a text from a resource package.

```
const char_t*
respack_text(const ResPack *pack,
             const ResId id);
```

pack Resource package.

id Resource identifier.

Return:

UTF8 C string terminated in null character `'\0'`.

respack_file

Get a pointer to the contents of a file, included in a resource package.

```
const byte_t*
respack_file(const ResPack *pack,
             const ResId id,
             uint32_t *size);
```

pack Resource package.

id Resource identifier.

size Get the file size in bytes.

Return:

Pointer to file content (raw bytes).

date_system

Get the system date.

```
Date
date_system(void);
```

Return:

The current date.

date_add_seconds

Calculate the date resulting from adding an amount of seconds to another date.

```
Date
date_add_seconds(const Date *date,
                 int32_t seconds);
```

date	The base date.
seconds	The number of seconds. If it is positive we will obtain a future date. If negative, a past date.

Return:

The result date.

date_add_minutes

Calculate the date resulting from adding an amount of minutes to another date.

```
Date
date_add_minutes(const Date *date,
                 int32_t minutes);
```

date	The base date.
minutes	The number of minutes. If it is positive we will obtain a future date. If negative, a past date.

Return:

The result date.

date_add_hours

Calculate the date resulting from adding an amount of hours to another date.

```
Date
date_add_hours(const Date *date,
               int32_t hours);
```

date	The base date.
hours	The number of hours. If it is positive we will obtain a future date. If negative, a past date.

Return:

The result date.

date_add_days

Calculate the date resulting from adding an amount of days to another date.

```
Date
date_add_days(const Date *date,
              int32_t days);
```

date The base date.

days The number of days. If it is positive we will obtain a future date. If negative, a past date.

Return:

The result date.

date_year

Obtiene el año actual.

```
int16_t
date_year(void);
```

Return:

El año actual.

date_cmp

Compare two dates. The most recent date is considered greater.

```
int
date_cmp(const Date *date1,
         const Date *date2);
```

date1 First date to compare.

date2 Second date to compare.

Return:

Comparison result.

date_between

Check if a date is within a range.

```
bool_t
date_between(const Date *date,
             const Date *from,
             const Date *to);
```

date Date to check.

from Start date.

to Final date.

Return:

TRUE if date is between from and to.

date_is_null

Checks if a date is null.

```
bool_t
date_is_null(const Date *date);
```

date Date to check.

Return:

TRUE if date is null.

date_DD_MM_YYYY_HH_MM_SS

Convert a date to string, with the format DD/MM/YYYY-HH:MM:SS.

```
String*
date_DD_MM_YYYY_HH_MM_SS(const Date *date);
```

date Date.

Return:

String object with conversion.

date_YYYY_MM_DD_HH_MM_SS

Convert a date to string, with the format YYYY/MM/DD-HH:MM:SS.

```
String*
date_YYYY_MM_DD_HH_MM_SS(const Date *date);
```

date Date.

Return:

String object with conversion.

date_month_en

Get the name of the month, in English.

```
const char_t*
date_month_en(const month_t month);
```

month The month, usually obtained with `btime_date`.

Return:

UTF8 string with the name (January, February, ...).

date_month_es

Get the name of the month, in Spanish.

```
const char_t*
date_month_es(const month_t month);
```

month The month, usually obtained with `btime_date`.

Return:

UTF8 string with the name (Enero, Febrero, ...).

clock_create

Create a clock.

```
Clock*
clock_create(const real64_t interval);
```

interval Time interval for animation control (in seconds).

Return:

The new clock.

clock_destroy

Destroy the clock.

```
void
clock_destroy(Clock **clk);
```

clk Clock. Will be set to NULL after destruction.

clock_frame

Detect if a new sequence in an animation has expired.

```
bool_t
clock_frame(Clock *clk,
            real64_t *prev_frame,
            real64_t *curr_frame);
```

clk Clock.

prev_frame Time mark of the previous instant. Only relevant if returns TRUE.

curr_frame Time mark of the current instant. Only relevant if returns TRUE.

Return:

TRUE if the time has come to launch a new sequence. FALSE if we have to wait.

clock_reset

Set the clock to 0.0.

```
void
clock_reset(Clock *clk);
```

clk Clock.

clock_elapsed

Gets the time elapsed since the object was created or since the last call to clock_reset.

```
real64_t
clock_elapsed(Clock *clk);
```

clk Clock.

Return:

The number of seconds (with precision of micro-seconds 0.000001).

Chapter 38

Geom2D library

38.1. Types and Constants

kZERO

The (0,0) vector.

```
const V2Df kV2D_ZEROf;
const V2Dd kV2D_ZEROd;
const V2D V2D::kZERO;
```

kX

The (1,0) vector.

```
const V2Df kV2D_Xf;
const V2Dd kV2D_Xd;
const V2D V2D::kX;
```

kY

The (,1) vector.

```
const V2Df kV2D_Yf;
const V2Dd kV2D_Yd;
const V2D V2D::kY;
```

kZERO

`[0,0]` value.

```
const S2Df kS2D_ZEROf;
const S2Dd kS2D_ZEROd;
const S2D S2D::kZERO;
```

kZERO

Value `[0,0,0,0]`.

```
const R2Df kR2D_ZEROf;
const R2Dd kR2D_ZEROd;
const R2D R2D::kZERO;
```

kIDENT

Represents the identity transformation.

```
const T2Df kT2D_IDENTf;
const T2Dd kT2D_IDENTd;
const T2D T2D::kIDENT;
```

kNULL

Represents a null circle (no geometry).

```
const Cir2Df kCIR2D_NULLf;
const Cir2Dd kCIR2D_NULLd;
const Cir2D Cir2D::kNULL;
```

kNULL

Represents a null box (without geometry).

```
const Box2Df kBOX2D_NULLf;
const Box2Dd kBOX2D_NULLd;
const Box2D Box2D::kNULL;
```

struct V2D

Represents a 2d vector or point. *"2D Vectors"* (page 237).

```
struct V2Df
{
    real32_t x;
    real32_t y;
```

```
};

struct V2Dd
{
    real64_t x;
    real64_t y;
};

struct V2D
{
    real x;
    real y;
};
```

x Coordinate x.

y Coordinate y.

struct S2D

Represents a 2d size. *"2D Size"* (page 240).

```
struct S2Df
{
    real32_t width;
    real32_t height;
};

struct S2Dd
{
    real64_t width;
    real64_t height;
};

struct S2D
{
    real width;
    real height;
};
```

width Width.

height Height.

struct R2D

2d rectangle. *"2D Rectangles"* (page 240).

```
struct R2Df
{
```

```
    V2Df pos;
    S2Df size;
};

struct R2Dd
{
    V2Dd pos;
    S2Dd size;
};

struct R2D
{
    V2D pos;
    S2D size;
};
```

`pos` Origin.

`size` Size.

struct T2D

2d affine transformation. *"2D Transformations"* (page 241).

```
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` Component i of the linear transformation.

`j` Component j of the linear transformation.

p Position.

struct Seg2D

2d line segment. *"2D Segments"* (page 246).

```
struct Seg2Df
{
    V2Df p0;
    V2Df p1;
};

struct Seg2Dd
{
    V2Dd p0;
    V2Dd p1;
};

struct Seg2D
{
    V2D p0;
    V2D p1;
};
```

p0 Coordinate of the first point of the segment.

p1 Coordinate of the second point of the segment.

struct Cir2D

2d circle. *"2D Circles"* (page 247).

```
struct Cir2Df
{
    V2Df c;
    real32_t r;
};

struct Cir2Dd
{
    V2Dd c;
    real64_t r;
};

struct Cir2D
{
    V2D c;
    real r;
};
```

`c` Center.

`r` Radix.

struct Box2D

2d bounding box. *"2D Boxes"* (page 247).

```
struct Box2Df
{
    V2Df min;
    V2Df max;
};

struct Box2Dd
{
    V2Dd min;
    V2Dd max;
};

struct Box2D
{
    V2D min;
    V2D max;
};
```

`min` Minimum bounding coordinate.

`max` Maximum bounding coordinate.

struct OBB2D

2d Oriented Bounding Box. *"2D Oriented Boxes"* (page 247).

```
struct OBB2Df;

struct OBB2Dd;

struct OBB2D;
```

struct Tri2D

2d triangle. *"2D Triangles"* (page 249).

```
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` Coordinate of the first point of the triangle.

`p1` Coordinate of the second point of the triangle.

`p2` Coordinate of the third point of the triangle.

struct Pol2D

2d convex polygon. *"2D Polygons"* (page 250).

```
struct Pol2Df;

struct Pol2Dd;

struct Pol2D;
```

struct Col2D

Collision data in 2d. *"2D Collisions"* (page 253).

```
struct Col2Df;

struct Col2Dd;

struct Col2D;
```

38.2. Functions

v2d

Create a 2d vector from its components.

```
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 X coordinate.

y Y coordinate.

Return:

2d vector.

v2d_tof

Convert a vector from double to float.

```
V2Df
v2d_tof(const V2Dd *v);
```

v Vector.

Return:

The 2d vector in simple precision.

v2d_tod

Convert a vector from float to double.

```
V2Dd
v2d_tod(const V2Df *v);
```

v Vector.

Return:

The 2d vector in double precision.

v2d_tofn

Converts a vector array from double to float.

```
void
v2d_tofn(V2Df *vf,
         const V2Dd *vd,
         const uint32_t n);
```

vf The destination array.

vd The source array.

n Number of elements.

v2d_todn

Converts a vector array from float to double.

```
void
v2d_todn(V2Dd *vd,
         const V2Df *vf,
         const uint32_t n);
```

vd The destination array.

vf The source array.

n Number of elements.

v2d_add

Add two vectors.

```
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.

Return:

The result vector.

v2d_sub

Subtract two vectors.

```
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.

Return:

The result vector.

v2d_mul

Multiply a vector by a scalar.

```
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 Scalar.

Return:

The result vector.

v2d_from

Create a vector from a point and a direction.

```
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	Initial vector.
dir	Direction.
length	Length.

Return:

The result vector.

Remarks:

It will perform the operation `r = v + length * dir`. `dir` does not need to be unitary, in which case `length` will behave as a scale factor.

v2d_mid

Returns the midpoint of two points.

```
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 First point.

v2 Second point.

Return:

The middle point.

v2d_unit

Unit vector (direction) from 1 to 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 Point 1 (origin).

v2 Point 2 (destination).

dist Distance between points. Can be NULL.

Return:

The unit vector.

v2d_unit_xy

Unit vector (direction) from 1 to 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 X coordinate of point 1 (origin).

y1 Y coordinate of point 1 (origin).

x2 X coordinate of point 2 (destination).

y2 Y coordinate of point 2 (destination).

dist Distance between points. Can be NULL.

Return:

The unit vector.

v2d_perp_pos

Gets the positive perpendicular vector.

```
V2Df
v2d_perp_posf(const V2Df *v);

V2Dd
v2d_perp_posd(const V2Dd *v);

V2D
V2D::perp_pos(const V2D *v);
```

v Initial vector.

Return:

The perpendicular vector.

Remarks:

It is the perpendicular obtained by positive angle (+ /2).

v2d_perp_neg

Gets the negative perpendicular vector.

```
V2Df
v2d_perp_negf(const V2Df *v);

V2Dd
v2d_perp_negd(const V2Dd *v);

V2D
V2D::perp_neg(const V2D *v);
```

v Initial vector.

Return:

The perpendicular vector.

Remarks:

It is the perpendicular obtained by negative angle (- /2).

v2d_from_angle

Gets the vector resulting from applying a rotation to the 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 Angle.

Return:

The vector.

Remarks:

For `a=0` we get `[1,0]`. For `aπ=/2` `[0,1]`.

v2d_norm

Normalize a vector, that is, make it a vector of length = 1.

```
bool_t
v2d_normf(V2Df *v);
```

```
bool_t
v2d_normd(V2Dd *v);

bool_t
V2D::norm(V2D *v);
```

v Vector that will be normalized.

Return:

FALSE if the vector cannot be normalized (vector 0).

v2d_length

Calculate the length of a vector.

```
real32_t
v2d_lengthf(const V2Df *v);

real64_t
v2d_lengthd(const V2Dd *v);

real
V2D::length(const V2D *v);
```

v Vector.

Return:

The vector module.

v2d_sqlength

Calculate the square of the length of a vector.

```
real32_t
v2d_sqlengthf(const V2Df *v);

real64_t
v2d_sqlengthd(const V2Dd *v);

real
V2D::sqlength(const V2D *v);
```

v Vector.

Return:

The square of the vector modulus.

Remarks:

Avoid using the square root, so it is more efficient than `v2d_lengthf`. Often used to compare distances.

v2d_dot

Product of two vectors.

```
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.

Return:

Scalar product.

v2d_dist

Calculate the distance between two points.

```
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 The first point.

v2 The second point.

Return:

Distance.

v2d_sqdist

Calculate the square of the distance between two points.

```
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 The first point.

v2 The second point.

Return:

The distance squared.

Remarks:

It avoids using the square root, so it is more efficient than `v2d_distf`. Often used to compare distances.

v2d_angle

Calculate the angle formed by two vectors.

```
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.

Return:

The angle in radians (-Pi, Pi)

Remarks:

Positive angles go from v1 to v2 counterclockwise. For angles greater than Pi radians (180°) it will return negative (clockwise).

v2d_rotate

Apply a rotation to a 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 to be rotated (origin/destination.

a Angle in radians.

Remarks:

This function involves calculating the sine and cosine. Use `t2d_vmultnf` if you have to apply the same rotation to multiple vectors.

s2d

Create a 2d size from two values.

```
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 Width.

height Height.

Return:

The size.

r2d

Create a rectangle from its components.

```
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 Origin x coordinate.

y Coordinate and origin.

width Width.

height Height.

Return:

The rectangle.

r2d_center

Gets the center point of the rectangle.

```
V2Df
r2d_centerf(const R2Df *r2d);

V2Dd
r2d_centerd(const R2Dd *r2d);
```

```
V2D
R2D::center(const R2D *r2d);
```

r2d Rectangle.

Return:

The center.

r2d_collide

Check if two rectangles collide.

```
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 Rectangle 1.

r2d2 Rectangle 2.

Return:

`TRUE` if there is collision, `FALSE` if they are separated.

r2d_contains

Check if a point is inside the rectangle.

```
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 Rectangle.

x X coordinate of the point.

y Coordinate and point.

Return:

`TRUE` if the point is inside.

r2d_clip

Check if a rectangle, or part of it, is contained in another rectangle.

```
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 Container rectangle.

r2d Rectangle to check.

Return:

`TRUE` if the `r2d` rectangle is completely outside of `viewport`.

Remarks:

Useful to avoid processing or drawing objects that are totally outside the viewing area.

r2d_join

Join two rectangles into one.

```
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 Destination rectangle. Its position and size will be modified to contain `src`.

src Rectangle to be added to `r2d`.

t2d_tof

Converts a transformation from double to float.

```
void
t2d_tof(T2Df *dest,
        const T2Dd *src);
```

dest Destination transformation.

src Origin transformation.

t2d_tod

Converts a transform from float to double.

```
void
t2d_tod(T2Dd *dest,
        const T2Df *src);
```

dest Destination transformation.

src Origin transformation.

t2d_move

Multiply a transformation by a translation `t2d = 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 Result transformation.

src Initial transformation.

x X coordinate of displacement.

y Y coordinate of displacement.

Remarks:

`dest` and `src` can point to the same matrix.

t2d_rotate

Multiply a transformation by a rotation `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 Result transformation.

src Initial transformation.

a Rotation angle in radians. Positive angles are those that rotate from the X axis to the Y axis.

Remarks:

`dest` and `src` can point to the same matrix.

t2d_scale

Multiply a transformation by an scale `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	Result transformation.
src	Initial transformation.
sx	Scaling on the x axis.
sy	Scaling on the y axis.

Remarks:

`dest` and `src` can point to the same matrix.

t2d_invfast

Calculate the inverse transformation, assuming the input is orthogonal.

```
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 Inverse transformation.

src Initial transformation.

Remarks:

The transformation will be orthogonal only if it contains rotations and translations, otherwise the result of applying it will be unpredictable. `dest` and `src` can point to the same matrix.

t2d_inverse

Calculate the inverse transformation.

```
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 Inverse transformation.

src Initial transformation.

Remarks:

`dest` and `src` can point to the same matrix.

t2d_mult

Multiply two transformations `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 Result transformation.

src1 First operating.

src2 Second operating.

Remarks:

`dest`, `src1` and `src2` can point to the same matrix.

t2d_vmult

Transform a 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 Transformed vector.

t2d Transformation.

src Original vector.

Remarks:

`dest` and `src` can point to the same vector.

t2d_vmultn

Transform a vector list `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	Transformed vector array.
t2d	Transformation.
src	Original vector array.
n	Number of vectors in `src`.

Remarks:

`dest` and `src` can point to the same array.

t2d_decompose

Gets the position, rotation, and scaling of a transformation.

```
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 Transformation.

pos Position. Can be NULL.

a Angle in radians (- /2, /2). Can be NULL.

sc Scaled. Can be NULL.

Remarks:

If the transformation is not made up of a sequence of translations, rotations, and scales, the result will not be valid.

seg2d

Create a 2d segment from its components.

```
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 X coordinate of the first point.

y0 Y coordinate of the first point.

x1 X coordinate of the second point.

y1 Y coordinate of the second point.

Return:

The 2d segment.

seg2d_v

Create a 2d segment from two points.

```
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 First point.

p1 Second point.

Return:

The 2d segment.

seg2d_length

Gets the length of the segment.

```
real32_t
seg2d_lengthf(const Seg2Df *seg);

real64_t
seg2d_lengthd(const Seg2Dd *seg);

real
Seg2D::length(const Seg2D *seg);
```

seg Segment.

Return:

Length.

seg2d_sqlength

Gets the square of the segment length.

```
real32_t
seg2d_sqlengthf(const Seg2Df *seg);

real64_t
seg2d_sqlengthd(const Seg2Dd *seg);
```

```
real
Seg2D::sqlength(const Seg2D *seg);
```

seg Segment.

Return:

Length square.

Remarks:

Avoid calculating square roots if we are only interested in comparing measurements.

seg2d_eval

Gets the point in the segment based on the parameter.

```
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 Segment.

t Parameter.

Return:

Point on the segment (or on the line that contains it).

Remarks:

If `t=0` it returns `p0`. If `t=1` it returns `p1`. Values between `(0,1)` points within the segment. Other values, points on the line that contains the segment.

seg2d_close_param

Gets the parameter of the segment closest to a given point.

```
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 Segment.

pnt Point.

Return:

Parameter. See `seg2d_evalf`.

seg2d_point_sqdist

Gets the squared distance from a point to the segment.

```
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 Segment.

pnt Point.

t Parameter on the line that contains the segment. See `seg2d_close_paramf`. It can be `NULL` if we don't need this value.

Return:

Distance square.

seg2d_sqdist

Gets the squared distance between two segments.

```
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 First segment.

seg2 Second segment.

t1 Nearest parameter in `seg1`. It can be `NULL` if we don't need this value.

t2 Nearest parameter in `seg2`. It can be `NULL` if we don't need this value.

Return:

Distance square.

cir2d

Create a 2d circle from its components.

```
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 Center x coordinate.

y Center y coordinate.

r Radius.

Return:

The 2d circle.

cir2d_from_box

Create a circle containing a 2D box.

```
Cir2Df
cir2d_from_boxf(const B2D *box);

Cir2Dd
cir2d_from_boxd(const B2D *box);

Cir2D
Cir2D::from_box(const B2D *box);
```

box The box.

Return:

The circle.

cir2d_from_points

Create a circle containing a set of points.

```
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 The points vector.

n The number of points.

Return:

The circle.

Remarks:

The center will be the midpoint of the set. The radius will be the distance to the farthest point from that center. Provides a good fit with linear cost.

cir2d_minimum

Calculate the circle of minimum radius that contains a set of points.

```
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 The points vector.

n The number of points.

Return:

The circle.

Remarks:

Provides optimal adjustment in linear time. However, it is slower than `cir2d_from_pointsf`.

cir2d_area

Gets the area of the circle.

```
real32_t
cir2d_areaf(const Cir2Df *cir);

real64_t
cir2d_aread(const Cir2Dd *cir);

real
Cir2D::area(const Cir2D *cir);
```

cir The circle.

Return:

The area $\pi(r^2)$.

cir2d_is_null

Check if a circle is null (dimensionless).

```
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 The circle.

Return:

`TRUE` if it is null, `FALSE` if it contains any point.

Remarks:

A single point is a valid circle with radius = 0.

box2d

Create a new box with the indicated limits.

```
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 The lower limit on X.

minY The lower limit on Y.

maxX The upper limit on X.

maxY The upper limit on Y.

Return:

The newly created box.

box2d_from_points

Create a new box containing a set of points.

```
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 2d point vector.

n Number of points in vector.

Return:

The newly created box.

box2d_center

Returns the center point.

```
V2Df
box2d_centerf(const Box2Df *box);

V2Dd
box2d_centerd(const Box2Dd *box);

V2D
Box2D::center(const Box2D *box);
```

box The container.

Return:

Center coordinates.

box2d_add

Expand the dimensions of the box to contain the entry point. If the point is already within its area, the box is not modified.

```
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 The container.

p The point to include.

box2d_addn

Expand the dimensions of the box to contain several points. It is equivalent to calling the method box2d_addf successively.

```
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 The container.

p Vector points to include.

n Number of points.

box2d_add_circle

Expand the dimensions of the container to accommodate a circle.

```
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 The container.

cir Circle.

box2d_merge

Expand the dimensions of `dest` to contain `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 The container that will be expanded.

src The container that must be added.

box2d_segments

Gets the four segments that make up the box.

```
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 The container.

segs Array of at least four segments.

box2d_area

Gets the area of the box.

```
real32_t
box2d_areaf(const Box2Df *box);

real64_t
box2d_aread(const Box2Dd *box);

real
Box2D::area(const Box2D *box);
```

box The container.

Return:

The area `(width * height)`.

box2d_is_null

Check if a container is null (without any geometry inside).

```
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 The container.

Return:

`TRUE` if is null, `FALSE` if contains any geometry.

obb2d_create

Create a new oriented box.

```
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	The central point.
width	The width of the box.
height	The height of the box.
angle	The angle with respect to the X axis, in radians.

Return:

The newly created box.

Remarks:

Positive angles are those that rotate from the X axis to the Y axis.

obb2d_from_line

Create a box from a segment.

```
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 The first point of the segment.

p1 The second point of the segment.

thickness The "thickness" of the segment.

Return:

The newly created box.

Remarks:

The width of the box will correspond to the length of the segment. The height will be `thickness` and the center will be the midpoint of the segment.

obb2d_from_points

Create an oriented box from a set of points.

```
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 Points array.

n Number of points.

Return:

The newly created box.

Remarks:

A good fit will be produced in "elongated" point distributions by calculating the covariance matrix and projecting points onto the director vector of that distribution. However, it does not provide the minimum volume box.

obb2d_copy

Create a copy of the box.

