USER'S MANUAL

Nema GUI Builder

A-SOCAPG-UMGA04EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Rev	Date	Description
1.0	December 2025	Initial Release

Reference Documents

Document ID	Description
A-SOCAPG-UMGA01EN	Nema DC API Library User's Manual
A-SOCAPG-UMGA02EN	Nema GFX API Library User's Manual
A-SOCAPG-ANGA01EN	Nema DC MiP Panels Configuration Application Note
A-SOCAPG-ANGA02EN	Nema GFX Extensions TSVG Supported Elements List Application Note
A-SOCAPG-SGGA01EN	Nema Pixpresso Starting Guide
A-SOCAPG-ANGA03EN	Nema GFX API Debugging Application Note
A-SOCAPG-UMGA04EN	Nema GFX Extension Vector Graphics User's Manual
A-SOCAPG-UMGA05EN	Nema GFX Benchmark Suite User's Manual
A-SOCAPG-UMGA06EN	Nema Pico Graphics Processing Unit User's Manual
A-SOCAPG-UMGA07EN	Nema Pico Platform Drivers User's Manual

Table of Contents

List of Tables

List of Figures

SECTION

1

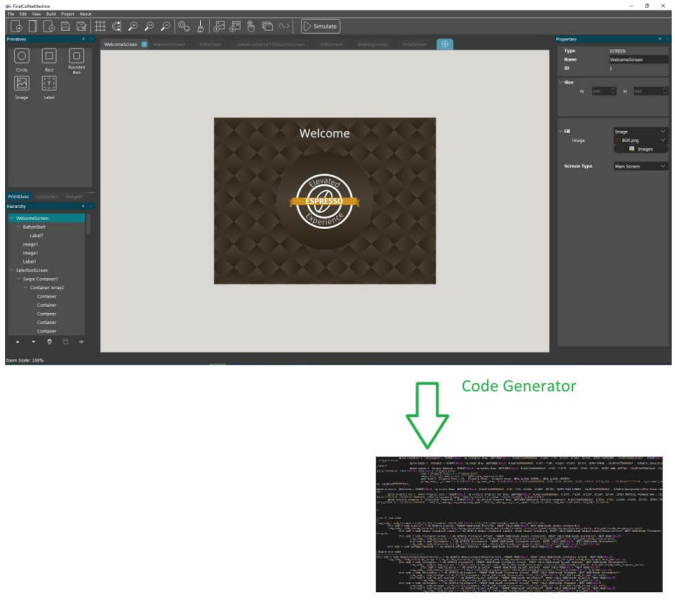
Overview

Nema GUI Builder is one of the main software tools available in Nema GPU's ecosystem. Its main purpose is to provide the end users a simple and flexible tool for rapid graphical user interfaces development, tailored for ultra-low power systems.

By taking advantage of the Nema GUI Builder, one can simply design high quality, seamless and interactive graphical interfaces within minutes instead of spending months on developing the corresponding code that produces equivalent graphical result. In addition, there is no need for the end users to know details about the underlying graphics API.

The procedure followed by this tool is simple and straightforward. The user performs drag-n-drop operations on the desired graphics items in a design area, selects the associated assets (images and fonts), sets the desired events and then the tool automatically generates the code that implements the designed graphical interface. The generated code can be afterwards compiled and deployed on a compatible system.

Figure 1-1: Nema GUI Builder



Think Silicon's proprietary Nema GFX is utilized in the generated code which makes it compatible with a great variety of computing systems (multiple operating systems or bare metal systems). In resource constrained devices (such as embedded and wearable devices) it is imperative that the software executed on the device will utilize the available resources in an optimal way. In practical terms, this means that metrics such as the CPU usage or the memory consumption need to be minimum. Keeping such metrics minimum will consequently be beneficial for the battery life of the target device.

This is achieved through the Nema GFX, as its modular architecture is designed specifically for this kind of applications. Its small memory footprint, command lists features (allowing optimal CPU-GPU decoupling), low overhead features and lack of any external dependencies makes it an ideal API for developing vivid graphics on ultra-low power devices.