```
OBB2Df*
obb2d_copyf(const OBB2Df obb);

OBB2Dd*
obb2d_copyd(const OBB2Dd obb);

OBB2D*
OBB2D::copy(const OBB2D obb);
```

obb Original box.

Return:

The copy.

obb2d_destroy

Destroy the box.

```
void
obb2d_destroyf(OBB2Df **obb);

void
obb2d_destroyd(OBB2Dd **obb);

void
OBB2D::destroy(OBB2D **obb);
```

obb The box. Will be set to `NULL` after destruction.

obb2d_update

Update the box parameters.

```
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	The box to update.
center	The central point.
width	The width.
height	The height.
angle	The angle.

Remarks:

See `obb2d_createf`.

obb2d_move

Move the box on the plane.

```
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	The box.
offset_x	X displacement.
offset_y	Y displacement.

obb2d_transform

Apply a transformation to the box.

```
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 The box.

t2d Affine transformation.

obb2d_corners

Gets the vertices bounding the box.

```
const V2Df*
obb2d_cornersf(const OBB2Df *obb);

const V2Dd*
obb2d_cornersd(const OBB2Dd *obb);

const V2D*
OBB2D::corners(const OBB2D *obb);
```

obb The box.

Return:

Pointer to an array of 4 vertices.

Remarks:

Do not modify the returned array. Copy if necessary.

obb2d_center

Gets the center point of the box.

```
V2Df
obb2d_centerf(const OBB2Df *obb);

V2Dd
obb2d_centerd(const OBB2Dd *obb);
```

```
V2D
OBB2D::center(const OBB2D *obb);
```

obb The box.

Return:

Center.

obb2d_width

Get the width of the box.

```
real32_t
obb2d_widthf(const OBB2Df *obb);

real64_t
obb2d_widthd(const OBB2Dd *obb);

real
OBB2D::width(const OBB2D *obb);
```

obb The box.

Return:

The width.

obb2d_height

Get the height of the box.

```
real32_t
obb2d_heightf(const OBB2Df *obb);

real64_t
obb2d_heightd(const OBB2Dd *obb);

real
OBB2D::height(const OBB2D *obb);
```

obb The box.

Return:

The height.

obb2d_angle

Get the angle of the box.

```
real32_t
obb2d_anglef(const OBB2Df *obb);

real64_t
obb2d_angled(const OBB2Dd *obb);

real
OBB2D::angle(const OBB2D *obb);
```

obb The box.

Return:

The angle in radians with respect to the X axis.

obb2d_area

Gets the box area.

```
real32_t
obb2d_areaf(const OBB2Df *obb);

real64_t
obb2d_aread(const OBB2Dd *obb);

real
OBB2D::area(const OBB2D *obb);
```

obb The box.

Return:

The area (`width * height`).

obb2d_box

Get the box limits.

```
Box2Df
obb2d_boxf(const OBB2Df *obb);

Box2Dd
obb2d_boxd(const OBB2Dd *obb);

Box2D
OBB2D::box(const OBB2D *obb);
```

obb The box.

Return:

Box aligned with the axes, defined by the minimum and maximum vectors.

tri2d

Triangle from its coordinates.

```
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 X coordinate of the first point.

y0 Y coordinate of the first point.

x1 X coordinate of the second point.

y1 Y coordinate of the second point.

x2 X coordinate of the third point.

y2 Y coordinate of the third point.

Return:

The triangle.

tri2d_v

Triangle from three points.

```
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 First point.

p1 Second point.

p2 Third point.

Return:

The triangle.

tri2d_transform

Apply a transformation to the triangle.

```
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 The triangle.

t2d Affine transformation.

tri2d_area

Gets the area of the triangle.

```
real32_t
tri2d_areaf(const Tri2Df *tri);

real64_t
tri2d_aread(const Tri2Dd *tri);

real
Tri2D::area(const Tri2D *tri);
```

tri The triangle.

Return:

The area.

tri2d_ccw

Obtains the order of the travel of the points of the triangle.

```
bool_t
tri2d_ccwf(const Tri2Df *tri);

bool_t
tri2d_ccwd(const Tri2Dd *tri);

bool_t
Tri2D::ccw(const Tri2D *tri);
```

tri The triangle.

Return:

TRUE counter-clockwise sense. FALSE *clockwise.*

Remarks:

See *"CW and CCW anglesCW and CCW angles"* (page 238).

tri2d_centroid

Gets the centroid (center of mass) of the triangle.

```
V2Df
tri2d_centroidf(const Tri2Df *tri);

V2Dd
```

```
tri2d_centroidd(const Tri2Dd *tri);

V2D
Tri2D::centroid(const Tri2D *tri);
```

tri The triangle.

Return:

Center of mass.

pol2d_create

Create a new polygon.

```
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 List of points that make up the polygon.

n Number of points.

Return:

The polygon created.

pol2d_convex_hull

Creates the minimum convex polygon that surrounds a set of points *(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 Points list.

n Number of points.

Return:

The polygon.

pol2d_copy

Create a copy of the polygon.

```
Pol2Df*
pol2d_copyf(const Pol2Df *pol);

Pol2Dd*
pol2d_copyd(const Pol2Dd *pol);

Pol2D*
Pol2D::copy(const Pol2D *pol);
```

pol The original polygon.

Return:

The copy.

pol2d_destroy

Destroy the polygon.

```
void
pol2d_destroyf(Pol2Df **pol);

void
pol2d_destroyd(Pol2Dd **pol);

void
Pol2D::destroy(Pol2D **pol);
```

pol The polygon. Will be set to `NULL` after destruction.

pol2d_transform

Apply a 2D transformation.

```
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 The polygon.

t2d 2D transformation.

Remarks:

The polygon does not save the original coordinates. Successive transformations will accumulate.

pol2d_points

Gets the vertices that make up the polygon.

```
const V2Df*
pol2d_pointsf(const Pol2Df *pol);

const V2Dd*
pol2d_pointsd(const Pol2Dd *pol);

const V2D*
Pol2D::points(const Pol2D *pol);
```

pol The polygon.

Return:

Pointer to an array of vertices.

Remarks:

Do not modify the returned array. Copy if necessary.

pol2d_n

Gets the number of vertices that make up the polygon.

```
uint32_t
pol2d_nf(const Pol2Df *pol);

uint32_t
```

```
pol2d_nd(const Pol2Dd *pol);

uint32_t
Pol2D::n(const Pol2D *pol);
```

pol The polygon.

Return:

The number of vertices.

Remarks:

It is the same value as the one used in the constructor `pol2d_createf`.

pol2d_area

Gets the area of the polygon.

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

real64_t
pol2d_aread(const Pol2Dd *pol);

real
Pol2D::area(const Pol2D *pol);
```

pol The polygon.

Return:

The area.

pol2d_box

Gets the geometric limits of the polygon.

```
Box2Df
pol2d_boxf(const Pol2Df *pol);

Box2Dd
pol2d_boxd(const Pol2Dd *pol);

Box2D
Pol2D::box(const Pol2D *pol);
```

pol The polygon.

Return:

Box aligned with the axes, defined by the minimum and maximum vectors.

pol2d_ccw

Gets the winding order of the polygon points.

```
bool_t
pol2d_ccwf(const Pol2Df *pol);

bool_t
pol2d_ccwd(const Pol2Dd *pol);

bool_t
Pol2D::ccw(const Pol2D *pol);
```

pol The polygon.

Return:

TRUE counter-clockwise. FALSE clockwise.

pol2d_convex

Gets whether or not the polygon is convex.

```
bool_t
pol2d_convexf(const Pol2Df *pol);

bool_t
pol2d_convexd(const Pol2Dd *pol);

bool_t
Pol2D::convex(const Pol2D *pol);
```

pol The polygon.

Return:

TRUE if is convex. FALSE if no.

pol2d_centroid

Gets the centroid (center of mass) of the polygon.

```
V2Df
pol2d_centroidf(const Pol2Df *pol);
```

```
V2Dd
pol2d_centroidd(const Pol2Dd *pol);

V2D
Pol2D::centroid(const Pol2D *pol);
```

pol The polygon.

Return:

Center of mass.

pol2d_visual_center

Gets the visual center or label point.

```
V2Df
pol2d_visual_centerf(const Pol2Df *pol);

V2Dd
pol2d_visual_centerd(const Pol2Dd *pol);

V2D
Pol2D::visual_center(const Pol2D *pol);
```

pol The polygon.

Return:

The labeling center.

Remarks:

It corresponds to a point within the polygon located at a maximum distance from any edge. In convex polygons it will coincide with the centroid. It implements an adaptation of the **polylabel** algorithm of the project MapBox[1].

pol2d_triangles

Gets a list of triangles that make up the polygon.

```
ArrSt(Tri2Df)*
pol2d_trianglesf(const Pol2Df *pol);

ArrSt(Tri2Df)*
pol2d_trianglesd(const Pol2Dd *pol);
```

[1] https://github.com/mapbox/polylabel

```
ArrSt(Tri2Df)*
Pol2D::triangles(const Pol2D *pol);
```

pol The polygon.

Return:

Triangle array. Must be destroyed with `arrst_destroy(&triangles, NULL, Tri2Df)`.

Remarks:

The union of all the triangles corresponds to the original polygon.

pol2d_convex_partition

Gets a list of the convex polygons that make up the polygon.

```
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 The polygon.

Return:

Array of convex polygons. It must be destroyed with `arrst_destroy(&polys, pol2d_destroyf, Pol2Df)`.

Remarks:

The union of all polygons corresponds to the original polygon.

col2d_point_point

Point-point collision.

```
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 First point.

pnt2 Second point.

tol Tolerance. Minimum distance to be considered a collision.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_segment_point

Segment-point collision.

```
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 Segment.

pnt Point.

tol Tolerance. Minimum distance to be considered a collision.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_segment_segment

Segment-segment collision.

```
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 First segment.

seg2 Second segment.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_circle_point

Circle-point collision.

```
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 Circle.

pnt Point.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_circle_segment

Circle-segment collision.

```
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 Circle.

seg Segment.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_circle_circle

Circle-circle collision.

```
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 First circle.

cir2 Second circle.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_box_point

Box-point collision.

```
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 Box.

pnt Point.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_box_segment

Box-segment collision.

```
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 Box.

seg Segment.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_box_circle

Box-circle collision.

```
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 Box.

cir Circle.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_box_box

Box-box collision.

```
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 First box.

box2 Second box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_obb_point

Point-oriented box collision.

```
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 Oriented box.

pnt Point.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_obb_segment

Segment-oriented box collision.

```
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 Oriented box.

seg Segment.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_obb_circle

Collision-oriented box-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 Oriented box.

cir Circle.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_obb_box

Box-oriented box collision.

```
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 Oriented box.

box Aligned box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_obb_obb

Oriented Box-Oriented Box collision.

```
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 First oriented box.

obb2 Second oriented box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_tri_point

Triangle-point collision.

```
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	Triangle.
pnt	Point.
col	Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_tri_segment

Triangle-segment collision.

```
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 Triangle.

seg Segment.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_tri_circle

Triangle-circle collision.

```
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 Triangle.

cir Circle.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_tri_box

Triangle-box collision.

```
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 Triangle.

box Aligned box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_tri_obb

Triangle-oriented box collision.

```
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 Triangle.

obb Oriented box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_tri_tri

Triangle-triangle collision.

```
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 First triangle.

tri2 Second triangle.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_poly_point

Polygon-point collision.

```
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 Polygon.

pnt Point.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_poly_segment

Polygon-segment collision.

```
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 Polygon.

seg Segment.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_poly_circle

Polygon-circle collision.

```
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 Polygon.

cir Circle.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_poly_box

Polygon-box collision.

```
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 Polygon.

cir Box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_poly_obb

Polygon-box collision.

```
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 Polygon.

cir Oriented box.

col Detailed data of the collision. It can be NULL if we don't need additional information.

Return:

TRUE if the objects intersect, FALSE otherwise.

col2d_poly_tri

Polygon-triangle collision.

```
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 Polygon.

tri Triangle.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

col2d_poly_poly

Polygon-polygon collision.

```
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 First polygon.

pol2 Second polygon.

col Detailed data of the collision. It can be `NULL` if we don't need additional information.

Return:

`TRUE` if the objects intersect, `FALSE` otherwise.

Chapter 39

Draw2D library

39.1. Types and Constants

kCOLOR_TRANSPARENT

Totally transparent color, absence of color or null color.

```
const color_t kCOLOR_TRANSPARENT;
```

kCOLOR_DEFAULT

Default color.

```
const color_t kCOLOR_DEFAULT;
```

kCOLOR_BLACK

BLACK color rgb(0,0,0).

```
const color_t kCOLOR_BLACK;
```

kCOLOR_WHITE

WHITE color rgb(255,255,255).

```
const color_t kCOLOR_WHITE;
```

kCOLOR_RED

RED color rgb(255,0,0).

```
const color_t kCOLOR_RED;
```

kCOLOR_GREEN

GREEN color rgb(0,255,0).

```
const color_t kCOLOR_GREEN;
```

kCOLOR_BLUE

BLUE color rgb(0,0,255).

```
const color_t kCOLOR_BLUE;
```

kCOLOR_YELLOW

YELLOW color rgb(255,255,0).

```
const color_t kCOLOR_YELLOW;
```

kCOLOR_CYAN

CYAN color rgb(0,255,255).

```
const color_t kCOLOR_CYAN;
```

kCOLOR_MAGENTA

MAGENTA color rgb(255,0,255).

```
const color_t kCOLOR_MAGENTA;
```

enum pixformat_t

Pixel format in an image. Number of bits per pixel and color model.

ekINDEX1	1 bit per pixel. 2 colors, indexed.
ekINDEX2	2 bits per pixel. 4 colors, indexed.
ekINDEX4	4 bits per pixel. 16 colors, indexed.
ekINDEX8	8 bits per pixel. 256 colors, indexed.
ekGRAY8	8 bits per pixel in grayscale. 256 shades of gray.

`ekRGB24` 24 bits per RGB pixel. 8 bits per channel (red, green, blue). The lowest order byte corresponds to the red one and the highest one to the blue one.

`ekRGBA32` 32 bits per pixel RGBA. 8 bits per channel (red, green, blue, alpha). The lowest order byte corresponds to the red one and the highest one to alpha (transparency).

`ekFIMAGE` Represents the original format of the image. Only valid at `image_pixels`.

enum codec_t

Image encoding and compression format.

`ekJPG` *Joint Photographic Experts Group.*

`ekPNG` *Portable Network Graphics.*

`ekBMP` *BitMaP.*

`ekGIF` *Graphics Interchange Format.*

enum fstyle_t

Style in typographic fonts. Multiple values can be combined with the OR operator `('|')`.

`ekFNORMAL` Normal font, no style. Also called *Regular*.

`ekFBOLD` **Bold** font.

`ekFITALIC` *Italic* font.

`ekFSTRIKEOUT` ~~Crossed out~~ font.

`ekFUNDERLINE` Underlined font.

`ekFSUBSCRIPT` Subscript. See `textview_fstyle`.

`ekFSUPSCRIPT` Superscript. See `textview_fstyle`

`ekFPIXELS` Font sizes will be indicated in pixels.

`ekFPOINTS` Font sizes will be indicated in points. *"Size in pointsSize in points"* (page 291).

enum linecap_t

Line end style.

ekLCFLAT	Flat termination at the last point of the line.
ekLCSQUARE	Termination in a box, whose center is the last point of the line.
ekLCROUND	Termination in a circle, whose center is the last point of the line.

enum linejoin_t

Line junction style.

ekLJMITER	Union at an angle. In very closed angles it is trimmed.
ekLJROUND	Rounded union.
ekLJBEVEL	Beveled union.

enum fillwrap_t

Behavior of the fill pattern in the limits.

ekFCLAMP	The last limit value is used to fill the outside area.
ekFTILE	Pattern is repeated.
ekFFLIP	The pattern is repeated, reversing the order.

enum drawop_t

Operation to be performed on graphic primitives.

ekSTROKE	Draw the outline of the figure with the default line style.
ekFILL	Fill the figure area with the default color or pattern.
ekSKFILL	First draw the outline and then fill in.
ekFILLSK	First fill in and then draw the outline.

enum align_t

Alignment values.

ekLEFT	Alignment to the left margin.
ekTOP	Alignment to the upper margin.
ekCENTER	Centered alignment.

ekRIGHT	Alignment to the right margin.
ekBOTTOM	Alignment to the lower margin.
ekJUSTIFY	Justification or expansion of content.

enum ellipsis_t

Position of the ellipsis (...) when clipping a text.

ekELLIPNONE	Without ellipsis.
ekELLIPBEGIN	Ellipsis at the beginning of the text.
ekELLIPMIDDLE	Ellipsis in the center of the text.
ekELLIPEND	Ellipsis at the end of the text.
ekELLIPMLINE	Multi-line text (without ellipsis).

struct color_t

32-bit integer representing an RGBA color. The lowest order byte corresponds to the red channel (Red) and the highest order to the Alpha channel (transparency). *"Colors"* (page 277).

```
struct color_t;
```

struct DCtx

2D drawing context, recipient for drawing commands. It is also known as canvas or surface. *"2D Contexts"* (page 257).

```
struct DCtx;
```

struct Draw

Drawing geometric entities.

```
struct Drawf;

struct Drawd;

struct Draw;
```

struct Palette

Color palette, usually related to indexed `Pixbuf`. *"Palettes"* (page 279).

```
struct Palette;
```

struct Pixbuf

In-memory buffer with pixel information. *"Pixel Buffer"* (page 280).

```
struct Pixbuf;
```

struct Image

Represents a bitmap image, composed of pixels. *"Images"* (page 283).

```
struct Image;
```

struct Font

Represents a typographic family, size and style with which the texts will be drawn. *"Typography fonts"* (page 288).

```
struct Font;
```

39.2. Functions

draw2d_start

Start the *draw2d* library, reserving space for global internal structures. Internally call `core_start`. In desktop applications, `osmain` call this function when starting the program.

```
void
draw2d_start(void);
```

draw2d_finish

Ends the *draw2d* library, freeing up the space of the global internal structures. Internally call `core_finish`. In desktop applications, `osmain` call this function when exiting the program.

```
void
draw2d_finish(void);
```

resid_image

Makes a casting ResId-Image.

```
const Image*
resid_image(const ResId id);
```

id The resource Id.

Return:

The Image Id.

dctx_bitmap

Create a memory context, in order to generate an image.

```
DCtx*
dctx_bitmap(const uint32_t width,
            const uint32_t height,
            const pixformat_t format);
```

width Image width in pixels.

height Image height in pixels.

format Pixel format of the generated image.

Return:

Drawing context.

Remarks:

When we finish drawing, we must call `dctx_image` to get the picture.

dctx_image

Get the result image after drawing in the context created with `dctx_bitmap`.

```
Image*
dctx_image(DCtx **ctx);
```

ctx The context, which will be destroyed after generating the image.

Return:

The image.

draw_clear

Clears the entire context area, using a solid color.

```
void
draw_clear(DCtx *ctx,
           const color_t color);
```

ctx Drawing context.

color Background color.

draw_matrix

Set the context reference system (affine transformation).

```
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 Drawing context.

t2d Transformation.

Remarks:

The origin of coordinates is in the upper left corner. The Y axis increases down.

draw_matrix_cartesian

Set the reference system in Cartesian coordinates.

```
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 Drawing context.

t2d Transformation.

Remarks:

The origin of coordinates is in the lower left corner. The Y axis increases upwards. See *"Cartesian systemsCartesian systems"* (page 262).

draw_antialias

Enable or disable antialiasing.

```
void
draw_antialias(DCtx *ctx,
               const bool_t on);
```

ctx Drawing context.

on `TRUE` active, `FALSE` inactive.

Remarks:

The antialias can change in each primitive. It is not necessary to establish a policy for the whole drawing. See *"AntialiasingAntialiasing"* (page 263).

draw_line

Draw a line.

```
void
draw_line(DCtx *ctx,
          const real32_t x0,
          const real32_t y0,
          const real32_t x1,
          const real32_t y1);
```

ctx Drawing context.

x0 X coordinate of the first point.

y0 Y coordinate of the first point.

x1 X coordinate of the second point.

y1 Y coordinate of the second point.

draw_polyline

Draw several joined lines.

```
void
draw_polyline(DCtx *ctx,
              const bool_t closed,
              const V2Df *points,
              const uint32_t n);
```

ctx	Drawing context.
closed	`TRUE` to join the last point with the first.
points	Array of points that compose the polyline.
n	Number of points.

draw_arc

Draw an arc (circle segment).

```
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	Drawing context.
x	X coordinate of the arc center.
y	Y coordinate of the arc center.
radius	Arc radius.
start	Initial angle with respect to the vector `X=[1,0]` in radians.
sweep	Sweep angle or arc size in radians.

Remarks:

Positive angles are those that rotate from vector X to vector Y. See *"2D Vectors"* (page 237).

draw_bezier

Draw a cubic Bézier curve (degree 3) using two endpoints (x0,y0)-(x3,y3) and two intermediate control points (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 Drawing context.

x0 X coordinate of the starting point.

y0 Y coordinate of the starting point.

x1 X coordinate of the first intermediate point.

y1 Y coordinate of the first intermediate point.

x2 X coordinate of the second intermediate point.

y2 Y coordinate of the second intermediate point.

x3 X coordinate of end point.

y3 Y coordinate of the end point.

draw_line_color

Set the color of drawing lines and contours.

```
void
draw_line_color(DCtx *ctx,
                const color_t color);
```

ctx Drawing context.

color Line color.

draw_line_fill

Sets the current fill pattern for line drawing.

```
void
draw_line_fill(DCtx *ctx);
```

ctx Drawing context.

Remarks:

The fill pattern must have been previously set by `draw_fill_linear`. See *"Gradients in linesGradients in lines"* (page 270).

draw_line_width

Set the line thickness.

```
void
draw_line_width(DCtx *ctx,
                const real32_t width);
```

ctx Drawing context.

width Line width.

draw_line_cap

Set the style of the line ends.

```
void
draw_line_cap(DCtx *ctx,
              const linecap_t cap);
```

ctx Drawing context.

cap Style.

draw_line_join

Set the style of line junctions.

```
void
draw_line_join(DCtx *ctx,
               const linejoin_t join);
```

ctx Drawing context.

join Union style.

draw_line_dash

Set a pattern for line drawing.

```
void
draw_line_dash(DCtx *ctx,
               const real32_t *pattern,
               const uint32_t n);
```

ctx Drawing context.

pattern Array of values that define the pattern.

n Number of values.

Remarks:

The first element of `pattern` defines the length of the first stroke and the second of the first hole, so on. Lengths are scaled by line width `draw_line_width`, that is, a stroke of length 1 will draw a square of side *line_width*. Lengths of value 2 equal to twice the line thickness, etc. The pattern will scale proportionally when changing the thickness or zooming through transformations.

draw_rect

Draw a rectangle.

```
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	Drawing context.
op	Drawing operation.
x	X coordinate of the upper left corner of the rectangle.
y	Y coordinate of the upper left corner of the rectangle.
width	Rectangle width.
height	Rectangle height.

Remarks:

In *"Cartesian systemsCartesian systems"* (page 262) (x,y) indicate the origin of the lower left corner.

draw_rndrect

Draw a rectangle with rounded edges.

```
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	Drawing context.
op	Drawing operation.
x	X coordinate of the upper left corner of the rectangle.
y	Y coordinate of the upper left corner of the rectangle.
width	Rectangle width.
height	Rectangle height.
radius	Corner curvature radius.

Remarks:

In *"Cartesian systemsCartesian systems"* (page 262) (x,y) indicate the origin of the lower left corner.

draw_circle

Draw a circle.

```
void
draw_circle(DCtx *ctx,
            const drawop_t op,
            const real32_t x,
            const real32_t y,
            const real32_t radius);
```

ctx	Drawing context.
op	Drawing operation.
x	X coordinate of the center.
y	Y coordinate of the center.
radius	Radius.

draw_ellipse

Draw an ellipse.

```
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	Drawing context.
op	Drawing operation.
x	X coordinate of the center.
y	Y coordinate of the center.
radx	X axis radius.
rady	Y axis radius.

draw_polygon

Draw a polygon.

```
void
draw_polygon(DCtx *ctx,
             const drawop_t op,
             const V2Df *points,
             const uint32_t n);
```

ctx	Drawing context.
op	Drawing operation.
points	Array of points that form the polygon.
n	Number of points.

draw_fill_color

Set a solid color for area filling.

```
void
draw_fill_color(DCtx *ctx,
                const color_t color);
```

ctx	Drawing context.
color	Fill color.

draw_fill_linear

Set a gradient for filling areas.

```
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	Drawing context.
color	Color array.
stop	Color positions.
n	Number of positions/colors.
x0	X coordinate of the starting point.
y0	Y coordinate of the starting point.
x1	X coordinate of the end point.
y1	Y coordinate of the end point.

Remarks:

The positions must go from the value 0 to 1. See *"GradientsGradients"* (page 267).

draw_fill_matrix

Sets the transformation matrix of the fill pattern.

```
void
draw_fill_matrix(DCtx *ctx,
                 const T2Df *t2d);
```

ctx	Drawing context.
t2d	Transformation.

Remarks:

It will only be effective in non-solid fills. See *"GradientsGradients"* (page 267).

draw_fill_wrap

Set the behavior of the gradient or fill pattern to the limits.

```
void
draw_fill_wrap(DCtx *ctx,
               const fillwrap_t wrap);
```

ctx	Drawing context.
wrap	Behavior at the edge.

Remarks:

It will only be effective in non-solid fills. See *"GradientsGradients"* (page 267).

draw_font

Set the font for text drawing.

```
void
draw_font(DCtx *ctx,
          const Font *font);
```

ctx Drawing context.

font Fuente tipográfica.

Remarks:

Tendrá efecto a partir del siguiente texto dibujado. Ver *"Typography fonts"* (page 288).

draw_text_color

Sets the text color.

```
void
draw_text_color(DCtx *ctx,
                const color_t color);
```

ctx Drawing context.

color Color.

draw_text

Draw a block of text.

```
void
draw_text(DCtx *ctx,
          const char_t *text,
          const real32_t x,
          const real32_t y);
```

ctx Drawing context.

text UTF8 string, terminated in a null character `'\0'`.

x X coordinate on the canvas of the text origin.

y Y coordinate on the canvas of the text origin.

Remarks:

The text will be drawn with the font and preset style and will be sensitive to the context transformation. See *"Drawing textDrawing text"* (page 271).

draw_text_path

Draw a block of text as a geometric area. Similar to `draw_text`, but allows you to use gradients or draw only the border of the text.

```
void
draw_text_path(DCtx *ctx,
               const drawop_t op,
               const char_t *text,
               const real32_t x,
               const real32_t y);
```

ctx	Drawing context.
op	Drawing operation.
text	UTF8 string, null-terminated `'\0'`.
x	X coordinate on the canvas of the text origin.
y	Y coordinate on canvas of text origin.

Remarks:

The text will be drawn with the preset font and style (fill and line) and will be context sensitive. See *"Drawing textDrawing text"* (page 271).

draw_text_width

Set the maximum width of the text blocks.

```
void
draw_text_width(DCtx *ctx,
                const real32_t width);
```

ctx	Drawing context.
width	Maximum width.

Remarks:

If the text to draw with `draw_text` is wider than `width`, it will fragment into several lines. Pass -1 to draw the entire block on a single line. Context scaling is not taken into account. The measurement is made based on the size of the preset font. See *"Drawing textDrawing text"* (page 271).

draw_text_trim

Sets how the text will be trimmed when it is wider than the value of `draw_text_width`.

```
void
draw_text_trim(DCtx *ctx,
               const ellipsis_t ellipsis);
```

ctx	Drawing context.
ellipsis	Trim style.

draw_text_align

Sets the alignment of the text with respect to the insertion point.

```
void
draw_text_align(DCtx *ctx,
                const align_t halign,
                const align_t valign);
```

ctx	Drawing context.
halign	Horizontal alignment.
valign	Vertical alignment.

Remarks:

The insertion point is the coordinate `(x,y)` from `draw_text`. See *"Drawing textDrawing text"* (page 271).

draw_text_halign

Set the internal horizontal alignment of the text, within a multi-line block.

```
void
draw_text_halign(DCtx *ctx,
                 const align_t halign);
```

ctx	Drawing context.
halign	Horizontal alignment.

Remarks:

In single-line texts, it has no effect. See *"Drawing textDrawing text"* (page 271).

draw_text_extents

Calculate the size of a block of text.

```
void
draw_text_extents(DCtx *ctx,
                  const char_t *text,
                  const real32_t refwidth,
                  real32_t *width,
                  real32_t *height);
```

ctx	Drawing context.
text	Text.
refwidth	Reference width.
width	Block width.
height	Block height.

Remarks:

If `refwidth` is greater than `0`, `width` will be bounded by this value and `height` will expand to accommodate all the text. Take into account possible new lines `'\n'` from `text`.

draw_image

Draw a image.

```
void
draw_image(DCtx *ctx,
           const real32_t x,
           const real32_t y);
```

ctx	Drawing context.
x	X coordinate on the canvas of the image origin.
y	Y coordinate on the canvas of the image origin.

Remarks:

The image will be drawn at its natural size and in the indicated position. Use `draw_matrixf` to perform scaling and rotation. See *"Drawing imagesDrawing images"* (page 274).

draw_image_frame

Like `draw_image`, but indicating the sequence number of an animation.

```
void
draw_image_frame(DCtx *ctx,
                 const uint32_t frame,
                 const real32_t x,
                 const real32_t y);
```

ctx	Drawing context.
frame	Sequence index *(frame)* of the animation.
x	X coordinate on the canvas of the image origin.
y	Y coordinate on the canvas of the image origin.

Remarks:

Only images created from a **GIF** file support multiple frames (animations). See `image_num_frames`.

draw_image_align

Sets the alignment of the image with respect to the insertion point.

```
void
draw_image_align(DCtx *ctx,
                 const align_t halign,
                 const align_t valign);
```

ctx	Drawing context.
halign	Horizontal alignment.
valign	Vertical alignment.

Remarks:

The insertion point is the coordinate `(x,y)` from `draw_image`. See *"Drawing images-Drawing images"* (page 274).

draw_v2d

Draw a 2D point.

```
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 Drawing context.

op Drawing operation.

v2d Point.

radius Radius.

draw_seg2d

Draw a 2D segment.

```
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 Drawing context.

seg Segment.

draw_cir2d

Draw a 2D circle.

```
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 Drawing context.

op Drawing operation.

cir Circle.

draw_box2d

Draw a 2D box.

```
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 Drawing context.

op Drawing operation.

box Aligned box.

draw_obb2d

Draw an oriented 2D box.

```
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 Drawing context.

op Drawing operation.

obb Oriented box.

draw_tri2d

Draw a 2D triangle.

```
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 Drawing context.

op Drawing operation.

tri Triangle.

draw_pol2d

Draw a 2D polygon.

```
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 Drawing context.

op Drawing operation.

pol Polygon.

color_rgb

Create a color from the channels **R** (red), **G** (green) y **B** (blue).

```
color_t
color_rgb(const uint8_t r,
          const uint8_t g,
          const uint8_t b);
```

r Red channel.

g Green channel.

b Blue channel.

Return:

Color.

Remarks:

The alpha channel is set to 255 (totally opaque).

color_rgba

Create a color from the channels **R** (red), **G** (green), **B** (blue) and **A** (alpha).

```
color_t
color_rgba(const uint8_t r,
           const uint8_t g,
           const uint8_t b,
           const uint8_t a);
```

r Red channel.

g Green channel.

b Blue channel.

a Alpha channel (transparency).

Return:

Color.

Remarks:

`a=0` not supported. Use `kCOLOR_TRANSPARENT` in those cases.

color_rgbaf

Create a color from the normalized RGBA channels from 0 to 1.

```
color_t
color_rgbaf(const real32_t r,
            const real32_t g,
            const real32_t b,
            const real32_t a);
```

r Red channel.

g Green channel.

b Blue channel.

a Alpha channel (transparency).

Return:

Color.

Remarks:

`a=0` not supported. Use `kCOLOR_TRANSPARENT` in those cases.

color_hsbf

Creates a color (rgb) from its components Hue-Saturation-Brightness.

```
color_t
color_hsbf(const real32_t hue,
           const real32_t sat,
           const real32_t bright);
```

hue	Hue component.
sat	Saturation component.
bright	Brightness component.

Return:

Color.

color_red

Create an RGB color using only the red channel.

```
color_t
color_red(const uint8_t r);
```

r	Red Channel.

Return:

Color.

Remarks:

Equivalent to `color_rgb(r, 0, 0)`.

color_green

Create an RGB color using only the green channel.

```
color_t
color_green(const uint8_t g);
```

g	Green channel.

Return:

Color.

Remarks:

Equivalent to `color_rgb(0, g, 0)`.

color_blue

Create an RGB color using only the blue channel.

```
color_t
color_blue(const uint8_t b);
```

b Blue channel.

Return:

Color.

Remarks:

Equivalent to `color_rgb(0, 0, b)`.

color_gray

Creates a gray RGB color from intensity value.

```
color_t
color_gray(const uint8_t l);
```

l Intensity (luminance).

Return:

Color.

Remarks:

Equivalent to `color_rgb(l, l, l)`.

color_bgr

Create a color from a 32-bit BGR value. Byte 0 corresponds to channel **B** , 1 to **G** and 2 to **R.** The highest order byte is ignored (set to 255).

```
color_t
color_bgr(const uint32_t bgr);
```

bgr The bgr 32bits value.

Return:

Color.

Remarks:

This byte order is typical in Web colors.

color_html

Create a color from a string in HTML or CSS format.

```
color_t
color_html(const char_t *html);
```

```
color_t c1 = color_html("#FF0000"); // Red
color_t c2 = color_html("#000080"); // Navy
```

html The text string with the HTML color.

Return:

The color transformed to RGB.

color_to_hsbf

Convert a color (rgb) to HSB space (hue, saturation, brightness).

```
void
color_to_hsbf(const color_t color,
              real32_t *hue,
              real32_t *sat,
              real32_t *sat);
```

color Color.

hue Hue component.

sat Saturation component.

sat Brightness component.

color_to_html

Convert a color to the HTML or CSS format (#RRGGBB).

```
void
color_to_html(const color_t color,
              char_t *html,
              const uint32_t size);
```

color The color to convert.

html Buffer where to write the result.

size Result buffer size.

color_get_rgb

Returns RGB color values.

```
void
color_get_rgb(const color_t color,
              uint8_t *r,
              uint8_t *g,
              uint8_t *b);
```

color Color.

r Red channel.

g Green channel.

b Blue channel.

Remarks:

In system or indexed colors, it makes effective the RGB value.

color_get_rgbf

Returns RGB color values, normalized from 0 to 1.

```
void
color_get_rgbf(const color_t color,
               real32_t *r,
               real32_t *g,
               real32_t *b);
```

color Color.

r Red channel.

g Green channel.

b Blue channel.

Remarks:

In system or indexed colors, it makes effective the RGB value.

color_get_rgba

Returns the RGBA values of the color.

```
void
color_get_rgba(const color_t color,
               uint8_t *r,
               uint8_t *g,
```

```
                uint8_t *b,
                uint8_t *a);
```

color Color.

r Red channel.

g Green channel.

b Blue channel.

a Alpha channel (transparency).

Remarks:

In system or indexed colors, it makes effective the RGBA value.

color_get_rgbaf

Returns the RGBA values of the color, normalized from 0 to 1.

```
void
color_get_rgbaf(const color_t color,
                real32_t *r,
                real32_t *g,
                real32_t *b,
                real32_t *a);
```

color Color.

r Red channel.

g Green channel.

b Blue channel.

a Alpha channel (transparency).

Remarks:

In system or indexed colors, it makes effective the RGBA value.

color_get_alpha

Get the alpha (transparency) color component.

```
uint8_t
color_get_alpha(const color_t color);
```

color Color.

Return:

The alpha component. If it is equal 0 it means that the color is indexed (does not contain RGB values).

color_set_alpha

Changes the alpha (transparency) value of a color.

```
color_t
color_set_alpha(const color_t color,
                const uint8_t alpha);
```

color Color.

alpha Alpha component.

Return:

The new color, with the altered alpha component.

palette_create

Create a palette.

```
Palette*
palette_create(const uint32_t size);
```

size The number of colors.

Return:

The palette. The initial content is undetermined. Edit with `palette_colors`.

palette_cga2

Create the 4-color (2-bit) palette of CGA cards.

```
Palette*
palette_cga2(const bool_t mode,
             const bool_t intense);
```

mode `TRUE` for CGA mode 1, `FALSE` mode 0.

intense `TRUE` for bright colors.

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_ega4

Create the default palette for EGA cards (16 colors, 4 bits).

```
Palette*
palette_ega4(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_rgb8

Create the default 8-bit RGB palette. Colors combine 8 tones of red, 8 green and 4 blue.

```
Palette*
palette_rgb8(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_gray1

Create a palette of 2 tones of gray (1 bit). Black (0) and white (1).

```
Palette*
palette_gray1(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_gray2

Create a palette of 4 tones of gray (2 bit). Black (0), White (3).

```
Palette*
palette_gray2(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_gray4

Create a palette of 16 tones of gray (4 bit). Black (0), White (15).

```
Palette*
palette_gray4(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_gray8

Create a palette of 256 shades of gray (8 bit). Black (0), White (255).

```
Palette*
palette_gray8(void);
```

Return:

The palette.

Remarks:

"Predefined palettePredefined palette" (page 280)

palette_binary

Create a two-color palette.

```
Palette*
palette_binary(const color_t zero,
               const color_t one);
```

zero Color associated with the `0` value.

one Color associated with the `1` value.

Return:

The palette.

palette_destroy

Destroy the palette.

```
void
palette_destroy(Palette **palette);
```

palette The palette. It will be set to `NULL` after the destruction.

palette_size

Returns the number of colors in the palette.

```
uint32_t
palette_size(const Palette *palette);
```

palette The palette.

Return:

The number of colors.

palette_colors

Get the color list.

```
color_t*
palette_colors(Palette *palette);
```

palette The palette.

Return:

Colors. The size of the array is given by `palette_size`.

Remarks:

The buffer is read/write.

palette_colors_const

Get the color list.

```
const color_t*
palette_colors_const(const Palette *palette);
```

palette The palette.

Return:

Colors. The size of the array is given by `palette_size`.

pixbuf_create

Create a new pixel buffer.

```
Pixbuf*
pixbuf_create(const uint32_t width,
              const uint32_t height,
              const pixformat_t format);
```

width Width.

height Height.

format Pixel format.

Return:

The buffer pixel.

Remarks:

Initial content will be undefined.

pixbuf_copy

Create a copy of the pixel buffer.

```
Pixbuf*
pixbuf_copy(const Pixbuf *pixbuf);
```

pixbuf The original buffer.

Return:

The copy.

pixbuf_trim

Crop a buffer pixel.

```
Pixbuf*
pixbuf_trim(const Pixbuf *pixbuf,
            const uint32_t x,
            const uint32_t y,
            const uint32_t width,
            const uint32_t height);
```

pixbuf The original buffer.

x X coordinate of the upper-left pixel.

y Y coordinate of the upper-left pixel.

width Number of pixels wide.

height Number of pixels high.

Return:

A new buffer pixel with clipping.

Remarks:

The function does not check that the limits are valid. You will get a segmentation error in such cases.

pixbuf_convert

Change the format of a buffer pixel.

```
Pixbuf*
pixbuf_convert(const Pixbuf *pixbuf,
               const Palette *palette,
               const pixformat_t oformat);
```

pixbuf The original buffer.

palette Color palette required for certain conversions.

oformat Result buffer format.

Return:

The converted buffer.

Remarks:

See *"Copy and conversionCopy and conversion"* (page 283).

pixbuf_destroy

Destroy the buffer.

```
void
pixbuf_destroy(Pixbuf **pixbuf);
```

pixbuf The buffer. It will be set to `NULL` after the destruction.

pixbuf_format

Get the pixel format.

```
pixformat_t
pixbuf_format(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

The format.

Remarks:

See *"Pixel formatsPixel formats"* (page 281).

pixbuf_width

Get the width of the buffer.

```
uint32_t
pixbuf_width(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Width.

pixbuf_height

Get the height of the buffer.

```
uint32_t
pixbuf_height(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Height.

pixbuf_size

Get the buffer size (in pixels).

```
uint32_t
pixbuf_size(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Width x height.

pixbuf_dsize

Gets the buffer size (in bytes).

```
uint32_t
pixbuf_dsize(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Number of total bytes in the buffer.

pixbuf_cdata

Gets a read-only pointer to the contents of the buffer.

```
const byte_t*
pixbuf_cdata(const Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Pointer to the first element.

Remarks:

Correctly manipulating the buffer requires knowing the *"Pixel formatsPixel formats"* (page 281) and sometimes using the operators at the bit level. Use `pixbuf_get` to correctly read a pixel.

pixbuf_data

Gets a read/write pointer to the contents of the buffer.

```
byte_t*
pixbuf_data(Pixbuf *pixbuf);
```

pixbuf The buffer.

Return:

Pointer to the first element.

Remarks:

Correctly manipulating the buffer requires knowing the *"Pixel formatsPixel formats"* (page 281) and sometimes using the operators at the bit level. Use `pixbuf_get` to correctly read a pixel.

pixbuf_format_bpp

Gets bits per pixel based on format.

```
uint32_t
pixbuf_format_bpp(const pixformat_t format);
```

format The format.

Return:

Bits per pixel.

Remarks:

See *"Pixel formatsPixel formats"* (page 281).

pixbuf_get

Get the value of a pixel.

```
uint32_t
pixbuf_get(const Pixbuf *pixbuf,
           const uint32_t x,
           const uint32_t y);
```

pixbuf The buffer.

x Pixel x-coordinate.

y Pixel y coordinate.

Return:

The color value.

Remarks:

See *"Pixel formatsPixel formats"* (page 281) to correctly interpret the value.

pixbuf_set

Sets the value of a pixel.

```
void
pixbuf_set(Pixbuf *pixbuf,
           const uint32_t x,
           const uint32_t y,
           const uint32_t value);
```

pixbuf The buffer.

x Pixel x-coordinate.

y Pixel y coordinate.

value The color value.

Remarks:

See *"Pixel formatsPixel formats"* (page 281) to correctly interpret the value.

image_from_pixels

Create an image from an array of pixels.

```
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	The image width (in pixels).
height	The image height (in pixels).
format	Pixel format.
data	Buffer that contains the color value of each pixel. It will depend on the resolution and format.
palette	Color palette required to render indexed images. If it is `NULL` a *"Predefined palettePredefined palette"* (page 280) will be used if necessary.
palsize	Number of colors in the palette.

Return:

The image.

Remarks:

See *"Pixel accessPixel access"* (page 285).

image_from_pixbuf

Create an image from a buffer pixel.

```
Image*
image_from_pixbuf(const Pixbuf *pixbuf,
                  const Palette *palette);
```

pixbuf	The buffer.
palette	The palette.

Return:

The image.

Remarks:

Equal to `image_from_pixels` avoiding indicating parameters separately.

image_from_file

Create an image from a file on disk.

```
Image*
image_from_file(const char_t *pathname,
                ferror_t *error);
```

pathname The file path. *"Filename and pathnameFilename and pathname" (page 178).*

error Error code if the function fails. Can be `NULL`.

Return:

The image.

Remarks:

Only formats *jpg*, *png*, *bmp* and *gif* are accepted.

image_from_data

Create an image from a buffer containing the encoded data.

```
Image*
image_from_data(const byte_t *data,
                const uint32_t size);
```

data The buffer with the image data.

size The buffer size in bytes.

Return:

The image.

Remarks:

The buffer represents data encoded in *jpg*, *png*, *bmp* or *gif*. To create the image directly from pixels use `image_from_pixels`.

image_from_resource

Get an image of a resource package.

```
const Image*
image_from_resource(const ResPack *pack,
                    const ResId id);
```

pack The resource package.

id The resource identifier.

Return:

The image.

Remarks:

The image should not be destroyed with `image_destroy` as it is part of the package itself (it is constant). Make a copy with `image_copy` in case it needs to be kept after destroying the resources. See *"Resources"* (page 129).

image_copy

Create a copy of the image.

```
Image*
image_copy(const Image *image);
```

image The source image.

Return:

The image copy.

Remarks:

Images are immutable objects. Copying really means increasing an internal counter without cloning the object. However, the application must destroy the copy with `image_destroy` just like those created with any other constructor. When all copies are destroyed, it will actually be removed from memory.

image_trim

Create an image by cropping another image.

```
Image*
image_trim(const uint32_t x,
           const uint32_t y,
           const uint32_t width,
           const uint32_t height);
```

x	X coordinate of the origin of the sub-image.
y	Y coordinate of the origin of the sub-image.
width	Width in pixels of the sub-image.
height	Height in pixels of the sub-image.

Return:

The new image.

image_rotate

Create a new image by rotating an existing one.

```
Image*
image_rotate(const Image *image,
             const real32_t angle,
             const bool_t nsize,
             const color_t background,
             T2Df *t2d);
```

image	The original image.
angle	Angle in radians.
nsize	`TRUE` the resulting image will be resized to fit the entire original. `FALSE` the resulting image will have the same dimensions as the original, cutting part of the content (clipping).
background	Background color. The new image will have "blank" areas due to rotation.
t2d	Saves the transformation applied to the image. They can be `NULL` if we don't need this value.

Return:

The newly created image.

image_scale

Create a copy of the image, with a new size.

```
Image*
image_scale(const Image *image,
            const uint32_t nwidth,
            const uint32_t nheight);
```

image	The source image.
nwidth	The width of the new image. Pass `UINT32_MAX` so that the aspect ratio with respect to `nheight`.
nheight	The height of the new image. Pass `UINT32_MAX` so that the aspect ratio with respect to `nwidth`.

Return:

The image.

Remarks:

If both values `nwidth`, `nheight` are `UINT32_MAX` or the new dimensions are identical to the current ones, the internal reference counter will increase, as is the case in `image_copy`.

image_read

Create an image from the data read from a *"Streams"* (page 193).

```
Image*
image_read(Stream *stm);
```

stm	Input stream. Data encoded in *jpg*, *png*, *bmp* or *gif* are expected. The function detects the format automatically.