Moreover, Nema GFX comes in two different flavors; Nema GPU or CPU based rendering. As a result, the generated code of Nema GUI Builder can run on CPU - Nema GPU systems (optimum performance, maximum energy efficiency) but also, on less sophisticated, CPU based embedded systems (where no GPU is available).

System Requirements

- OS: Windows (10), Linux (Ubuntu 32/64-bit)
- Screen resolution: 800x600 or higher
- RAM: at least 256MB
- Hard Drive: 100MB available space.

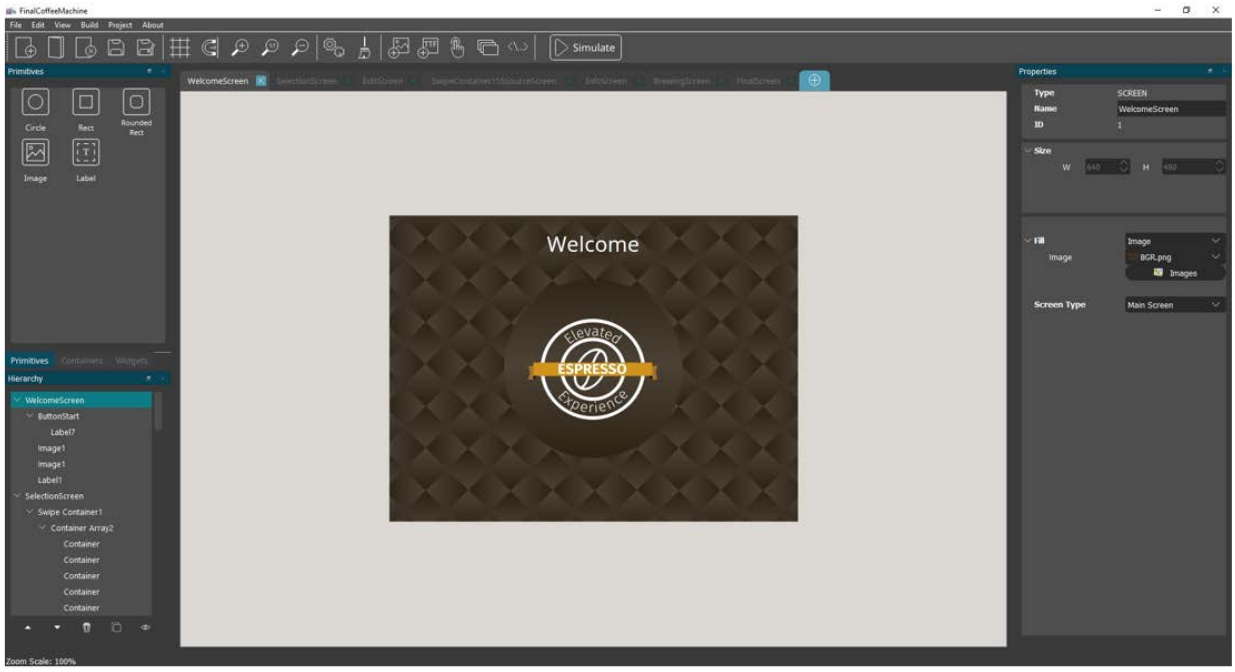
SECTION

2

User Interface

Nema GUI Builder layout is illustrated in Figure 2-1. As one can observe, the user interface consists of four windows (Graphics Items, Screen Hierarchy, Design Area and Properties), each of which (except the Design Area) can be independently displayed or hidden.

Figure 2-1: Nema GUI Builder User Interface



The Graphics Items window (explained in more details in following section) contains the items that the user can drag-and-drop in the Design Area. When such operations take place (or when an item is deleted from the Design Area) the Hierarchy window updates its contents which summarize the contents of the designed GUI. Additionally, one can also drag-and-drop existing items in the Hierarchy window (this is a comfortable way of assigning items to different parent-items). Furthermore, the user is able to review or edit the properties of each graphics item in the Properties window at any time.