Return:

The image.

image_to_file

Save an image to disk, using the codec associated with it.

```
bool_t
image_to_file(const Image *image,
              const char_t *pathname,
              ferror_t *error);
```

image	The image.
pathname	The path of the destination file. *"Filename and pathnameFilename and pathname"* (page 178).
error	Error code if the function fails. Can be `NULL`.

Return:

`TRUE` if it was saved correctly or `FALSE` and an error has occurred.

Remarks:

Use `image_codec` to change the default codec.

image_write

Write an image in an output stream, using the codec associated with it.

```
void
image_write(Stream *stm,
            const Image *image);
```

stm Writing stream. Data encoded in *jpg*, *png*, *bmp* or *gif* will be written.

image The image.

Remarks:

Use `image_codec` to change the default codec.

image_destroy

Destroy the image.

```
void
image_destroy(Image **image);
```

image The image. Will be set to `NULL` after destruction.

image_format

Get the pixel format of the image.

```
pixformat_t
image_format(const Image *image);
```

image The image.

Return:

Pixel format.

image_width

Get the width of the image in pixels.

```
uint32_t
image_width(const Image *image);
```

image The image.

Return:

Number of pixels wide.

image_height

Get the height of the image in pixels.

```
uint32_t
image_height(const Image *image);
```

image The image.

Return:

Number of pixels in height.

image_pixels

Get a buffer with the pixels that make up the decoded image.

```
Pixbuf*
image_pixels(const Image *image,
             const pixformat_t format);
```

image The image.

format The required pixel format.

Return:

Pixel buffer with image content.

Remarks:

If in `pixformat` we indicate `ekFIMAGE` it will return the buffer with the original format of the image. We can indicate `ekRGB24`, `ekRGBA32` or `ekGRAY8` if we need a specific format. Cannot use indexed formats.

image_codec

Change the default codec associated with the image.

```
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 The image.

codec The new codec.

Return:

TRUE if the graphical API supports the selected codec. FALSE otherwise.

Remarks:

The change will take effect the next time we save or write the image. By default, the image retains the codec with which it was read. When we create it with image_from_pixels ekJPG codec is assigned as default. For images from 2d contexts dctx_image, the default codec is ekPNG. All codecs are supported by all graphical APIs, except ekGIF in some versions of Linux. Check the return value if it is imperative that your application export images in GIF.

image_get_codec

Get the codec associated with the image.

```
codec_t
image_get_codec(const Image *image);
```

image The image.

Return:

El codec.

Remarks:

See image_codec.

image_num_frames

Get the number of sequences in animated images.

```
uint32_t
image_num_frames(const Image *image);
```

image The image.

Return:

The number of sequences or *frames*.

Remarks:

Only the *gif* format supports animations. For the rest 1 will always be returned.

image_frame_length

Get the time of an animation sequence.

```
real32_t
image_frame_length(const Image *image,
                   const uint32_t findex);
```

image	The image.
findex	The frame index.

Return:

Sequence time in seconds.

Remarks:

Only *gif* format supports animations.

image_data

Link user data with the image.

```
void
image_data(Image *image,
           type *data,
           FPtr_destroy func_destroy_data,
           type );
```

image	The image.
data	The user data.
func_destroy_data	Destructor of user data.
	User data type.

image_get_data

Gets the user data of the image.

```
type*
image_get_data(const Image *image,
               type );
```

image The image.

User data type.

Return:

The user data.

image_native

Gets the image in the native format of each platform.

```
void*
image_native(const Image *image);
```

image The image.

Return:

The native image. `Gdiplus::Bitmap` in Windows, `GdkPixbuf` in Linux and `NSImage` in macOS.

font_create

Create a font.

```
Font*
font_create(const char_t *family,
            const real32_t size,
            const uint32_t style);
```

family Typographic family. Eg: “Arial”, “Times New Roman”, etc.

size Font size. Default in pixels. Use `ekFPOINTS` in `style` to change the unit.

style Operation `OR` `|` over the fields of the `fstyle_t` structure. Eg: `ekFBOLD | ekFITALIC`.

Return:

The font.

font_system

Create a font, with the system default family.

```
Font*
font_system(const real32_t size,
            const uint32_t style);
```

size Font size. Default in pixels. Use `ekFPOINTS` in `style` to change the unit.

style Operation `OR` `|` over the fields of the `fstyle_t` structure. Eg: `ekFBOLD | ekFITALIC`.

Return:

The font.

font_monospace

Create a font, with the default system mono-space family.

```
Font*
font_monospace(const real32_t size,
               const uint32_t style);
```

size Font size. Default in pixels. Use `ekFPOINTS` in `style` to change the unit.

style Operation `OR` `|` over the fields of the `fstyle_t` structure. Eg: `ekFBOLD | ekFITALIC`.

Return:

The font.

font_with_style

Create a copy of an existing font, changing the style.

```
Font*
font_with_style(const Font *font,
                const uint32_t style);
```

font Original font.

style Operation `OR` `|` over the fields of the `fstyle_t` structure. Eg: `ekFBOLD | ekFITALIC`.

Return:

A copy of `font` with another style.

font_copy

Create an exact copy of a font.

```
Font*
font_copy(const Font *font);
```

font Source font.

Return:

The copy of `font`.

Remarks:

Fonts are immutable objects. Copying really means increasing an internal counter without cloning the object. However, the application must destroy the copy with `font_destroy` just like those created with any other constructor.

font_destroy

Destroy the font.

```
void
font_destroy(Font **font);
```

font The font. Will be set to `NULL` after destruction.

font_equals

Compare two fonts. They are considered equal if they have the same family, size and style.

```
bool_t
font_equals(const Font *font1,
            const Font *font2);
```

font1 First font to compare.

font2 Second font to compare.

Return:

`TRUE` if they are the same, `FALSE` if not.

font_regular_size

Get the default font size for interface controls.

```
real32_t
font_regular_size(void);
```

Return:

The default size in pixels.

font_small_size

Get the *small* font size by default for interface controls.

```
real32_t
font_small_size(void);
```

Return:

The size in pixels.

Remarks:

This size is slightly smaller than that obtained by `font_regular_size`.

font_mini_size

Get the default *mini* font size for interface controls.

```
real32_t
font_mini_size(void);
```

Return:

The size in pixels.

Remarks:

This size is slightly smaller than that obtained by `font_small_size`.

font_family

Get the font type family.

```
const char_t*
font_family(const Font *font);
```

font The font.

Return:

The typographic family in UTF8.

font_size

Get the font size.

```
real32_t
font_size(const Font *font);
```

font The font.

Return:

The size. The units depend on the parameter `style`.

font_height

Get the height of the cell or line of text with this font.

```
real32_t
font_height(const Font *font);
```

font The font.

Return:

Cell height.

font_style

Get the font style.

```
uint32_t
font_style(const Font *font);
```

font The font.

Return:

The style. Combination of `fstyle_t` structure values. Eg: `ekFBOLD | ekFITALIC`.

font_extents

Gets the size in pixels of a text string, based on the font.

```
void
font_extents(const Font *font,
             const char_t *text,
             const real32_t refwidth,
             real32_t *width,
             real32_t *height);
```

font	The font.
text	The text string to size.
refwidth	Maximum width of the text box.
width	Text box width.
height	Text box height.

font_exists_family

Check if a typeface family is installed in the operating system.

```
bool_t
font_exists_family(const char_t *family);
```

family	UTF8 string with family name, terminated in a null character `'\0'`.

Return:

`TRUE` if the family exists, `FALSE` if not.

font_installed_families

Get a list with the names of all the typographic families installed in the operating system.

```
ArrPt(String)*
font_installed_families(void);
```

```
ArrPt(String) *families = font_installed_families();
...
arrpt_destroy(&families, str_destroy, String);
```

Return:

Array of `String` with the names of the families, arranged alphabetically. It must be destroyed with `arrpt_destroy`.

font_native

Gets the font in the native format of each platform.

```
void*
font_native(const Font *font);
```

font The font.

Return:

The native font. `HFONT` in Windows, `PangoFontDescription` in Linux and `NSFont` in macOS.

Chapter 40

Gui library

40.1. Types and Constants

enum gui_orient_t

Orientation.

ekGUI_HORIZONTAL	Horizontal.
ekGUI_VERTICAL	Vertical.

enum gui_state_t

State values.

ekGUI_OFF	Off.
ekGUI_ON	On.
ekGUI_MIXED	Medium/undetermined.

enum gui_mouse_t

Mouse buttons.

ekGUI_MOUSE_LEFT	Left.
ekGUI_MOUSE_RIGHT	Right.
ekGUI_MOUSE_MIDDLE	Center.

enum gui_cursor_t

Cursors. See `window_cursor`.

ekGUI_CURSOR_ARROW	Arrow (default).
ekGUI_CURSOR_HAND	Hand.
ekGUI_CURSOR_IBEAM	Vertical bar (text editing).
ekGUI_CURSOR_CROSS	Cross.
ekGUI_CURSOR_SIZEWE	Horizontal resize (left-right).
ekGUI_CURSOR_SIZENS	Vertical resize (top-bottom).
ekGUI_CURSOR_USER	Created from an image.

enum gui_close_t

Reason for closing a window.

ekGUI_CLOSE_ESC	The `[ESC]` key has been pressed (cancel).
ekGUI_CLOSE_INTRO	The `[ENTER]` key has been pressed (accept).
ekGUI_CLOSE_BUTTON	The close button `[X]` has been pressed in the title bar.
ekGUI_CLOSE_DEACT	The parent window has been hidden.

enum gui_scale_t

Scaling modes.

ekGUI_SCALE_AUTO	Automatic scaling, the proportion may change.
ekGUI_SCALE_NONE	No scaling.
ekGUI_SCALE_ASPECT	Automatic scaling, but maintaining the proportion *(aspect ratio)*.
ekGUI_SCALE_ASPECTDW	Same as above, but does not increase the original size, only reduce it if appropriate.

enum gui_scroll_t

Types of scroll.

ekGUI_SCROLL_LEFT	Pressed the `left` or `up` button of a scroll bar.
ekGUI_SCROLL_RIGHT	Pressed the `right` or `down` button on a scroll bar.
ekGUI_SCROLL_WHEEL	Mouse wheel.

ekGUI_SCROLL_THUMB	Drag the thumb control of a scroll bar.

enum gui_focus_t

Result when changing the keyboard focus.

ekGUI_FOCUS_CHANGED	The focus has changed to the specified control.
ekGUI_FOCUS_KEEP	The focus has not moved, it remains on the same control.
ekGUI_FOCUS_NO_NEXT	Target control not found, hidden, or disabled.
ekGUI_FOCUS_NO_RESIGN	The current control does not allow changing focus.
ekGUI_FOCUS_NO_ACCEPT	The new control does not accept focus.

enum gui_event_t

Event type. See *"GUI EventsGUI Events"* (page 300).

ekGUI_EVENT_LABEL	Click on a Label control.
ekGUI_EVENT_BUTTON	Click on a Button control.
ekGUI_EVENT_POPUP	The selection of a PopUp control has been changed.
ekGUI_EVENT_LISTBOX	The selection of a control has been changed ListBox.
ekGUI_EVENT_SLIDER	You are moving an Slidercontrol.
ekGUI_EVENT_UPDOWN	Click on a UpDown control.
ekGUI_EVENT_TXTFILTER	The text of a Edit or Combo control is being edited.
ekGUI_EVENT_TXTCHANGE	You have finished editing the text of a Edit or Combo control.
ekGUI_EVENT_FOCUS_RESIGN	Ask a control if it wants to give up keyboard focus.
ekGUI_EVENT_FOCUS_ACCEPT	Ask a control if it wants to accept keyboard focus.
ekGUI_EVENT_FOCUS	A control has received or lost keyboard focus.
ekGUI_EVENT_MENU	Click on a menu.
ekGUI_EVENT_DRAW	The view content must be redrawn.
ekGUI_EVENT_RESIZE	The size of a view has changed.
ekGUI_EVENT_ENTER	The mouse has entered the view area.
ekGUI_EVENT_EXIT	The mouse has left the view area.

ekGUI_EVENT_MOVED	The mouse is moving on the view surface.
ekGUI_EVENT_DOWN	A mouse button was pressed.
ekGUI_EVENT_UP	A mouse button has been released.
ekGUI_EVENT_CLICK	Click on a view.
ekGUI_EVENT_DRAG	*Dragging* is being done over.
ekGUI_EVENT_WHEEL	Mouse wheel has moved.
ekGUI_EVENT_KEYDOWN	A key has been pressed.
ekGUI_EVENT_KEYUP	A key has been released.
ekGUI_EVENT_SCROLL	The scroll bars are being manipulated.
ekGUI_EVENT_WND_MOVED	The window is moving across the desktop.
ekGUI_EVENT_WND_SIZING	The window is being resized.
ekGUI_EVENT_WND_SIZE	The window has been resized.
ekGUI_EVENT_WND_CLOSE	The window has been closed.
ekGUI_EVENT_COLOR	An update color of `comwin_color`.
ekGUI_EVENT_THEME	Desktop theme has changed.
ekGUI_EVENT_OBJCHANGE	An object linked to a layout has been edited. *"Notifications and calculated fieldsNotifications and calculated fields"* (page 361).
ekGUI_EVENT_TBL_NROWS	A table needs to know the number of rows. `tableview_OnData`.
ekGUI_EVENT_TBL_BEGIN	A table will begin to draw the visible part of the data. `tableview_OnData`.
ekGUI_EVENT_TBL_END	A table has finished drawing. `tableview_OnData`.
ekGUI_EVENT_TBL_CELL	A table needs the data of a cell. `tableview_OnData`.
ekGUI_EVENT_TBL_SEL	The selected rows in a table have changed. `tableview_OnSelect`.
ekGUI_EVENT_TBL_HEADCLICK	Click on a table header. `tableview_OnHeaderClick`.

enum window_flag_t

Window creation attributes.

ekWINDOW_FLAG	Default attributes.
ekWINDOW_EDGE	The window draws an outer border.
ekWINDOW_TITLE	The window has a title bar.
ekWINDOW_MAX	The window shows the maximize button.
ekWINDOW_MIN	The window shows the minimize button.
ekWINDOW_CLOSE	The window shows the close button.
ekWINDOW_RESIZE	The window has resizable borders.
ekWINDOW_RETURN	The window will process the pressing of the [RETURN] key as a possible closing event, sending the message OnClose.
ekWINDOW_ESC	The window will process the pressing of the [ESC] key as a possible closing event, sending the message OnClose.
ekWINDOW_MODAL_NOHIDE	Avoids hiding a modal window when the modal cycle has finished. See *"Modal windowsModal windows"* (page 350).
ekWINDOW_STD	Combination ekWINDOW_TITLE | ekWINDOW_MIN | ekWINDOW_CLOSE.
ekWINDOW_STDRES	Combination ekWINDOW_STD | ekWINDOW_MAX | ekWINDOW_RESIZE.

enum gui_notif_t

Notifications sent by the `gui` library.

ekGUI_NOTIF_LANGUAGE	The interface language has been changed.
ekGUI_NOTIF_WIN_DESTROY	A window has been destroyed.
ekGUI_NOTIF_MENU_DESTROY	A menu has been destroyed.

struct GuiControl

Interface Control (abstract).

```
struct GuiControl;
```

struct Label

Interface control that contains static text, usually limited to a single line. *"Label"* (page 304).

```
struct Label;
```

struct Button

Interface control representing a button. *"Button"* (page 306).

```
struct Button;
```

struct PopUp

Control button with drop-down list. *"PopUp"* (page 308).

```
struct PopUp;
```

struct Edit

Text editing control *"Edit"* (page 309).

```
struct Edit;
```

struct Combo

Control that combines an edit box with a drop-down list. *"Combo"* (page 312).

```
struct Combo;
```

struct ListBox

List control. *"ListBox"* (page 312).

```
struct ListBox;
```

struct UpDown

Control that shows two small increase and decrease buttons. *"UpDown"* (page 314).

```
struct UpDown;
```

struct Slider

Control that shows a bar with a slider. *"Slider"* (page 314).

```
struct Slider;
```

struct Progress

Progress bar. *"Progress"* (page 315).

```
struct Progress;
```

struct View

Custom View that allows to create our own controls, drawing what we want. *"View"* (page 315)

```
struct View;
```

struct TextView

Text view with several paragraphs and different attributes. *"TextView"* (page 321).

```
struct TextView;
```

struct ImageView

Image viewer control. *"ImageView"* (page 322).

```
struct ImageView;
```

struct TableView

Table view with multiple rows and columns. *"TableView"* (page 323).

```
struct TableView;
```

struct SplitView

Resizable horizontal or vertical split view. *"SplitView"* (page 329).

```
struct SplitView;
```

struct Layout

Invisible grid where the controls of a `Panel` are organized. *"Layout"* (page 332).

```
struct Layout;
```

struct Cell

Each of the cells that form a `Layout`. *"Cell"* (page 340).

```
struct Cell;
```

struct Panel

Internal area of a window, which allows you to group different controls. *"Panel"* (page 341).

```
struct Panel;
```

struct Window

Interface window. *"Window"* (page 347).

```
struct Window;
```

struct Menu

Menu or submenu. *"Menu"* (page 363).

```
struct Menu;
```

struct MenuItem

Item within a menu. *"MenuItem"* (page 363).

```
struct MenuItem;
```

struct EvButton

Parameters of the *OnClick* event of a button or *OnSelect* of a popup.

```
struct EvButton
{
    uint32_t index;
    gui_state_t state;
    const char_t* text;
};
```

`index` Button or item index.

`state` State.

`text` Text.

struct EvSlider

Parameters of the *OnMoved* event of a slider.

```
struct EvSlider
{
    real32_t pos;
    real32_t incr;
    uint32_t step;
};
```

`pos` Normalized slider position (0, 1).

`incr` Increase with respect to the previous position.

`step` Interval index (only for discrete ranges).

struct EvText

Parameters of the *OnChange* event of the text boxes.

```
struct EvText
{
    const char_t* text;
    uint32_t cpos;
    int32_t len;
    GuiControl* next;
};
```

`text` Text.

`cpos` Cursor position *(caret)*.

`len` Number of characters inserted or deleted.

`next` Next control to which keyboard focus will be moved.

struct EvTextFilter

Result of the *OnFilter* event of the text boxes.

```
struct EvTextFilter
{
    bool_t apply;
    char_t* text;
    uint32_t cpos;
};
```

`apply` `TRUE` if the original control text should be changed.

`text` New control text, which is a revision (filter) of the original text.

`cpos` Cursor position *(caret)*.

struct EvDraw

OnDraw event parameters.

```
struct EvDraw
{
    DCtx* ctx;
    real32_t x;
    real32_t y;
    real32_t width;
    real32_t height;
};
```

`ctx` 2D drawing context.

`x` X coordinate of the drawing area (viewport).

`y` Y coordinate of the drawing area.

`width` Width of the drawing area.

`height` Height of the drawing area.

struct EvMouse

Mouse event parameters.

```
struct EvMouse
{
    real32_t x;
    real32_t y;
    real32_t lx;
    real32_t ly;
    gui_mouse_t button;
    uint32_t count;
};
```

`x` X coordinate of the pointer in the drawing area.

`y` Y coordinate of the pointer in the drawing area.

`lx` X coordinate of the pointer on the control. Same as `x` if there are no scroll bars.

`ly` Y coordinate of the pointer on the control. Same as `y` if there are no scroll bars.

`button` Active button.

`count` Number of clicks.

struct EvWheel

OnWheel event *parameters*.

```
struct EvWheel
{
    real32_t x;
    real32_t y;
    real32_t dx;
    real32_t dy;
    real32_t dz;
};
```

`x` Pointer x coordinate.

`y` Pointer y coordinate.

`dx` Increase in x of the wheel or *trackpad*.

`dy` Increase in x of the wheel or *trackpad*.

`dz` Increase in x of the wheel or *trackpad*.

struct EvKey

Keyboard event parameters.

```
struct EvKey
{
    vkey_t key;
};
```

`key` Pulsed key or released.

struct EvPos

Parameters of change of position events.

```
struct EvPos
{
    real32_t x;
    real32_t y;
};
```

`x` X coordinate.

`y` Y coordinate.

struct EvSize

Resize event parameters.

```
struct EvSize
{
    real32_t width;
    real32_t height;
};
```

`width` Width (size in x).

`height` Height (size in y).

struct EvWinClose

Window closing Event Parameters.

```
struct EvWinClose
{
    gui_close_t origin;
};
```

`origin` Origin of the close.

struct EvMenu

Menu event parameters.

```
struct EvMenu
{
    uint32_t index;
    gui_state_t state;
    const char_t* str;
};
```

`index` Pressed *item* index.

`state` Pressed *item* status.

`str` Pressed *item* text.

struct EvScroll

Scroll event parameters.

```
struct EvScroll
{
    gui_orient_t orient;
    gui_orient_t scroll;
    real32_t cpos;
};
```

`orient` Scroll bar orientation.

`scroll` Scroll type.

`cpos` Scroll position.

struct EvTbPos

Location of a cell in a table.

```
struct EvTbPos
{
    uint32_t col;
    uint32_t row;
};
```

`col` Column index.

`row` Row index.

struct EvTbRect

Group of cells in a table.

```
struct EvTbRect
{
    uint32_t stcol;
    uint32_t edcol;
    uint32_t strow;
    uint32_t edrow;
};
```

`stcol` Initial column index.

`edcol` End column index.

`strow` Initial row index.

`edrow` End row index.

struct EvTbSel

Selection in a table.

```
struct EvTbSel
{
    ArrSt(uint32_t)* sel;
};
```

`sel` Row indices.

struct EvTbCell

Data from a cell in a table.

```
struct EvTbCell
{
    const char_t* text;
    align_t align;
};
```

`text` Cell text.

`align` Text alignment.

40.2. Functions

gui_start

Start the *Gui* library, reserving space for global internal structures. Internally call `draw2d_start`. It is called automatically by `osmain`.

```
void
gui_start(void);
```

gui_finish

Finish the *Gui* library, freeing up the space of global internal structures. Internally call `draw2d_finish`. It is called automatically by `osmain`.

```
void
gui_finish(void);
```

gui_respack

Register a resource package.

```
void
gui_respack(FPtr_respack func_respack);
```

func_respack Resource constructor.

Remarks:

See *"Resources"* (page 129).

gui_language

Set the language of the registered resources with `gui_respack`.

```
void
gui_language(const char_t *lang);
```

lang Language.

Remarks:

See *"Resources"* (page 129).

gui_text

Get a text string through its resource identifier.

```
const char_t*
gui_text(const ResId id);
```

id Resource Identifier.

Return:

The text string or `NULL` if it is not found.

Remarks:

The resource must belong to a package registered with `gui_respack`.

gui_image

Get an image through its resource identifier.

```
const Image*
gui_image(const ResId id);
```

id Resource Identifier.

Return:

The image or NULL if it is not found.

Remarks:

The resource must belong to a package registered with gui_respack. Do not destroy the image as it is managed by Gui.

gui_file

Get the contents of a file through its resource identifier.

```
const byte_t*
gui_file(const ResId id,
         uint32_t *size);
```

id Resource Identifier.

size Buffer size in bytes.

Return:

File data or NULL if it is not found.

Remarks:

The resource must belong to a package registered with gui_respack. The data is managed by Gui, so there is no need to free memory.

gui_dark_mode

Determines if the window environment has a light or dark theme.

```
bool_t
gui_dark_mode(void);
```

Return:

TRUE for *Dark mode*, FALSE for *light mode*.

gui_alt_color

Create a color with two alternative versions.

```
color_t
gui_alt_color(const color_t light_color,
              const color_t dark_color);
```

light_color Color for LIGHT desktop themes.

dark_color Color for DARK desktop themes.

Return:

The color.

Remarks:

The system will set the final color based on the "lightness" of the window manager colors (Light/Dark). Nested alternate colors ARE NOT ALLOWED. The `light` and `dark` values must be RGB or system colors.

gui_label_color

Returns the default color of text labels `Label`.

```
color_t
gui_label_color(void);
```

Return:

The color.

gui_view_color

Returns the background color in controls `View`.

```
color_t
gui_view_color(void);
```

Return:

The color.

gui_line_color

Returns the color of lines in tables or window separator elements.

```
color_t
gui_line_color(void);
```

Return:

The color.

gui_link_color

Returns the color of the text in hyperlinks.

```
color_t
gui_link_color(void);
```

Return:

The color.

gui_border_color

Returns the border color in button controls, popups, etc..

```
color_t
gui_border_color(void);
```

Return:

The color.

gui_resolution

Returns screen resolution.

```
S2Df
gui_resolution(void);
```

Return:

Resolution.

gui_mouse_pos

Returns the position of the mouse cursor.

```
V2Df
gui_mouse_pos(void);
```

Return:

Position.

gui_update

Refreshes all application windows, after a theme change.

```
void
gui_update(void);
```

Remarks:

Normally it is not necessary to call this method. It is called automatically from `osapp`.

gui_OnThemeChanged

Set a handler to detect the change of the visual theme of the windows environment.

```
void
gui_OnThemeChanged(Listener *listener);
```

listener The event handler.

gui_update_transitions

Update the automatic animations of the interface.

```
void
gui_update_transitions(const real64_t prtime,
                       const real64_t crtime);
```

prtime Time of the previous instant.

crtime Time of the current instant.

Remarks:

Normally it is not necessary to call this method. It is called automatically from `osapp`.

gui_OnNotification

Sets up a handler to receive notifications from `gui`.

```
void
gui_OnNotification(Listener *listener);
```

listener The event handler.

Remarks:

See gui_notif_t.

evbind_object

Gets the object linked to a layout within a callback function.

```
type*
evbind_object(Event *e,
              type);
```

e	The event.
type	The object type.

Return:

The object.

Remarks:

See *"Notifications and calculated fieldsNotifications and calculated fields"* (page 361).

evbind_modify

Checks, inside a callback function, if the object's field has been modified.

```
bool_t
evbind_modify(Event *e,
              type,
              mtype,
              mname);
```

e	The event.
type	The object type.
mtype	The type of the field to check.
mname	The name of the field to check.

Return:

TRUE if the field has been modified.

Remarks:

See *"Notifications and calculated fieldsNotifications and calculated fields"* (page 361).

guicontrol

Gets the base object from a derived control.

```
GuiControl*
guicontrol(type *control);
```

control The derived control. `Label`, `Button`, etc.

Return:

Object.

guicontrol_label

Convert a control to a label.

```
Label*
guicontrol_label(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_button

Convert a control to a button.

```
Button*
guicontrol_button(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_popup

Convert a control to a popup.

```
PopUp*
guicontrol_popup(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_edit

Convert a control to a edit.

```
Edit*
guicontrol_edit(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_combo

Convert a control to a combo.

```
Combo*
guicontrol_combo(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_listbox

Convert a control to a listbox.

```
ListBox*
guicontrol_listbox(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_updown

Convert a control to a updown.

```
UpDown*
guicontrol_updown(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_slider

Convert a control to a slider.

```
Slider*
guicontrol_slider(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_progress

Convert a control to a progress.

```
Progress*
guicontrol_progress(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_view

Convert a control to a view.

```
View*
guicontrol_view(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_textview

Convert a control to a textview.

```
TextView*
guicontrol_textview(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_imageview

Convert a control to a imageview.

```
ImageView*
guicontrol_imageview(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_tableview

Convert a control to a tableview.

```
TableView*
guicontrol_tableview(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_splitview

Convert a control to a splitview.

```
SplitView*
guicontrol_splitview(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_panel

Convert a control to a panel.

```
Panel*
guicontrol_panel(GuiControl *control);
```

control Control.

Return:

Object or `NULL` if `control` cannot be converted.

guicontrol_enabled

Activate or deactivate a control.

```
void
guicontrol_enabled(GuiControl *control,
                   const bool_t enabled);
```

control Control.

enabled Enabled or not.

guicontrol_visible

Show or hide a control.

```
void
guicontrol_visible(GuiControl *control,
                   const bool_t visible);
```

control Control.

visible Visible or not.

label_create

Create a text control.

```
Label*
label_create(void);
```

Return:

The new label.

label_multiline

Create a multi-line text control.

```
Label*
label_multiline(void);
```

Return:

The new label.

label_OnClick

Set the `OnClick` event handler.

```
void
label_OnClick(Label *label,
              Listener *listener);
```

```
static void i_OnClick(UserData *data, Event *e)
{
    const EvText *p = event_params(e, EvText);
    do_something_onclick(data, p->text);
}
...
label_OnClick(label, listener(data, i_OnClick, UserData));
```

label	The label.
listener	Event handler.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

label_text

Set the text that the label will display.

```
void
label_text(Label *label,
           const char_t *text);
```

label	The label.
text	UTF8 C-string terminated in null character `'\0'`.

label_font

Set the text font.

```
void
label_font(Label *label,
           const Font *font);
```

label The label.

font Font.

label_style_over

Set the font modifiers, when the mouse is over the control.

```
void
label_style_over(Label *label,
                 const uint32_t style);
```

label The label.

style Combination of values `fstyle_t`.

label_align

Sets the horizontal alignment of the text with respect to the size of the control.

```
void
label_align(Label *label,
            const align_t align);
```

label The label.

align Alignment.

label_color

Set the text color.

```
void
label_color(Label *label,
            const color_t color);
```

label The label.

color The color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

label_color_over

Set the color of the text, when the mouse is over the control.

```
void
label_color_over(Label *label,
                 const color_t color);
```

label The label.

color The color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

label_bgcolor

Set the background color of the text.

```
void
label_bgcolor(Label *label,
              const color_t color);
```

label The label.

color The color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

label_bgcolor_over

Set the background color of the text, when the mouse is over the control.

```
void
label_bgcolor_over(Label *label,
                   const color_t color);
```

label The label.

color El color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

button_push

Create a push button, the typical `[Accept]`, `[Cancel]`, etc.

```
Button*
button_push(void);
```

Return:

The button.

button_check

Create a checkbox.

```
Button*
button_check(void);
```

Return:

The button.

button_check3

Create a checkbox with three states.

```
Button*
button_check3(void);
```

Return:

The button.

button_radio

Create a radio button.

```
Button*
button_radio(void);
```

Return:

The button.

button_flat

Create a flat button, to which an image can be assigned. It is the typical toolbar button.

```
Button*
button_flat(void);
```

Return:

The button.

button_flatgle

Create a flat button with status. The button will alternate between pressed/released each time you click on it.

```
Button*
button_flatgle(void);
```

Return:

The button.

button_OnClick

Set a function for pressing the button.

```
void
button_OnClick(Button *button,
               Listener *listener);
```

```
static void i_OnClick(UserData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    do_something_onclick(data, p->state);
}
...
button_OnClick(button, listener(data, i_OnClick, UserData));
```

button The button.

listener *Callback* function to be called after clicking.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

button_text

Set the text that the button will display.

```
void
button_text(Button *button,
            const char_t *text);
```

button The button.

text UTF8 C-string terminated in null character `'\0'`.

Remarks:

In flat buttons, the text will be displayed as *tooltip*.

button_text_alt

Set an alternative text.

```
void
button_text_alt(Button *button,
                const char_t *text);
```

button The button.

text UTF8 C-string terminated in null character `'\0'`.

Remarks:

Only applicable on flat buttons with status `button_flatgle`. It will be displayed when the button is in `ekGUI_ON` status.

button_tooltip

Set a tooltip for the button. It is a small explanatory text that will appear when the mouse is over the control.

```
void
button_tooltip(Button *button,
               const char_t *text);
```

button The button.

text UTF8 C-string terminated in null character `'\0'`.

button_font

Set the button font.

```
void
button_font(Button *button,
            const Font *font);
```

button The button.

font Font.

button_image

Set the icon that will show the button.

```
void
button_image(Button *button,
             const Image *image);
```

button The button.

image Image.

Remarks:

Not applicable in checkbox or radiobutton. In flat buttons, the size of the control will be adjusted to the image.

button_image_alt

Set an alternative image for the button.

```
void
button_image_alt(Button *button,
                 const Image *image);
```

button The button.

image Image.

Remarks:

Only applicable on flat buttons with status `button_flatgle`. It will be displayed when the button is in `ekGUI_ON` status.

button_state

Set the button status.

```
void
button_state(Button *button,
             const gui_state_t state);
```

button The button.

state State.

Remarks:

Not applicable on push buttons `button_push`.

button_tag

Sets a numeric tag for the button.

```
void
button_tag(Button *button,
           const uint32_t tag);
```

button The button.

tag The tag.

button_vpadding

Sets the inner vertical margin.

```
void
button_vpadding(Button *button,
                const real32_t padding);
```

button The botón.

padding If `0` there will be no margin between the text and the border of the control. If `<0` the default margin will be set.

button_get_state

Get button status.

```
gui_state_t
button_get_state(Button *button);
```

button The button.

Return:

The state.

Remarks:

Not applicable on push buttons `button_push`.

button_get_tag

Gets the button's tag.

```
uint32_t
button_get_tag(const Button *button);
```

button The button.

Return:

The tag value.

button_get_height

Gets the current height of the control.

```
real32_t
button_get_height(const Button *button);
```

button The button.

Return:

The height of the control, which will change depending on the font size and `vpadding`.

popup_create

Create a new popup control *(PopUp button)*.

```
PopUp*
popup_create(void);
```

Return:

The newly popup.

popup_OnSelect

Set an event handler for the selection of a new item.

```
void
popup_OnSelect(PopUp *popup,
               Listener *listener);
```

```
static void i_OnSelect(UserData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    do_something_onselect(data, p->index);
}
...
popup_OnSelect(popup, listener(data, i_OnSelect, UserData));
```

popup The popup.

listener *Callback* function to be called after selecting a new item from the list.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

popup_tooltip

Assign a tooltip to the popup control.

```
void
popup_tooltip(PopUp *popup,
              const char_t *text);
```

popup The popup.

text UTF8 C-string terminated in null character `'\0'`.

popup_add_elem

Add a new item to the popup list.

```
void
popup_add_elem(PopUp *popup,
               const char_t *text,
               const Image *image);
```

popup The popup.

text The text of the element in UTF-8 or the resource identifier. *"Resources"* (page 129).

image Icon associated with the resource element or identifier. For space, it will scale to a maximum maximum of 16 pixels.

popup_set_elem

Edit an item from the drop-down list.

```
void
popup_set_elem(PopUp *popup,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

popup The popup.

index The index of the item to replace.

text The text of the element in UTF-8 or the resource identifier. *"Resources"* (page 129).

image Icon associated with the resource element or identifier. For space, it will scale to a maximum maximum of 16 pixels.

popup_clear

Remove all items from the dropdown list.

```
void
popup_clear(PopUp *popup);
```

popup The popup.

popup_count

Gets the number of items in the list.

```
uint32_t
popup_count(const PopUp *popup);
```

popup The popup.

Return:

The number of elements.

popup_list_height

Set the size of the drop-down list.

```
void
popup_list_height(PopUp *popup,
                  const uint32_t elems);
```

popup The popup.

elems Number of visible elements. If the control has more, a scroll bar will appear.

popup_selected

Set the selected popup element.

```
void
popup_selected(PopUp *popup,
               const uint32_t index);
```

popup The popup.

index The item to select. If we pass `UINT32_MAX` the selection is removed.

popup_get_selected

Get the selected popup item.

```
uint32_t
popup_get_selected(PopUp *popup);
```

popup The popup.

Return:

The selected item.

edit_create

Create a text edit control.

```
Edit*
edit_create(void);
```

Return:

The edit.

edit_multiline

Create a text editing control that allows multiple lines.

```
Edit*
edit_multiline(void);
```

Return:

The edit.

edit_OnFilter

Set a function to filter the text while editing.

```
void
edit_OnFilter(Edit *edit,
              Listener *listener);
```

edit The edit.

listener *Callback* function to be called after each key press. In `EvTextFilter` from `event_result` filtered text will be returned.

Remarks:

See *"Filter textsFilter texts"* (page 309) and *"GUI EventsGUI Events"* (page 300).

edit_OnChange

Set a function to detect when the text has changed.

```
void
edit_OnChange(Edit *edit,
              Listener *listener);
```

edit The edit.

listener *Callback* function to be called when the control loses focus on the keyboard, indicating the end of the edition.

Remarks:

See *"Filter textsFilter texts"* (page 309) and *"GUI EventsGUI Events"* (page 300).

edit_OnFocus

Sets a handler for keyboard focus.

```
void
edit_OnFocus(Edit *edit,
             Listener *listener);
```

edit The edit.

listener *Callback* function that will be called when keyboard focus is received or lost.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

edit_text

Set the edit control text.

```
void
edit_text(Edit *edit,
          const char_t *text);
```

edit The edit.

text UTF8 C-string terminated in null character `'\0'`.

edit_font

Set the font of the edit control.

```
void
edit_font(Edit *edit,
          const Font *font);
```

edit The edit.

font Font.

edit_align

Set text alignment.

```
void
edit_align(Edit *edit,
           const align_t align);
```

edit The edit.

align Alignment.

edit_passmode

Activate the password mode, which will hide the typed characters.

```
void
edit_passmode(Edit *edit,
              const bool_t passmode);
```

edit	The edit.
passmode	Enable or disable password mode.

edit_editable

Enable or disable editing in the control.

```
void
edit_editable(Edit *edit,
              const bool_t is_editable);
```

edit	The edit.
is_editable	`TRUE`will allow to edit the text (by default).

edit_autoselect

Activate or deactivate auto-selection of text.

```
void
edit_autoselect(Edit *edit,
                const bool_t autoselect);
```

edit	The edit.
autoselect	`TRUE` the control text will be fully selected when it receives focus.

Remarks:

Default `FALSE`.

edit_select

Select text.

```
void
edit_select(Edit *edit,
            const int32_t start,
            const int32_t end);
```

edit The edit.

start Position of the initial character. If `<0`, all text is deselected.

end Position of the final character. If `<0`, all text starting from `start` is selected.

edit_tooltip

Assigns a tooltip to the edit control.

```
void
edit_tooltip(Edit *edit,
             const char_t *text);
```

edit The edit.

text UTF8 C-string terminated in null character `'\0'`.

edit_color

Set the text color.

```
void
edit_color(Edit *edit,
           const color_t color);
```

edit The edit.

color Text color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

edit_color_focus

Sets the color of the text, when the control has the keyboard focus.

```
void
edit_color_focus(Edit *edit,
                 const color_t color);
```

edit The edit.

color Text color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

edit_bgcolor

Set the background color.

```
void
edit_bgcolor(Edit *edit,
             const color_t color);
```

edit The edit.

color Background color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

edit_bgcolor_focus

Sets the background color, when the control has keyboard focus.

```
void
edit_bgcolor_focus(Edit *edit,
                   const color_t color);
```

edit The edit.

color Background color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

edit_phtext

Set an explanatory text for when the control is blank (placeholder).

```
void
edit_phtext(Edit *edit,
            const char_t *text);
```

edit The edit.

text UTF8 C-string terminated in null character `'\0'`.

edit_phcolor

Set the color of the placeholder text.

```
void
edit_phcolor(Edit *edit,
             const color_t color);
```

edit The edit.

color Text color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

edit_phstyle

Set the font style for the placeholder.

```
void
edit_phstyle(Edit *edit,
             const uint32_t fstyle);
```

edit The edit.

fstyle Combination of values of `fstyle_t`.

edit_vpadding

Sets the inner vertical margin.

```
void
edit_vpadding(Edit *edit,
              const real32_t padding);
```

edit The edit.

padding If `0` there will be no margin between the text and the border of the control. If `<0` the default margin will be set.

edit_get_text

Get control text.

```
const char_t*
edit_get_text(const Edit *edit);
```

edit The edit.

Return:

UTF8 C-string terminated in null character `'\0'`.

edit_get_height

Gets the current height of the control.

```
real32_t
edit_get_height(const Edit *edit);
```

edit The edit.

Return:

The height of the control, which will change depending on the font size and `vpadding`.

edit_copy

Copies the selected text to the clipboard.

```
void
edit_copy(const Edit *edit);
```

edit The edit.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

edit_cut

Cuts the selected text, copying it to the clipboard.

```
void
edit_cut(Edit *edit);
```

edit The edit.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

edit_paste

Pastes the text from the clipboard into the caret position.

```
void
edit_paste(Edit *edit);
```

edit The edit.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

combo_create

Create a combo control.

```
Combo*
combo_create(void);
```

Return:

The combo.

combo_OnFilter

Set a function to filter the text while editing.

```
void
combo_OnFilter(Combo *combo,
               Listener *listener);
```

combo	The combo.
listener	*Callback* function to be called after each key press. In `EvTextFilter` from `event_result` filtered text will be returned.

Remarks:

See *"Filter textsFilter texts"* (page 309) and *"GUI EventsGUI Events"* (page 300).

combo_OnChange

Set a function to be called when the text has changed.

```
void
combo_OnChange(Combo *combo,
               Listener *listener);
```

combo	The combo.
listener	*Callback* function to be called when the control loses focus on the keyboard, indicating the end of the edition.

Remarks:

This event will also be launched when you select an item from the list, a sign that the text has changed in the edit box. See *"Filter textsFilter texts"* (page 309) and *"GUI EventsGUI Events"* (page 300).

combo_text

Set the combo edit text.

```
void
combo_text(Combo *combo,
           const char_t *text);
```

combo The combo.

text UTF8 C-string terminated in null character `'\0'`.

combo_align

Set text alignment.

```
void
combo_align(Combo *combo,
            const align_t align);
```

combo The combo.

align Alignment.

combo_tooltip

Assign a tooltip to the control combo.

```
void
combo_tooltip(Combo *combo,
              const char_t *text);
```

combo The combo.

text UTF8 C-string terminated in null character `'\0'`.

combo_color

Set the color of the combo text.

```
void
combo_color(Combo *combo,
            const color_t color);
```

combo The combo.

color Text color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

combo_color_focus

Sets the color of the text, when the control has the keyboard focus.

```
void
combo_color_focus(Combo *combo,
                  const color_t color);
```

combo The combo.

color Text color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

combo_bgcolor

Set the background color.

```
void
combo_bgcolor(Combo *combo,
              const color_t color);
```

combo The combo.

color Background color.

Remarks:

RGB values may not be fully portable. See *"Colors"* (page 277).

combo_bgcolor_focus

Sets the background color when the control has keyboard focus.

```
void
combo_bgcolor_focus(Combo *combo,
                    const color_t color);
```

combo The combo.

color Background color.

combo_phtext

Set an explanatory text for when the control is blank.

```
void
combo_phtext(Combo *combo,
             const char_t *text);
```

combo The combo.

text UTF8 C-string terminated in null character `'\0'`.

combo_phcolor

Set the color of the placeholder text.

```
void
combo_phcolor(Combo *combo,
              const color_t color);
```

combo The combo.

color Text color.

combo_phstyle

Set the font style for the placeholder.

```
void
combo_phstyle(Combo *combo,
              const uint32_t fstyle);
```

combo The combo.

fstyle Combination of values of `fstyle_t`.

combo_get_text

Get control text.

```
const char_t*
combo_get_text(const Combo *combo);
```

combo The combo.

Return:

UTF8 C-string terminated in null character `'\0'`.

combo_count

Gets the number of items in the dropdown list.

```
uint32_t
combo_count(const Combo *combo);
```

combo The combo.

Return:

The number of elements.

combo_add_elem

Add a new item to the drop-down list.

```
void
combo_add_elem(Combo *combo,
               const char_t *text,
               const Image *image);
```

combo The combo.

text The text of the element in UTF-8 or the resource identifier. *"Resources"* (page 129).

image Icon associated with the resource element or identifier. For space, it will scale to a maximum maximum of 16 pixels.

combo_set_elem

Edit an item from the drop-down list.

```
void
combo_set_elem(Combo *combo,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

combo The combo.

index The index of the item to replace.

text The text of the element in UTF-8 or the resource identifier. *"Resources"* (page 129).

image Icon associated with the resource element or identifier. For space, it will scale to a maximum maximum of 16 pixels.

combo_ins_elem

Insert an item in the drop-down list.

```
void
combo_ins_elem(Combo *combo,
               const uint32_t index,
               const char_t *text,
               const Image *image);
```

combo The combo.

index Insertion position.

text The text of the element in UTF-8 or the resource identifier. *"Resources"* (page 129).

image Icon associated with the resource element or identifier. For space, it will scale to a maximum maximum of 16 pixels.

combo_del_elem

Remove an item from the drop-down list.

```
void
combo_del_elem(Combo *combo,
               const uint32_t index);
```

combo The combo.

index The index of the item to delete.

combo_duplicates

Prevents duplicate texts from the drop-down list.

```
void
combo_duplicates(Combo *combo,
                 const bool_t duplicates);
```

combo The combo.

duplicates `TRUE`to allow duplicate texts.

listbox_create

Create a new list control.

```
ListBox*
listbox_create(void);
```

Return:

The newly created listbox.

listbox_OnSelect

Set an event handler for the selection of a new item.

```
void
listbox_OnSelect(ListBox *listbox,
                 Listener *listener);
```

```
static void i_OnSelect(UserData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    do_something_onselect(data, p->index);
}
...
listbox_OnSelect(list, listener(data, i_OnSelect, UserData));
```

listbox The ListBox.

listener *Callback* function to be called after selecting a new item from the list.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

listbox_size

Set the default size of the list.

```
void
listbox_size(ListBox *listbox,
             const S2Df size);
```

listbox The ListBox.

size The size.

Remarks:

It corresponds to *"Natural sizingNatural sizing"* (page 334) of control Default 128x128.

listbox_checkbox

Show or hide checkboxes to the left of items.

```
void
listbox_checkbox(ListBox *listbox,
                 const bool_t show);
```

listbox ListBox.

show TRUE for show them.

listbox_multisel

Enable multiple selection.

```
void
listbox_multisel(ListBox *listbox,
                 const bool_t multisel);
```

listbox ListBox.

multisel TRUE to allow multiple selected items at the same time.

listbox_add_elem

Adds a new element.

```
void
listbox_add_elem(ListBox *listbox,
                 const char_t *text,
                 const Image *image);
```

listbox ListBox.

text The text of the element in UTF-8 or the identifier of the resource. *"Resources"* (page 129).

image Icon associated with the element or resource identifier.

listbox_set_elem

Edit a list item.

```
void
listbox_set_elem(ListBox *listbox,
                 const uint32_t index,
                 const char_t *text,
                 const Image *image);
```

listbox ListBox.

index The index of the element to replace.

text The text of the element in UTF-8 or the identifier of the resource. *"Resources"* (page 129).

image Icon associated with the element or resource identifier.

listbox_clear

Remove all items from the list.

```
void
listbox_clear(ListBox *listbox);
```

listbox ListBox.

listbox_color

Sets the text color of an element.

```
void
listbox_color(ListBox *listbox,
              const color_t color);
```

listbox ListBox.

color The. By default `kCOLOR_DEFAULT`.

listbox_select

Select an item from the program code.

```
void
listbox_select(ListBox *listbox,
               const uint32_t index,
               const bool_t select);
```

listbox ListBox.

index The index of the item to select.

select Select or deselect.

Remarks:

If multiple selection is not enabled, selecting one item implies de-selecting all the others.

listbox_check

Check or uncheck the checkbox of the element from the program code.

```
void
listbox_check(ListBox *listbox,
              const uint32_t index,
              const bool_t check);
```

listbox ListBox.

index The item index.

check Check or uncheck.

Remarks:

Checking an item is independent of selecting it. Items can be marked even if checkboxes are not visible. See `listbox_checkbox`.

listbox_count

Returns the number of elements in the list.

```
uint32_t
listbox_count(const ListBox *listbox);
```

listbox ListBox.

Return:

The number of elements.

listbox_text

Returns the text of an element.

```
const char_t*
listbox_text(const ListBox *listbox);
```

listbox ListBox.

Return:

The UTF-8 text terminated in null character '\0'.

listbox_selected

Returns whether or not an element is selected.

```
bool_t
listbox_selected(const ListBox *listbox);
```

listbox ListBox.

Return:

The selection state.

listbox_checked

Returns whether an element is checked or not.

```
bool_t
listbox_checked(const ListBox *listbox);
```

listbox ListBox.

Return:

The checkbox state.

Remarks:

Checking an item is independent of selecting it. Items can be marked even if checkboxes are not visible. See `listbox_checkbox`.

updown_create

Create an updown control.

```
UpDown*
updown_create(void);
```

Return:

The updown.

updown_OnClick

Set an event handler for pressing the button.

```
void
updown_OnClick(UpDown *updown,
               Listener *listener);
```

```
static void i_OnClick(UserData *data, Event *e)
{
    const EvButton *p = event_params(e, EvButton);
    if (p->index == 0)
        do_something_onup(data);
    else
        do_something_ondown(data);
}
...
updown_OnClick(updown, listener(data, i_OnClick, UserData));
```

updown The updown.

listener *Callback* function to be called after clicking.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

updown_tooltip

Set a tooltip for the button. It is a small explanatory text that will appear when the mouse is over the control.

```
void
updown_tooltip(UpDown *updown,
               const char_t *text);
```

updown The updown.

text UTF8 C-string terminated in null character `'\0'`.

slider_create

Create a new slider control.

```
Slider*
slider_create(void);
```

Return:

Slider.

slider_vertical

Create a new vertical slider.

```
Slider*
slider_vertical(void);
```

Return:

Slider.

slider_OnMoved

Set an event handler for slider movement.

```
void
slider_OnMoved(Slider *slider,
               Listener *listener);
```

```
static void i_OnMoved(UserData *data, Event *e)
{
    const EvSlider *p = event_params(e, EvSlider);
    do_something_onmoved(data, p->pos);
}
...
slider_OnMoved(slider, listener(data, i_OnMoved, UserData));
```

slider Slider.

listener *Callback* function that will be called continuously while the user moves a slider.

Remarks:

`EvSlider` contains the event parameters, see *"GUI EventsGUI Events"* (page 300).

slider_tooltip

Set a tooltip for the slider. It is a small explanatory text that will appear when the mouse is over the control.

```
void
slider_tooltip(Slider *slider,
               const char_t *text);
```

slider Slider.

text UTF8 C-string terminated in null character `'\0'`.

slider_steps

Changes the slider from continuous range to discrete intervals.

```
void
slider_steps(Slider *slider,
             const uint32_t steps);
```

slider Slider.

steps Number of intervals. Use `UINT32_MAX` to return to continuous range.

slider_value

Set the slider position.

```
void
slider_value(Slider *slider,
             const real32_t value);
```

slider Slider.

value The position between 0.0 and 1.0.

slider_get_value

Get the slider position.

```
real32_t
slider_get_value(const Slider *slider);
```

slider Slider.

Return:

The normalized position between 0.0 and 1.0.

progress_create

Create a progress bar.

```
Progress*
progress_create(void);
```

Return:

The progress.

progress_undefined

Set the progress bar as undefined.

```
void
progress_undefined(Progress *progress,
                   const bool_t running);
```

progress The progress.

running TRUEto activate the animation.

progress_value

Set the progress position.

```
void
progress_value(Progress *progress,
               const real32_t value);
```

progress The progress.

value The position between 0.0 and 1.0.

view_create

Create a new custom view.

```
View*
view_create(void);
```

Return:

The view.

view_scroll

Create a new custom view with scrollbars.

```
View*
view_scroll(void);
```

Return:

The view.

view_custom

Create a new view with all the options.

```
View*
view_custom(const bool_t scroll,
            const bool_t border);
```

scroll	Use of scroll bars.
border	Draw a border.

Return:

The view.

Remarks:

Many window managers highlight the border when the view has keyboard focus.

view_data

Associate user data with the view.

```
void
view_data(View *view,
          type **data,
          FPtr_destroy func_destroy_data,
          type);
```

view	The view.
data	User data.
func_destroy_data	Destructor of user data. It will be called upon destroying the view.
type	Type of user data.

view_get_data

Obtiene los datos de usuario asociados con la vista.

```
type*
view_get_data(const View *view,
              type);
```

view	The view.
type	Type of user data.

Return:

Los datos de usuario.

view_size

Set the default view size.

```
void
view_size(View *view,
          const S2Df size);
```

view	The view.
size	The size.

Remarks:

It corresponds to *"Natural sizingNatural sizing"* (page 334) of control Default 128x128.

view_OnDraw

Set an event handler to draw in the view.

```
void
view_OnDraw(View *view,
            Listener *listener);
```

view	The view.
listener	*Callback* function to be called every time the drawing needs to be refreshed.

Remarks:

See *"Draw in viewsDraw in views"* (page 316) and *"GUI EventsGUI Events"* (page 300).

view_OnSize

Set an event handler for resizing.

```
void
view_OnSize(View *view,
            Listener *listener);
```

view	The view.
listener	*Callback* function to be called every time the view changes size.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

view_OnEnter

Set an event handler for mouse enter.

```
void
view_OnEnter(View *view,
             Listener *listener);
```

view — The view.

listener — *Callback* function to be called when the mouse cursor enters the view area.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnExit

Set an event handle for mouse exit.

```
void
view_OnExit(View *view,
            Listener *listener);
```

view — The view.

listener — *Callback* function to be called when the mouse cursor exits the view area.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnMove

Set an event handler for mouse movement.

```
void
view_OnMove(View *view,
            Listener *listener);
```

view The view.

listener *Callback* function to be called as the mouse cursor moves over the view.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnDown

Sets an event handler for a mouse button down.

```
void
view_OnDown(View *view,
            Listener *listener);
```

view The view.

listener *Callback* function that will be called every time the button is down.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnUp

Sets an event handler for a mouse button up.

```
void
view_OnUp(View *view,
          Listener *listener);
```

view The view.

listener *Callback* function that will be called every time the button is up.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnClick

Set an event handler for mouse click.

```
void
view_OnClick(View *view,
             Listener *listener);
```

view The view.

listener *Callback* function that will be called every time the view is clicked.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnDrag

Set an event handler for mouse drag.

```
void
view_OnDrag(View *view,
            Listener *listener);
```

view The view.

listener *Callback* function to be called while dragging the mouse cursor over the view.

Remarks:

"Drag" is to move the mouse with one of the buttons pressed. See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnWheel

Set an event handler for mouse wheel.

```
void
view_OnWheel(View *view,
             Listener *listener);
```

view The view.

listener *Callback* function that will be called when the mouse wheel moves over the view.

Remarks:

See *"Using the mouseUsing the mouse"* (page 318) and *"GUI EventsGUI Events"* (page 300).

view_OnKeyDown

Set an event handler for a keystroke.

```
void
view_OnKeyDown(View *view,
               Listener *listener);
```

view The view.

listener *Callback* function to be called when a key is pressed and the view has the keyboard focus.

Remarks:

See *"Using the keyboardUsing the keyboard"* (page 319) and *"GUI EventsGUI Events"* (page 300).

view_OnKeyUp

Set an event handler for releasing a key.

```
void
view_OnKeyUp(View *view,
             Listener *listener);
```

view The view.

listener *Callback* function to be called when a key is released and the view has the keyboard focus.

Remarks:

See *"Using the keyboardUsing the keyboard"* (page 319) and *"GUI EventsGUI Events"* (page 300).

view_OnFocus

Sets an event handler for keyboard focus.

```
void
view_OnFocus(View *view,
             Listener *listener);
```

view The view.

listener *Callback* function to be called when keyboard focus is received or lost.

Remarks:

See *"Using the keyboardUsing the keyboard"* (page 319) and *"GUI EventsGUI Events"* (page 300).

view_OnResignFocus

Set a handler to avoid losing keyboard focus.

```
void
view_OnResignFocus(View *view,
                   Listener *listener);
```

view The view.

listener *Callback* function that is called when the view is about to lose focus. If we return FALSE, the focus will not go to another control, it will remain in the view.

Remarks:

See *"Using the keyboardUsing the keyboard"* (page 319) and *"GUI EventsGUI Events"* (page 300).

view_OnAcceptFocus

Set a handler to prevent getting keyboard focus.

```
void
view_OnAcceptFocus(View *view,
                   Listener *listener);
```

view The view.

listener *Callback* function that is called when the view is about to get focus. If we return FALSE, the focus will remain on the current control and will not move to the view.

Remarks:

See *"Using the keyboardUsing the keyboard"* (page 319) and *"GUI EventsGUI Events"* (page 300).

view_OnScroll

Set an event handler for the scroll bars.

```
void
view_OnScroll(View *view,
              Listener *listener);
```

view The view.

listener *Callback* function to be called when the user manipulates the scroll bars.

Remarks:

It is not common to have to respond to these events, since the view generates `OnDraw` events automatically when the scroll bars are manipulated. See *"Scrolling viewsScrolling views"* (page 317) and *"GUI EventsGUI Events"* (page 300).

view_keybuf

Sets a keyboard buffer for synchronous or asynchronous query of key state.

```
void
view_keybuf(View *view,
            Keybuf *buffer);
```

view — The view.

buffer — Keyboard buffer that will be maintained by the view, capturing the `OnKeyDown` and `OnKeyUp`events.

Remarks:

It just keeps a reference to the buffer, which will need to be destroyed by the object that created it. See *"Keyboard buffer"* (page 231). The application will still be able to receive keyboard events through `view_OnKeyDown` and `view_OnKeyUp`.

view_get_size

Gets the current size of the view.

```
void
view_get_size(const View *view,
              S2Df *size);
```

view — The view.

size — The size.

view_content_size

Set the size of the drawing area when scroll bars exist.

```
void
view_content_size(View *view,
                  const S2Df size);
```

view — The view.

size — The internal size of the drawing area.

Remarks:

When creating a scroll view, this method indicates the entire drawing area. The control will use it to size and position the scroll bars.

view_scroll_x

Move the horizontal scroll bar to the indicated position.

```
void
view_scroll_x(View *view,
              const real32_t pos);
```

view The view.

pos New horizontal bar position.

view_scroll_y

Move the vertical scroll bar to the indicated position.

```
void
view_scroll_y(View *view,
              const real32_t pos);
```

view The view.

pos New vertical bar position.

view_viewport

Gets the dimensions of the visible area of the view.

```
void
view_viewport(const View *view,
              V2Df *pos,
              S2Df *size);
```

view The view.

pos The position of the viewport. It can be `NULL`.

size The size of the viewport. It can be `NULL`.

Remarks:

If the view does not have scroll bars, `pos` will be `(0,0)`.

view_point_scale

Gets the scaling of the point.

```
void
view_point_scale(const View *view,
                 real32_t *scale);
```

view The view.

scale The scaling.

Remarks:

The view size and drawing coordinates are expressed in points, which typically correspond to pixels (1pt = 1px). In *"Retina displaysRetina displays"* (page 264) it can happen that (1pt = 2px). Although *"2D Contexts"* (page 257) handles this automatically, we may need to know the number of pixels to create another type of *framebuffers* (OpenGL, DirectX, etc). Pixels = `view_get_size` * `view_point_scale`.

view_update

Send an order to the operating system that the view should be refreshed.

```
void
view_update(View *view);
```

view The view.

view_native

Gets a pointer to the native control.

```
void*
view_native(View *view);
```

view The view.

Return:

`HWND` in Windows, `GtkWidget` in Linux and `NSView` in macOS.

Remarks:

Do not use this function if you do not know very well what you are doing.

textview_create

Create a text view.

```
TextView*
textview_create(void);
```

Return:

The text view.

textview_OnFilter

Set a handler to filter text while editing.

```
void
textview_OnFilter(TextView *view,
                  Listener *listener);
```

view The view.

listener *Callback* function that will be called after each key press. In `EvTextFilter` of `event_result` the filtered text will be returned.

Remarks:

See *"Filter textsFilter texts"* (page 309) and *"GUI EventsGUI Events"* (page 300).

textview_OnFocus

Sets a handler for keyboard focus.

```
void
textview_OnFocus(TextView *view,
                 Listener *listener);
```

view The view.

listener *callback* function that will be called when keyboard focus is received or lost.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

textview_size

Sets the default size of the view.

```
void
textview_size(TextView *view,
              const S2Df size);
```

view The view.

size The size.

Remarks:

It corresponds to the *"Natural sizingNatural sizing"* (page 334) of the control. Default 245x144.

textview_clear

Clears all content from view.

```
void
textview_clear(TextView *view);
```

view The view.

textview_printf

Writes text to the view, using the format of the `printf`.

```
uint32_t
textview_printf(TextView *view,
                const char_t *format,
                ...);
```

```
textview_printf(view, Code: %-10s Price %5.2f\n", code, price);
```

view The view.

format String in type-printf format with a variable number of parameters.

... Printf arguments or variables.

Return:

The number of bytes written.

textview_writef

Write a C UTF8 string to the view.

```
void
textview_writef(TextView *view,
                const char_t *str);
```

view The view.

str String C UTF8 terminated in null character '\0'.

textview_rtf

Insert text in Microsoft **RTF** format.

```
void
textview_rtf(TextView *view,
             Stream *rtf_in);
```

view The view.

rtf_in Reading stream with RTF content.

textview_units

Sets the text units.

```
void
textview_units(TextView *view,
               const uint32_t units);
```

view The view.

units Units ekFPIXELS or ekFPOINTS.

Remarks:

ekFPOINTS is the default value and the one normally used by word processors. See *"Size in pointsSize in points"* (page 291).

textview_family

Sets the font family of the text (*"Arial", "Times New Roman", "Helvetica", etc*).

```
void
textview_family(TextView *view,
                const char_t *family);
```

view The view.

family The font family.

Remarks:

Not all families will be present on all platforms. Use `font_exists_family` or `font_installed_fam` to check.

textview_fsize

Set the text size.

```
void
textview_fsize(TextView *view,
               const real32_t size);
```

view	The view.
size	The size.

Remarks:

The value is conditional on the units established in `textview_units`.

textview_fstyle

Sets the text style.

```
void
textview_fstyle(TextView *view,
                const uint32_t fstyle);
```

view	The view.
fstyle	Combination of `ekFBOLD`, `ekFITALIC`, `ekFSTRIKEOUT`, `ekFUNDERLINE` `ekFSUBSCRIPT`, `ekFSUPSCRIPT`. To override any previous style use `ekFNORMAL`.

textview_color

Sets the text color.

```
void
textview_color(TextView *view,
               const color_t color);
```

view	The view.
color	The color. Use `kCOLOR_DEFAULT` to restore the default color.

textview_bgcolor

Sets the background color of the text.

```
void
textview_bgcolor(TextView *view,
                 const color_t color);
```

view The view.

color The color. Use `kCOLOR_DEFAULT` to restore the default color.

textview_pgcolor

Sets the background color of the control.

```
void
textview_pgcolor(TextView *view,
                 const color_t color);
```

view The view.

color The color. Use `kCOLOR_DEFAULT` to restore the default color.

textview_halign

Sets the alignment of text in a paragraph.

```
void
textview_halign(TextView *view,
                const align_t align);
```

view The view.

align The alignment. By default `ekLEFT`.

textview_lspacing

Sets the line spacing of the paragraph.

```
void
textview_lspacing(TextView *view,
                  const real32_t scale);
```

view The view.

scale Scale factor in font height. `1` is the default value, `2` twice this height, `3` triple, etc. Intermediate values are also valid (eg `1.25`).

textview_bfspace

Sets a vertical space before the paragraph.

```
void
textview_bfspace(TextView *view,
                 const real32_t space);
```

view The view.

space The space in the preset units.

textview_afspace

Sets a vertical space after the paragraph.

```
void
textview_afspace(TextView *view,
                 const real32_t space);
```

view The view.

space The space in the preset units.

textview_scroll_visible

Show or hide scroll bars.

```
void
textview_scroll_visible(TextView *view,
                        const bool_t horizontal,
                        const bool_t vertical);
```

view The view.

horizontal Horizontal bar.

vertical Vertical bar.

textview_move_caret

Position the cursor on the indicated character.

```
void
textview_move_caret(TextView *view,
                    const int64_t pos);
```

view The view.

pos Position.

textview_editable

Sets whether or not the control text is editable.

```
void
textview_editable(TextView *view,
                  const bool_t is_editable);
```

view The view.

is_editable TRUE will allow you to edit the text. By default FALSE.

textview_get_text

Gets the text of the control.

```
const char_t*
textview_get_text(const TextView *view);
```

view The view.

Return:

Null-terminated UTF8 C string '\0'.

textview_copy

Copies the selected text to the clipboard.

```
void
textview_copy(const TextView *view);
```

view The view.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

textview_cut

Cuts the selected text, copying it to the clipboard.

```
void
textview_cut(TextView *view);
```

view The view.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

textview_paste

Pastes the text from the clipboard into the caret position.

```
void
textview_paste(TextView *view);
```

view The view.

Remarks:

See *"Clipboard operationsClipboard operations"* (page 311).

imageview_create

Create an image view control.

```
ImageView*
imageview_create(void);
```

Return:

The image view.

imageview_size

Set the default control size.

```
void
imageview_size(ImageView *view,
               const S2Df size);
```

view The view.

size The size.

imageview_scale

Set the scaling to apply to the image.

```
void
imageview_scale(ImageView *view,
                const gui_scale_t scale);
```

view The view.

scale Scaling.

imageview_image

Set the image to be displayed in the control.

```
void
imageview_image(ImageView *view,
                const Image *image);
```

view The view.

image The image to show.

imageview_OnClick

Set a handle for the event click on the image.

```
void
imageview_OnClick(ImageView *view,
                  Listener *listener);
```

view The view.

listener *Callback* function to be called after clicking.

imageview_OnOverDraw

Allows you to draw an *overlay* on the image when the mouse is over it.

```
void
imageview_OnOverDraw(ImageView *view,
                     Listener *listener);
```

view The view.

listener *Callback* function that will be called when the mouse is over the image. Here we will include the additional drawing code.

tableview_create

Creates a new table view.

```
TableView*
tableview_create(void);
```

Return:

The table.

tableview_OnData

Sets up a handler to read data from the application.

```
void
tableview_OnData(TableView *view,
                 Listener *listener);
```

view The table.

listener A *callback* function that will be called each time the table needs to update its content.

Remarks:

See *"Data connectionData connection"* (page 323).

tableview_OnSelect

Notifies that the selection has changed.

```
void
tableview_OnSelect(TableView *view,
                   Listener *listener);
```

view The table.

listener A *callback* function that will be called whenever the selection in the table changes.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_OnRowClick

Notify each time a row is clicked.

```
void
tableview_OnRowClick(TableView *view,
                     Listener *listener);
```

view The table.

listener *Callback* function that will be called every time a row is clicked.

tableview_OnHeaderClick

Notifies each time a header is clicked.

```
void
tableview_OnHeaderClick(TableView *view,
                        Listener *listener);
```

view The table.

listener *Callback* function that will be called every time a table header is clicked.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_font

Sets the general font for the entire table.

```
void
tableview_font(TableView *view,
               const Font *font);
```

view The table.

font Font.

tableview_size

Sets the default size of the table control.

```
void
tableview_size(TableView *view,
               const S2Df size);
```

view The table.

size The size.

Remarks:

Corresponds to the *"Natural sizingNatural sizing"* (page 334) of the control. By default 256x128.

tableview_new_column_text

Adds a new column to the table.

```
uint32_t
tableview_new_column_text(TableView *view);
```

view The table.

Return:

The column identifier (index).

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_column_width

Sets the width of a column.

```
void
tableview_column_width(TableView *view,
                       const uint32_t column_id,
                       const real32_t width);
```

view The table.

column_id The column id.

width The column width.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_column_limits

Sets the size limits of a column.

```
void
tableview_column_limits(TableView *view,
                        const uint32_t column_id,
                        const real32_t min,
                        const real32_t max);
```

view The table.

column_id The column id.

min The minimum width.

max The maximum width.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_column_resizable

Sets whether a column is resizable or not.

```
void
tableview_column_resizable(TableView *view,
                           const uint32_t column_id,
                           const bool_t resizable);
```

view	The table.
column_id	The column id.
resizable	TRUE if resizable.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_column_freeze

Allows freeze the first columns of the table.

```
void
tableview_column_freeze(TableView *view,
                        const uint32_t last_column_id);
```

view	The table.
last_column_id	The identifier of the last column set.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_title

Sets the text of a column header.

```
void
tableview_header_title(TableView *view,
                       const uint32_t column_id,
                       const char_t *text);
```

view	The table.
column_id	The column id.
text	The text in UTF-8 or the identifier of the resource. *"Resources"* (page 129).

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_align

Sets the alignment of the header text.

```
void
tableview_header_align(TableView *view,
                       const uint32_t column_id,
                       const align_t align);
```

view	The table.
column_id	The column id.
align	The alignment.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_visible

Sets whether the table header is visible or not.

```
void
tableview_header_visible(TableView *view,
                         const bool_t visible);
```

view	The table.
visible	`TRUE` to display the header.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_clickable

Sets whether the table header can be clicked as a button.

```
void
tableview_header_clickable(TableView *view,
                           const bool_t clickable);
```

view The table.

clickable TRUE to allow clicks.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_resizable

Sets whether the header allows column resizing.

```
void
tableview_header_resizable(TableView *view,
                           const bool_t resizable);
```

view The table.

resizable TRUE if resizable.

Remarks:

See *"Configure columnsConfigure columns"* (page 329).

tableview_header_height

Force the height of the header.

```
void
tableview_header_height(TableView *view,
                        const real32_t height);
```

view The table.

height The height of the header.

Remarks:

The height of the header is automatically calculated from the content. Forcing this value may cause the table to not display correctly. Its use is not recommended.

tableview_row_height

Force the height of the row.

```
void
tableview_row_height(TableView *view,
                     const real32_t height);
```

view The table.

height The height of the row.

Remarks:

The row height is automatically calculated from the content. Forcing this value may cause the table to not display correctly. its use is not recommended.

tableview_hkey_scroll

Sets the horizontal scrolling when pressing the `[LEFT]` and `[RIGHT]` keys.

```
void
tableview_hkey_scroll(TableView *view,
                      const bool_t force_column,
                      const real32_t scroll);
```

view The table.

force_column If `TRUE` the horizontal content will scroll column by column.

scroll If `force_column=FALSE` indicates the amount in pixels that the table will be moved horizontally when pressing the cursor keys.

tableview_multisel

Sets the row selection mode.

```
void
tableview_multisel(TableView *view,
                   const bool_t multisel,
                   const bool_t preserve);
```

view The table.

multisel `TRUE` to allow multiple selection.

preserve `TRUE` to preserve the selection while browsing.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_grid

Sets the drawing of the interior lines.

```
void
tableview_grid(TableView *view,
               const bool_t hlines,
               const bool_t vlines);
```

view	The table.
hlines	TRUE to draw horizontal lines.
vlines	TRUE to draw vertical lines.

Remarks:

See *"Grid drawingGrid drawing"* (page 329).

tableview_update

Synchronizes the table with the data source.

```
void
tableview_update(TableView *view);
```

view	The table.

Remarks:

See *"Data connectionData connection"* (page 323). We must call this function from the application whenever the data linked to the table changes, in order to update the view.

tableview_select

Selects rows in the table.

```
void
tableview_select(TableView *view,
                 const uint32_t *rows,
                 const uint32_t n);
```

view	The table.
rows	Vector of line indices.
n	Number of elements in the vector.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_deselect

Deselects rows in the table.

```
void
tableview_deselect(TableView *view,
                   const uint32_t *rows,
                   const uint32_t n);
```

view The table.

rows Vector of line indices.

n Number of elements in the vector.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_deselect_all

Deselects all rows in the table.

```
void
tableview_deselect_all(TableView *view);
```

view The table.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_selected

Returns the currently selected rows.

```
const ArrSt(uint32_t)*
tableview_selected(const TableView *view);
```

view The table.

Return:

Array with the indices of the selected rows.

Remarks:

See *"Multiple selectionMultiple selection"* (page 328).

tableview_focus_row

Set keyboard focus to a specific row.

```
void
tableview_focus_row(TableView *view,
                    const uint32_t row,
                    const align_t align);
```

view	The table.
row	The row that will receive focus.
align	Vertical adjustment.

Remarks:

Setting keyboard focus to a row only has effects on navigation, but does not involve selecting the row. The table is automatically scrolled so that the row is visible. In this case, `align` indicates where the vertical scroll is adjusted (up, down or centered).

tableview_get_focus_row

Gets the row that has keyboard focus.

```
void
tableview_get_focus_row(const TableView *view,
                        uint32_t);
```

view	The table.
NOPARAM	The row that has the focus.

tableview_scroll_visible

Show or hide scroll bars.

```
void
tableview_scroll_visible(TableView *view,
                         const bool_t horizontal,
                         const bool_t vertical);
```

view	The table.
horizontal	Horizontal bar.
vertical	Vertical bar.

splitview_horizontal

Create a splitview with horizontal split.

```
SplitView*
splitview_horizontal(void);
```

Return:

The newly created split view.

splitview_vertical

Create a splitview with vertical split.

```
SplitView*
splitview_vertical(void);
```

Return:

The newly created split view.

splitview_size

Sets the default size of the view.

```
void
splitview_size(SplitView *split,
               const S2Df size);
```

split The view.

size The size.

Remarks:

It corresponds to the *"Natural sizingNatural sizing"* (page 334) of the control. Default 128x128.

splitview_view

Add a custom view to the splitview.

```
void
splitview_view(SplitView *split,
               View *view,
               const bool_t tabstop);
```

split The splitview.

view The custom view.

tabstop `TRUE` if we want the view to be part of the tablist. See *"TabstopsTabstops"* (page 338).

Remarks:

See *"Add controlsAdd controls"* (page 330).

splitview_text

Add a text view to the splitview.

```
void
splitview_text(SplitView *split,
               TextView *view,
               const bool_t tabstop);
```

split The splitview.

view The text view.

tabstop `TRUE` if we want the view to be part of the tablist. See *"TabstopsTabstops"* (page 338).

Remarks:

See *"Add controlsAdd controls"* (page 330).

splitview_split

Add a splitview (child) to the splitview.

```
void
splitview_split(SplitView *split,
                SplitView *child);
```

split The splitview.

child The splitview to add.

Remarks:

See *"Add controlsAdd controls"* (page 330).

splitview_panel

Add a panel to the splitview.

```
void
splitview_panel(SplitView *split,
                Panel *panel);
```

split The splitview.

panel The panel.

Remarks:

See *"Add controlsAdd controls"* (page 330).

splitview_pos

Sets the position of the view separator.

```
void
splitview_pos(SplitView *split,
              const real32_t pos);
```

split The splitview.

pos The new position of the separator.

Remarks:

See *"Split modesSplit modes"* (page 331).

layout_create

Create a new layout specifying the number of columns and rows.

```
Layout*
layout_create(const uint32_t ncols,
              const uint32_t nrows);
```

ncols The number of columns.

nrows The number of rows.

Return:

The layout.

layout_cell

Get a layout cell.

```
Cell*
layout_cell(Layout *layout,
            const uint32_t col,
            const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The cell.

layout_control

Gets the control assigned to a cell in the layout.

```
GuiControl*
layout_control(Layout *layout,
               const uint32_t col,
               const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_label

Insert a `Label` control in a layout.

```
void
layout_label(Layout *layout,
             Label *label,
             const uint32_t col,
             const uint32_t row);
```

layout The layout.

label The control to insert.

col Column, cell x coordinate.

row Row, cell y coordinate.

layout_button

Insert a `Button` control in a layout.

```
void
layout_button(Layout *layout,
              Button *button,
              const uint32_t col,
              const uint32_t row);
```

layout	The layout.
button	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_popup

Insert a `PopUp` control in a layout.

```
void
layout_popup(Layout *layout,
             PopUp *popup,
             const uint32_t col,
             const uint32_t row);
```

layout	The layout.
popup	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_edit

Insert an `Edit` control in a layout.

```
void
layout_edit(Layout *layout,
            Edit *edit,
            const uint32_t col,
            const uint32_t row);
```

layout	The layout.
edit	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_combo

Insert a `Combo` control in a layout.

```
void
layout_combo(Layout *layout,
             Combo *combo,
             const uint32_t col,
             const uint32_t row);
```

layout	The layout.
combo	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_listbox

Insert a `ListBox` control in a layout.

```
void
layout_listbox(Layout *layout,
               ListBox *list,
               const uint32_t col,
               const uint32_t row);
```

layout	The layout.
list	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_updown

Insert an `UpDown` control in a layout.

```
void
layout_updown(Layout *layout,
              UpDown *updown,
              const uint32_t col,
              const uint32_t row);
```

layout	The layout.
updown	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_slider

Insert an `Slider` control in a layout.

```
void
layout_slider(Layout *layout,
              Slider *slider,
              const uint32_t col,
              const uint32_t row);
```

layout	The layout.
slider	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_progress

Insert a `Progress` control in a layout.

```
void
layout_progress(Layout *layout,
                Progress *progress,
                const uint32_t col,
                const uint32_t row);
```

layout	The layout.
progress	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_view

Insert `View` in a layout.

```
void
layout_view(Layout *layout,
            View *view,
            const uint32_t col,
            const uint32_t row);
```

layout	The layout.
view	The view to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_textview

Insert a `TextView` control in a layout.

```
void
layout_textview(Layout *layout,
                TextView *view,
                const uint32_t col,
                const uint32_t row);
```

layout	The layout.
view	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_imageview

Insert an `ImageView` control in a layout.

```
void
layout_imageview(Layout *layout,
                 ImageView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
view	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_tableview

Insert an `TableView` control in a layout.

```
void
layout_tableview(Layout *layout,
                 TableView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
view	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_splitview

Insert an `SplitView` control in a layout.

```
void
layout_splitview(Layout *layout,
                 SplitView *view,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
view	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_panel

Insert a `Panel` control in a layout.

```
void
layout_panel(Layout *layout,
             Panel *panel,
             const uint32_t col,
             const uint32_t row);
```

layout	The layout.
panel	The control to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_layout

Insert a layout into a cell in another layout.

```
void
layout_layout(Layout *layout,
              Layout *sublayout,
              const uint32_t col,
              const uint32_t row);
```

layout	The main layout.
sublayout	The layout to insert.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

layout_get_label

Gets the `Label` of a cell.

```
Label*
layout_get_label(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_button

Gets the `Button` of a cell.

```
Button*
layout_get_button(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_popup

Gets the `PopUp` of a cell.

```
PopUp*
layout_get_popup(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_edit

Gets the `Edit` of a cell.

```
Edit*
layout_get_edit(const Layout *layout,
                const uint32_t col,
                const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_combo

Gets the `Combo` of a cell.

```
Combo*
layout_get_combo(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_listbox

Gets the `ListBox` of a cell.

```
ListBox*
layout_get_listbox(const Layout *layout,
                   const uint32_t col,
                   const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_updown

Gets the `UpDown` of a cell.

```
UpDown*
layout_get_updown(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_slider

Gets the `Slider` of a cell.

```
Slider*
layout_get_slider(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_progress

Gets the `Progress` of a cell.

```
Progress*
layout_get_progress(const Layout *layout,
                    const uint32_t col,
                    const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_view

Gets the `View` of a cell.

```
View*
layout_get_view(const Layout *layout,
                const uint32_t col,
                const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The view.

layout_get_textview

Gets the `TextView` of a cell.

```
TextView*
layout_get_textview(const Layout *layout,
                    const uint32_t col,
                    const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_imageview

Gets the `ImageView` of a cell.

```
ImageView*
layout_get_imageview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_tableview

Gets the `TableView` of a cell.

```
TableView*
layout_get_tableview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.

Return:

The control.

layout_get_splitview

Gets the `SplitView` of a cell.

```
SplitView*
layout_get_splitview(const Layout *layout,
                     const uint32_t col,
                     const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The control.

layout_get_panel

Gets the `Panel` of a cell.

```
Panel*
layout_get_panel(const Layout *layout,
                 const uint32_t col,
                 const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The panel.

layout_get_layout

Gets the `Layout` of a cell.

```
Layout*
layout_get_layout(const Layout *layout,
                  const uint32_t col,
                  const uint32_t row);
```

layout The layout.

col Column, cell x coordinate.

row Row, cell y coordinate.

Return:

The sublayout.

layout_taborder

Set how the keyboard focus will move when you press `[TAB]`.

```
void
layout_taborder(Layout *layout,
                const gui_orient_t order);
```

layout	The layout.
order	Loop through rows or columns.

Remarks:

See *"TabstopsTabstops"* (page 338).

layout_tabstop

Sets whether or not a cell in the layout will receive keyboard focus when navigating with `[TAB]-[SHIFT][TAB]`.

```
void
layout_tabstop(Layout *layout,
               const uint32_t col,
               const uint32_t row,
               const bool_t tabstop);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.
tabstop	Enable or disable cell tabstop.

Remarks:

See *"TabstopsTabstops"* (page 338).

layout_hsize

Set a fixed width for a layout column.

```
void
layout_hsize(Layout *layout,
             const uint32_t col,
             const real32_t width);
```

layout	The layout.
col	Column index.
width	Width.

layout_vsize

Force a fixed height for the layout row.

```
void
layout_vsize(Layout *layout,
             const uint32_t row,
             const real32_t height);
```

layout The layout.

row Row index.

height Height.

layout_hmargin

Establish an inter-column margin within the layout. It is the separation between two consecutive columns.

```
void
layout_hmargin(Layout *layout,
               const uint32_t col,
               const real32_t margin);
```

layout The layout.

col Index of the column. The index 0 refers to the separation between column 0 and column 1. `ncols-2` is the maximum accepted value.

margin Margin.

layout_vmargin

Set an inter-row margin within the layout. It is the separation between two consecutive rows.

```
void
layout_vmargin(Layout *layout,
               const uint32_t row,
               const real32_t margin);
```

layout The layout.

row Row index Index 0 refers to the separation between row 0 and row 1. `nrows-2` is the maximum accepted value.

margin Margin.

layout_hexpand

Set the column to expand horizontally.

```
void
layout_hexpand(Layout *layout,
               const uint32_t col);
```

layout The layout.

col Column index.

Remarks:

See *"Cell expansionCell expansion"* (page 338).

layout_hexpand2

Set the two columns that will expand horizontally.

```
void
layout_hexpand2(Layout *layout,
                const uint32_t col1,
                const uint32_t col2,
                const real32_t exp);
```

layout The layout.

col1 Index of column 1.

col2 Index of column 2.

exp Expansion of `col1` between 0 and 1.

Remarks:

The expansion of `col2 = 1 - exp`. See *"Cell expansionCell expansion"* (page 338).

layout_hexpand3

Set the three columns that will expand horizontally.

```
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	The layout.
col1	Index of column 1.
col2	Index of column 2.
col3	Index of column 3.
exp1	Expansion of `col1` between 0 and 1.
exp2	Expansion of `col2` between 0 and 1.

Remarks:

`exp1 + exp2 <= 1`. The expansion of `col3 = 1 - exp1 - exp2`. See *"Cell expansionCell expansion"* (page 338).

layout_vexpand

Set the row that will expand vertically.

```
void
layout_vexpand(Layout *layout,
               const uint32_t row);
```

layout	The layout.
row	Row index.

Remarks:

See *"Cell expansionCell expansion"* (page 338).

layout_vexpand2

Set the two rows that will expand vertically.

```
void
layout_vexpand2(Layout *layout,
                const uint32_t row1,
                const uint32_t row2,
                const real32_t exp);
```

layout	The layout.
row1	Index of row 1.
row2	Index of row 2.
exp	Expansion of `row1` between 0 and 1.

Remarks:

The expansion of `row2 = 1 - exp`. See *"Cell expansionCell expansion"* (page 338).

layout_vexpand3

Set the three rows that will expand horizontally.

```
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	The layout.
row1	Index of row 1.
row2	Index of row 2.
row3	Index of row 3.
exp1	Expansion of `row1` between 0 and 1.
exp2	Expansion of `row2` between 0 and 1.

Remarks:

`exp1 + exp2 < = 1`. The expansion of `row3 = 1 - exp1 - exp2`. See *"Cell expansionCell expansion"* (page 338).

layout_halign

Sets the horizontal alignment of a cell. It will take effect when the column is wider than the cell.

```
void
layout_halign(Layout *layout,
              const uint32_t col,
              const uint32_t row,
              const align_t align);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.
align	Horizontal alignment.

layout_valign

Sets the vertical alignment of a cell. It will take effect when the row is taller than the cell.

```
void
layout_valign(Layout *layout,
              const uint32_t col,
              const uint32_t row,
              const align_t align);
```

layout	The layout.
col	Column, cell x coordinate.
row	Row, cell y coordinate.
align	Vertical alignment.

layout_show_col

Show or hide a layout column.

```
void
layout_show_col(Layout *layout,
                const uint32_t col,
                const bool_t visible);
```

layout	The layout.
col	Column index.
visible	Visible or hidden.

layout_show_row

Show or hide a layout row.

```
void
layout_show_row(Layout *layout,
                const uint32_t row,
                const bool_t visible);
```

layout	The layout.
row	Row index.
visible	Visible or hidden.

layout_margin

Set a uniform margin for the layout border.

```
void
layout_margin(Layout *layout,
              const real32_t mall);
```

layout The layout.

mall Margin for all four sides (left, right, up and down).

layout_margin2

Set a horizontal and vertical margin for the layout edge.

```
void
layout_margin2(Layout *layout,
               const real32_t mtb,
               const real32_t mlr);
```

layout The layout.

mtb Upper and lower margin.

mlr Left and right margin.

layout_margin4

Set margins for the layout border.

```
void
layout_margin4(Layout *layout,
               const real32_t mt,
               const real32_t mr,
               const real32_t mb,
               const real32_t ml);
```

layout The layout.

mt Top edge margin.

mr Right edge margin.

mb Bottom edge margin.

ml Left edge margin.

layout_bgcolor

Assign a background color to the layout.

```
void
layout_bgcolor(Layout *layout,
               const color_t color);
```

layout The layout.

color The color. With `ekCOLOR_TRANSPARENT` default color is restored.

layout_skcolor

Assign a color to the edge of the layout.

```
void
layout_skcolor(Layout *layout,
               const color_t color);
```

layout The layout.

color The color. With `ekCOLOR_TRANSPARENT` default color is restored.

layout_update

Update the window associated with the layout.

```
void
layout_update(Layout *layout);
```

layout The layout.

Remarks:

It is equivalent to calling `window_update`.

layout_dbind

Associate a type `struct` with a layout.

```
void
layout_dbind(Layout *layout,
             Listener *listener,
             type);
```

layout The layout.

listener Will notify through this `listener` every time the object changes. Can be `NULL`.

type The **`struct`** type.

Remarks:

See *"GUI Data binding"* (page 352).

layout_dbind_obj

Associate an object with a layout to view and edit it.

```
void
layout_dbind_obj(Layout *layout,
                 type *obj,
                 type);
```

layout	The layout.
obj	The object to edit.
type	Object type.

Remarks:

See *"GUI Data binding"* (page 352).

layout_dbind_update

Updates the interface of the object associated with the layout.

```
void
layout_dbind_update(Layout *layout,
                    type,
                    mtype,
                    mname);
```

layout	The layout.
type	The object type.
mtype	The type of the field to update.
mname	The name of the field to update.

Remarks:

See *"GUI Data binding"* (page 352).

cell_control

Get control of the inside of the cell.

```
GuiControl*
cell_control(Cell *cell);
```

cell The cell.

Return:

The control.

cell_label

Get the label inside the cell.

```
Label*
cell_label(Cell *cell);
```

cell The cell.

Return:

The control.

cell_button

Get the button inside the cell.

```
Button*
cell_button(Cell *cell);
```

cell The cell.

Return:

The control.

cell_popup

Get the popup inside the cell.

```
PopUp*
cell_popup(Cell *cell);
```

cell The cell.

Return:

The control.

cell_edit

Get the edit inside the cell.

```
Edit*
cell_edit(Cell *cell);
```

cell The cell.

Return:

The control.

cell_combo

Get the combo inside the cell.

```
Combo*
cell_combo(Cell *cell);
```

cell The cell.

Return:

The control.

cell_listbox

Get the listbox inside the cell.

```
ListBox*
cell_listbox(Cell *cell);
```

cell The cell.

Return:

The control.

cell_updown

Get the updown inside the cell.

```
UpDown*
cell_updown(Cell *cell);
```

cell The cell.

Return:

The control.

cell_slider

Get the slider inside the cell.

```
Slider*
cell_slider(Cell *cell);
```

cell The cell.

Return:

The control.

cell_progress

Get the progress inside the cell.

```
Progress*
cell_progress(Cell *cell);
```

cell The cell.

Return:

The control.

cell_view

Get the view inside the cell.

```
View*
cell_view(Cell *cell);
```

cell The cell.

Return:

The view.

cell_textview

Get the textview inside the cell.

```
TextView*
cell_textview(Cell *cell);
```

cell The cell.

Return:

The control.

cell_imageview

Get the imageview inside the cell.

```
ImageView*
cell_imageview(Cell *cell);
```

cell The cell.

Return:

The control.

cell_tableview

Get the tableview inside the cell.

```
TableView*
cell_tableview(Cell *cell);
```

cell The cell.

Return:

The control.

cell_splitview

Get the splitview inside the cell.

```
SplitView*
cell_splitview(Cell *cell);
```

cell The cell.

Return:

The control.

cell_panel

Get the panel inside the cell.

```
Panel*
cell_panel(Cell *cell);
```

cell The cell.

Return:

The control.

cell_layout

Get the layout inside the cell.

```
Layout*
cell_layout(Cell *cell);
```

cell The cell.

Return:

El layout.

cell_enabled

Activate or deactivate a cell.

```
void
cell_enabled(Cell *cell,
             const bool_t enabled);
```

cell The cell.

enabled Enabled or not.

Remarks:

If the cell contains a sublayout, the command will affect all controls recursively.

cell_visible

Show or hide a cell.

```
void
cell_visible(Cell *cell,
             const bool_t visible);
```

cell The cell.

visible Visible or not.

Remarks:

If the cell contains a sublayout, the command will affect all controls recursively.

cell_padding

Set an inner margin.

```
void
cell_padding(Cell *cell,
             const real32_t pall);
```

cell The cell.

pall Inner margin.

cell_padding2

Set an inner margin.

```
void
cell_padding2(Cell *cell,
              const real32_t ptb,
              const real32_t plr);
```

cell The cell.

ptb Upper and lower margin.

plr Left and right margin.

cell_padding4

Set an inner margin.

```
void
cell_padding4(Cell *cell,
              const real32_t pt,
              const real32_t pr,
              const real32_t pb,
              const real32_t pl);
```

cell	The cell.
pt	Top margin.
pr	Right margin.
pb	Bottom margin.
pl	Left margin.

cell_dbind

Associates a cell with the field of a `struct`.

```
void
cell_dbind(Cell *cell,
           type,
           mtype,
           mname);
```

```
cell_dbind(cell, Product, String*, description);
```

cell	The cell.
type	The **struct** type.
mtype	The **struct** field type.
mname	Field name.

Remarks:

See *"GUI Data binding"* (page 352).

panel_create

Create a panel.

```
Panel*
panel_create(void);
```

Return:

The new panel.

panel_scroll

Create a panel with scroll bars.

```
Panel*
panel_scroll(const bool_t hscroll,
             const bool_t vscroll);
```

hscroll TRUE if we want horizontal scroll bar.

vscroll TRUE if we want vertical scroll bar.

Return:

The new panel.

Remarks:

See *"Understanding panel sizingUnderstanding panel sizing"* (page 343).

panel_data

Associate user data with the panel.

```
void
panel_data(Panel *panel,
           type **data,
           FPtr_destroy func_destroy_data,
           type);
```

panel The panel.

data User data.

func_destroy_data Destructor of user data. It will be called when the panel is destroyed.

type Type of user data.

panel_get_data

Get the user data associated with the panel.

```
type*
panel_get_data(const Panel *panel,
               type);
```

panel The panel.

type Type of user data.

Return:

User data.

panel_size

Sets the default size of the visible area of a panel.

```
void
panel_size(Panel *panel,
           const S2Df size);
```

panel The panel.

size The default size.

Remarks:

See *"Understanding panel sizingUnderstanding panel sizing"* (page 343).

panel_layout

Add a layout to a panel.

```
uint32_t
panel_layout(Panel *panel,
             Layout *layout);
```

panel The panel.

layout Layout.

Return:

The newly added layout index.

panel_get_layout

Get a layout of a panel.

```
Layout*
panel_get_layout(Panel *panel,
                 const uint32_t index);
```

panel The panel.

index The layout index.

Return:

Layout.

panel_visible_layout

Set the active layout inside the panel.

```
void
panel_visible_layout(Panel *panel,
                     const uint32_t index);
```

panel The panel.

index The layout index.

Remarks:

To make the change effective, you have to call `panel_update`.

panel_update

Update the window that contains the panel.

```
void
panel_update(Panel *panel);
```

panel The panel.

Remarks:

It is equivalent to calling `window_update`.

panel_scroll_width

Gets the width of the scroll bar of the associated panel.

```
real32_t
panel_scroll_width(const Panel *panel);
```

panel The panel.

Return:

The width of the bar.

Remarks:

Only valid if the panel has been created with `panel_scroll`. Useful if we want to take into account the size of the scroll bars when setting the margins of the Layout.

panel_scroll_height

Gets the height of the scroll bar.

```
real32_t
panel_scroll_height(const Panel *panel);
```

panel The panel.

Return:

The height of the bar.

Remarks:

See panel_scroll_width.

window_create

Create a new window.

```
Window*
window_create(const uint32_t flags);
```

flags Combination of window_flag_t values.

Return:

The window.

window_destroy

Destroy the window and all its contents.

```
void
window_destroy(Window **window);
```

window The window. Will be set to NULL after destruction.

Remarks:

Panels, layouts and components will be recursively destroyed.

window_panel

Associate the main panel with a window.

```
void
window_panel(Window *window,
             Panel *panel);
```

window The window.

panel Main panel, which integrates all the content of the window (views, controls, etc).

Remarks:

The size of the window will be adjusted based on the *"Natural sizingNatural sizing"* (page 334) of the main panel.

window_OnClose

Set an event handler for the window closing.

```
void
window_OnClose(Window *window,
               Listener *listener);
```

window The window.

listener *Callback* function to be called before closing a window.

Remarks:

See *"Closing the windowClosing the window"* (page 348).

window_OnMoved

Set an event handler for moving the window on the desktop.

```
void
window_OnMoved(Window *window,
               Listener *listener);
```

window The window.

listener *Callback* function to be called as the title bar is dragged and the window moves across the desktop.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

window_OnResize

Set an event handler for window resizing.

```
void
window_OnResize(Window *window,
                Listener *listener);
```

window The window.

listener *Callback* function to be called as the outer edges of the window are dragged to resize.

Remarks:

The resizing and relocation of elements is done automatically based on the main `Layout`, so it is not usually necessary for the application to respond to this event. See *"GUI EventsGUI Events"* (page 300).

window_title

Set the text that will display the window in the title bar.

```
void
window_title(Window *window,
             const char_t *text);
```

window The window.

text UTF8 C-string terminated in null character `'\0'`.

window_show

Show the window. By default windows are created hidden. You have to show them explicitly.

```
void
window_show(Window *window);
```

window The window.

window_hide

Hide the window.

```
void
window_hide(Window *window);
```

window The window.

window_modal

Launch a window in **modal** mode.

```
uint32_t
window_modal(Window *window,
             Window *parent);
```

window	The window.
parent	The parent window.

Return:

Value returned by `window_stop_modal`.

Remarks:

`parent` stop receiving events until you call `window_stop_modal`.

window_stop_modal

Ends the **modal** cycle of a window.

```
void
window_stop_modal(Window *window,
                  const uint32_t return_value);
```

window	The window previously launched with `window_modal`.
return_value	Value to be returned `window_modal`.

window_hotkey

Sets an action associated with pressing a key.

```
void
window_hotkey(Window *window,
              const vkey_t key,
              const uint32_t modifiers);
```

window	The window.
key	The key.
modifiers	Modifiers. `0` or combination of `mkey_t`.

Remarks:

See *"HotkeysHotkeys"* (page 352).

window_cycle_tabstop

Activate or deactivate the cyclic behavior of tabstops.

```
void
window_cycle_tabstop(Window *window,
                     const bool_t cycle);
```

window The window.

cycle `TRUE` to activate cycles in tabstops (default).

Remarks:

See *"TabstopsTabstops"* (page 338).

window_next_tabstop

Moves keyboard focus to the next control in the *tab-list*. It has the same effect as pressing `[TAB]`.

```
gui_focus_t
window_next_tabstop(Window *window);
```

window The window.

Return:

Result.

window_previous_tabstop

Moves the keyboard focus to the previous control in the *tab-list*. This has the same effect as pressing `[SHIFT]+[TAB]`.

```
gui_focus_t
window_previous_tabstop(Window *window);
```

window The window.

Return:

Result.

window_focus

Set keyboard focus to a specific control.

```
gui_focus_t
window_focus(Window *window,
             GuiControl *control);
```

window The window.

control The control that the focus will receive.

Return:

Result.

window_get_focus

Gets the control that keyboard focus has.

```
GuiControl
window_get_focus(Window *window);
```

window The window.

Return:

The control.

window_update

Recalculate the position and size of the controls after modifying any `Layout`.

```
void
window_update(Window *window);
```

window The window.

window_origin

Move the window to specific desktop coordinates.

```
void
window_origin(Window *window,
              const V2Df origin);
```

window The window.

origin Position `(x,y)` of the upper-left corner of the window.

window_size

Set the size of the client area of the window.

```
void
window_size(Window *window,
            const S2Df size);
```

window The window.

size Main panel size.

Remarks:

The final size will depend on the window frame and desktop theme settings. This measure only refers to the interior area.

window_get_origin

Get the window position.

```
V2Df
window_get_origin(const Window *window);
```

window The window.

Return:

Position `(x,y)` from the upper-left corner of the window.

window_get_size

Get the total dimensions of the window.

```
S2Df
window_get_size(const Window *window);
```

window The window.

Return:

Window size.

Remarks:

The frame and title bar are taken into account.

window_get_client_size

Get the dimensions of the client area of the window.

```
S2Df
window_get_client_size(const Window *window);
```

window The window.

Return:

Main panel size.

window_defbutton

Set the default window button. It will be activated when pressed `[Intro]`.

```
void
window_defbutton(Window *window,
                 Button *button);
```

window The window.

button The button.

window_cursor

Change the mouse cursor.

```
void
window_cursor(Window *window,
              const gui_cursor_t cursor,
              const Image *image,
              const real32_t hot_x,
              const real32_t hot_y);
```

window The window.

cursor Identifier of the new cursor.

image Custom image. Only valid in `ekGUI_CURSOR_USER`.

hot_x The x coordinate of the click point. Only valid in `ekGUI_CURSOR_USER`.

hot_y The y coordinate of the click point. Only valid in `ekGUI_CURSOR_USER`.

Remarks:

`hot_x, hot_y` indicate the "sensitive" point within the image, which will indicate the exact position of the mouse.

menu_create

Create a new menu.

```
Menu*
menu_create(void);
```

Return:

The new menu.

menu_destroy

Destroy a menu and its entire hierarchy.

```
void
menu_destroy(Menu **menu);
```

menu The menu. Will be set to `NULL` after destruction.

menu_launch

Launch a menu as secondary or *PopUp*.

```
void
menu_launch(Menu *menu,
            const V2Df position);
```

menu The menu.

position Coordinates of the upper left corner.

menu_hide

Hides a secondary *PopUp* menu.

```
void
menu_hide(Menu *menu);
```

menu The menu.

menu_item

Add an item to the menu.

```
void
menu_item(Menu *menu,
          MenuItem *item);
```

menu The menu.

item The item to add.

menu_off_items

Set status `ekGUI_OFF` for all menu items.

```
void
menu_off_items(Menu *menu);
```

menu The menu.

menu_get_item

Get an item from the menu.

```
MenuItem*
menu_get_item(Menu *menu,
              const uint32_t index);
```

menu The menu.

index The index of the item.

Return:

The item.

menu_size

Gets the number of items.

```
uint32_t
menu_size(const Menu *menu);
```

menu The menu.

Return:

Number of items.

menuitem_create

Create a new item for a menu.

```
MenuItem*
menuitem_create(void);
```

Return:

The newly item.

menuitem_separator

Create a new separator for a menu.

```
MenuItem*
menuitem_separator(void);
```

Return:

The newly item.

menuitem_OnClick

Set an event handle for item click.

```
void
menuitem_OnClick(MenuItem *item,
                 Listener *listener);
```

item	The item.
listener	*Callback* function to be called after clicking.

Remarks:

See *"GUI EventsGUI Events"* (page 300).

menuitem_enabled

Enables or disables a menu item.

```
void
menuitem_enabled(MenuItem *item,
                 const bool_t enabled);
```

item	The item.
enabled	Enabled or not.

menuitem_visible

Show or hide a menu item.

```
void
menuitem_visible(MenuItem *item,
                 const bool_t enabled);
```

item The item.

enabled Enabled or not.

menuitem_text

Set the item text.

```
void
menuitem_text(MenuItem *item,
              const char_t *text);
```

item The item.

text UTF8 C-string terminated in null character `'\0'`.

menuitem_image

Set the icon that will display the item.

```
void
menuitem_image(MenuItem *item,
               const Image *image);
```

item The item.

image Image.

menuitem_key

Set a keyboard shortcut to select the menu item.

```
void
menuitem_key(MenuItem *item,
             const vkey_t key,
             const uint32_t modifiers);
```

item The item.

key Key code.

modifiers Modifiers.

menuitem_submenu

Assign a drop-down submenu when selecting the item.

```
void
menuitem_submenu(MenuItem *item,
                 Menu **submenu);
```

item	The item.
submenu	The submenu.

menuitem_state

Set the status of the item, which will be reflected with a mark next to the text.

```
void
menuitem_state(MenuItem *item,
               const gui_state_t state);
```

item	The item.
state	State.

comwin_open_file

Launch the open file dialog.

```
const char_t*
comwin_open_file(Window *parent,
                 const char_t **ftypes,
                 const uint32_t size,
                 const char_t *start_dir);
```

parent	Parent window.
ftypes	File types for the filter.
size	Number of file types.
start_dir	Start directory of the dialog. It can be `NULL`.

Return:

The name of the selected file or `NULL` if the user has aborted the dialog.

Remarks:

It will be launched in **modal**. `parent` will remain locked until the dialog is accepted.

comwin_save_file

Launch the save file dialog.

```
const char_t*
comwin_save_file(Window *parent,
                 const char_t **ftypes,
                 const uint32_t size,
                 const char_t *start_dir);
```

parent	Parent window.
ftypes	File types for the filter.
size	Number of file types.
start_dir	Start directory of the dialog. It can be `NULL`.

Return:

The name of the selected file or `NULL` if the user has aborted the dialog.

Remarks:

It will be launched **modal**. `parent` will remain locked until the dialog is accepted.

comwin_color

Launch the color selection dialog.

```
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	Parent window.
x	Initial x position.
y	Initial y position.
halign	Horizontal alignment with respect to `x`.
valign	Vertical alignment with respect to `y`.
current	Current color the panel will display.
colors	Custom colors that the panel will show and that can also be edited. It can be `NULL` only if `n = 0`.
n	Number of custom colors.
OnChange	*Callback* function to be called after each color change.

Remarks:

On Windows and Linux systems the dialog will be launched modally and must be accepted for a color change notification to occur via `OnChange`. On macOS, notifications will be launched continuously as the dialog is manipulated.

Chapter 41

OSApp library

41.1. Functions

FPtr_app_create

An application constructor prototype.

```
type*
(*FPtr_app_create)(void);
```

Return:

Application object.

FPtr_app_update

Function prototype for update a synchronous application.

```
void
(*FPtr_app_update)(type *app,
                   const real64_t prtime,
                   const real64_t ctime);
```

app	Application object.
prtime	Previous update time.
ctime	Current time.

FPtr_task_main

Function prototype for start a task.

```
uint32_t
(*FPtr_task_main)(type *data);
```

data Initial task data.

Return:

Task return value.

FPtr_task_update

Function prototype of a task update.

```
void
(*FPtr_task_update)(type *data);
```

data Task data.

FPtr_task_end

Function prototype of a task completion.

```
void
(*FPtr_task_end)(type *data,
                 const uint32_t rvalue);
```

data Task Data.

rvalue Task return value.

osmain

Start a desktop application.

```
void
osmain(FPtr_app_create func_create,
       FPtr_destroy func_destroy,
       const char_t *options,
       type);
```

func_create Application object constructor.

func_destroy Application object destructor.

options Options string.

type Type of application object.

Remarks:

In *"Hello World!"* (page 23) you have a simple example of desktop application.

osmain_sync

Start a synchronous desktop application.

```
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	Time in seconds of the update interval (0.04 = 25 fps).
func_create	Application object constructor.
func_destroy	Application object destructor.
func_update	Function to be called in each update interval.
options	Options string.
type	Type of application object.

Remarks:

See *"Synchronous applicationsSynchronous applications"* (page 373).

osapp_finish

End a desktop application, destroying the message cycle and the application object.

```
void
osapp_finish(void);
```

osapp_task

Launch a task in parallel, avoiding the thread lock that controls the user interface.

```
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	Initial task data.
updtime	Update interval time, if required.
func_main	Task start function.
func_update	Task update function.
func_end	Function to be called when finishing the task.
type	Type of initial task data.

Remarks:

See *"Multi-threaded tasksMulti-threaded tasks"* (page 374).

osapp_menubar

Set the general menu bar of the application.

```
void
osapp_menubar(Menu *menu,
              Window *window);
```

menu	The menu.
window	The window that will host the menu.

Remarks:

In macOS the application menu is not linked to any window.

osapp_open_url

Open an Internet address using the default operating system browser.

```
void
osapp_open_url(const char_t *url);
```

url	URL address.

Chapter 42

INet library

42.1. Types and Constants

enum ierror_t

Error codes of network connections.

ekINONET	There is no internet connection on the device.
ekINOHOST	Unable to connect to the remote server.
ekITIMEOUT	Maximum timeout for connection has been exceeded.
ekISTREAM	Error in the I/O channel when reading or writing.
ekISERVER	Error in server response format.
ekINOIMPL	Functionality not implemented.
ekIUNDEF	Undetermined error.
ekIOK	No error.

struct Http

Manage an HTTP connection initiated from the client process.

```
struct Http;
```

struct Url

Allows access to individual fields of a URL (web address) *"URL"* (page 387).

```
struct Url;
```

struct JsonOpts

Options when processing a JSON script.

```
struct JsonOpts;
```

42.2. Functions

http_create

Create an HTTP session.

```
Http*
http_create(const char_t *host,
            const uint16_t port);
```

host Server name.

port Connection port. If we pass `UINT16_MAX` it will use 80 (by default for HTTP).

Return:

HTTP session.

http_secure

Create an HTTPS session.

```
Http*
http_secure(const char_t *host,
            const uint16_t port);
```

host Server name.

port Connection port. If we pass `UINT16_MAX` it will use 413 (by default for HTTPS).

Return:

HTTP session.

http_destroy

Destroy an HTTP object.

```
void
http_destroy(Http **http);
```

http The HTTP object. Will be set to NULL after destruction.

http_clear_headers

Remove previously assigned HTTP headers.

```
void
http_clear_headers(Http *http);
```

http HTTP session.

http_add_header

Add a header to the HTTP request.

```
void
http_add_header(Http *http,
                const char_t *name,
                const char_t *value);
```

http HTTP session.

name The name of the header.

value The header value.

http_get

Make a GET request.

```
bool_t
http_get(Http *http,
         const char_t *path,
         const byte_t *data,
         const uint32_t size,
         ierror_t *error);
```

http HTTP session.

path Resource.

data Data to add in the body of the request. It can be NULL.

size Data block size in bytes.

error Error code if the function fails. It can be NULL.

Return:

TRUE if the request has been processed correctly. If FALSE, in error we will have the cause.

Remarks:

The request is synchronous, that is, the program will be stopped until the server responds. If we want an asynchronous model we will have to create a parallel thread that manages the request. HTTP redirections are resolved automatically.

http_post

Make a POST request.

```
bool_t
http_post(Http *http,
          const char_t *path,
          const byte_t *data,
          const uint32_t size,
          ierror_t *error);
```

http	HTTP session.
path	Resource.
data	Data to add in the body of the request. It can be NULL.
size	Data block size in bytes.
error	Error code if the function fails. It can be NULL.

Return:

TRUE if the request has been processed correctly. If FALSE, in error we will have the cause.

Remarks:

See http_get.

http_response_status

Returns the response code of an HTTP request.

```
uint32_t
http_response_status(const Http *http);
```

http	HTTP session.

Return:

The response code from the server.

http_response_protocol

Returns the protocol used by the HTTP server.

```
const char_t*
http_response_protocol(const Http *http);
```

http HTTP session.

Return:

The server protocol.

http_response_message

Returns the response message from the HTTP server.

```
const char_t*
http_response_message(const Http *http);
```

http HTTP session.

Return:

The response message from the server.

http_response_size

Returns the number of response headers from an HTTP request.

```
uint32_t
http_response_size(const Http *http);
```

http HTTP session.

Return:

The number of headers.

http_response_name

Returns the name of the response header of an HTTP request.

```
const char_t*
http_response_name(const Http *http,
                   const uint32_t index);
```

http HTTP session.

index The index of the header (0, size-1).

Return:

The name of the header.

http_response_value

Returns the value of the response header of an HTTP request.

```
const char_t*
http_response_value(const Http *http,
                    const uint32_t index);
```

http HTTP session.

index The index of the header (0, size-1).

Return:

The value of the header.

http_response_header

Returns the value of a response header from an HTTP request.

```
const char_t*
http_response_header(const Http *http,
                     const char_t *name);
```

http HTTP session.

name The name of the desired header.

Return:

The value of the header. If the header does not exist, it will return an empty string "".

http_response_body

Returns the response body of an HTTP request.

```
bool_t
http_response_body(const Http *http,
                   Stream *body,
                   ierror_t *error);
```

http	HTTP session.
body	Write stream where the response content will be stored.
error	Error code if the function fails. It can be NULL.

Return:

TRUE if it was read successfully. If FALSE, in error we will have the cause.

http_dget

Make a direct request for a web resource.

```
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	Resource URL.
result	Server response code. It can be NULL.
error	Error code if the function fails. It can be NULL.

Return:

Stream with the result of the request.

Remarks:

Use this function for direct access to an isolated resource. If you need to make several requests or configure the headers, use http_create or http_secure.

http_exists

Check if a web resource is available / accessible.

```
bool_t
http_exists(const char_t *url);
```

url Resource URL.

Return:

TRUE if the resource (web page, file, etc) is accessible.

Remarks:

HTTP redirections are not resolved. It will return FALSE if the URL as is is not valid.

json_read

Parse a JSON script. It will transform JSON text into a type or object in C.

```
type*
json_read(Stream *stm,
          const JsonOpts *opts,
          type);
```

stm Data entry in JSON format.

opts Options.

type Type of data.

Return:

Result object.

Remarks:

See *"JSON parsing and conversion to data in CJSON parsing and conversion to data in C"* (page 381).

json_write

Write data in C to JSON format.

```
void
json_write(Stream *stm,
           type *data,
           const JsonOpts *opts,
           type);
```

stm	Data output in JSON format.
data	Object.
opts	Options.
type	Type of data.

Remarks:

See *"Convert from C to JSONConvert from C to JSON"* (page 384).

json_destroy

Destroys a JSON object, previously created with `json_read` .

```
void
json_destroy(type **data,
             type);
```

data	Object.
type	Type of data.

json_destopt

Destroys a JSON object, previously created with `json_read`, if it is not `NULL`.

```
void
json_destopt(type **data,
             type);
```

data	Object.
type	Type of data.

url_parse

Create a URL object from a text string.

```
Url*
url_parse(const char_t *url);
```

url	Null-terminated UTF8 C text string `'\0'`.

Return:

Result URL object after parsing the string.

url_destroy

Destroy the URL object.

```
void
url_destroy(Url **url);
```

url URL object. Will be set to `NULL` after destruction.

url_scheme

Gets the scheme (protocol) of the URL.

```
const char_t*
url_scheme(const Url *url);
```

url URL object.

Return:

Protocol (http, https, ftp, etc).

url_user

Gets the user.

```
const char_t*
url_user(const Url *url);
```

url URL object.

Return:

User or "" if not specified.

url_pass

Get the password.

```
const char_t*
url_pass(const Url *url);
```

url URL object.

Return:

Password or "" if not specified.

url_host

Gets the name of the server.

```
const char_t*
url_host(const Url *url);
```

url URL object.

Return:

Host (Pe. `www.google.com`).

url_path

Gets the path (directories + name) of the requested file or resource.

```
const char_t*
url_path(const Url *url);
```

url URL object.

Return:

Pathname (Pe. `/dir1/dir2/file.html`).

url_params

Gets the parameters (from `';'`) of the URL.

```
const char_t*
url_params(const Url *url);
```

url URL object.

Return:

Parameters or `""` if not specified.

url_query

Gets the parameters (from `'?'`) of the URL.

```
const char_t*
url_query(const Url *url);
```

url URL object.

Return:

Parameters or `""` if not specified.

url_fragment

Gets the fragment (position or anchor of the document) of the URL.

```
const char_t*
url_fragment(const Url *url);
```

url URL object.

Return:

Fragment or `""` if not specified.

url_resource

Get the full address of a resource within the server.

```
String*
url_resource(const Url *url);
```

url URL object.

Return:

Resource. `path + ";"+ params + "?"+ query + "#"+ fragment`.

url_port

Gets the access port to the server.

```
uint16_t
url_port(const Url *url);
```

url URL object.

Return:

Port. `UINT16_MAX` if not specified.

b64_encoded_size

Get the number of bytes needed to encode a memory block in format **base64**.

```
uint32_t
b64_encoded_size(const uint32_t data_size);
```

data_size The original block size.

Return:

Base64 size.

b64_decoded_size

Get the number of bytes needed to decode a block of memory in **base64** format.

```
uint32_t
b64_decoded_size(const uint32_t data_size);
```

data_size The block size encoded in base64.

Return:

The size in bytes.

b64_encode

Encode a block of memory in **base64**.

```
uint32_t
b64_encode(const byte_t *data,
           const uint32_t size,
           char_t *base64);
```

data The data block.

size Block size.

base64 The buffer where to store the result.

Return:

The size in bytes.

Remarks:

The buffer `base64` must be at least the size returned by `b64_encoded_size`.

b64_decode

De-encode a block **base64**.

```
uint32_t
b64_decode(const char_t *base64,
           const uint32_t size,
           byte_t *data);
```

base64	The base64 block.
size	Block size.
data	The buffer where to store the result.

Return:

The size in bytes.

Remarks:

The buffer `data` must be at least the size returned by `b64_decoded_size`.

Index