SECTION

3

Graphics Items

This section provides a simple application that demonstrates how Nema GFX VG Extensions is Any GUI designed in the Nema GUI Builder consists of several graphics items which are the fundamental items of the GUI. The user can drag-and-drop such items in the **Design Area** and in this stepwise way a GUI can be designed within a minimum amount of time. Graphics items are divided into three categories: primitives, containers and widgets: **Primitives** (circle, rectangle, rounded rectangle, image and label) are elementary items that perform basic drawing operations:

Table 3-1: Primitive Graphics Items

Items	Description
Circle	Circle of desired radius. Can be either filled with a specified color or not.
Rect	Rectangle of desired size (width/height). Filled or not.
Rounded Rect	Same as Rect, corners are rounded according to a desired radius value.
Image	Image item that must be associated to an image asset.
Label	Label for displaying text information. Must be associated to a font asset.

Containers (container, table, window, swipe window) are more complex items than primitives as they can be used in order to group together several other items. More specifically, items inside a container are grouped together so that they can move altogether along with their parent item (container). **Tables** are used to group together several containers for the creation of list-like tables.

Table 3-2: Container Graphics Items

Items	Description
Container	Containers act as parent items to the items they contain. They can be filled or not with a selected color or an image.
Table	Tables consist of several sub-items (containers) in a tabular layout. The user can configure the number of rows and columns, their dimensions and the distance between them. When adding new items, the last added item is copied to the new ones, so that uniform tables can be created in minimum time.
Window	The Window is a special item because of its property to display any screen except its parent within its area (this can be selected by its corresponding Source Screen property). The displayed content is scrollable at runtime and therefore Windows can be used to create scrollable items (e.g., scrolling tables).
Swipe Window	A special case of window (a graphics item that accepts a source screen). Its source screen contains an array of containers that act as swipe pages. The swipe window displays one of these swipe pages and the user can make swipe gestures in order to navigate from one to another.

Widgets (label button, icon button, toggle button, radio button, horizontal slider, vertical slider, digital meter, icon, progress bar, gauge, circular progress, watch face, digital clock) are the graphics items that handle user interactions during application runtime. Widgets are able to send or receive events to/from other widgets. It must be noted that under certain circumstances, some primitives can also act as widgets (i.e. when an image graphics item needs to be displayed at the press of a button, the image receives an event). This functionality is achieved by setting the graphics item's **Interactive** property in the **Properties** window.

Table 3-3: Widget Graphics Items

Items	Description
Label Button	Button containing text. The background of the text (image or color) can be configured when the button is pressed or released.
Icon Button	Button containing an icon. The background of the icon (image or color) can be configured when the button is pressed or released.
Toggle Button	Toggle button consists of many states each of which is visually displayed by an image. In addition, when a toggle button is pressed (highlighted) it can scale its resolution.
Radio Button	Radio buttons must be placed inside a table for grouping multiple radio buttons. Checking a radio button will uncheck all the other radio buttons that belong to the same group (table).

Table 3-3: Widget Graphics Items *(Continued)*

Items	Description
Horizontal Slider	The slider consists of two rectangles (filled and empty) and a container as its indicator. The properties of each sub-item can be edited by selecting the respective item in the Hierarchy window.
Vertical Slider	Same as Horizontal.
Digital Meter	Widget for displaying digital values. The user can edit its background color, precision (number of decimal digits) and initial value.
Icon	Icon consists of an image and a label. Useful in cases whereas pressing it should activate a specific action.
Horizontal Progress Bar	Widget for displaying the progress attribute. Respective events about setting its value have to be manually configured.
Vertical Progress Bar	Same as Horizontal.
Gauge	A gauge with a needle that displays its current value.
Circular Progress	Widget used to display a circular progress by making use of two images (background and foreground).
Watch Face	Displays time as an analog watch. At runtime, the system's wall time is used by default.
Digital Clock	Displays the time in many different formats. At runtime, the system's wall time is used by default.

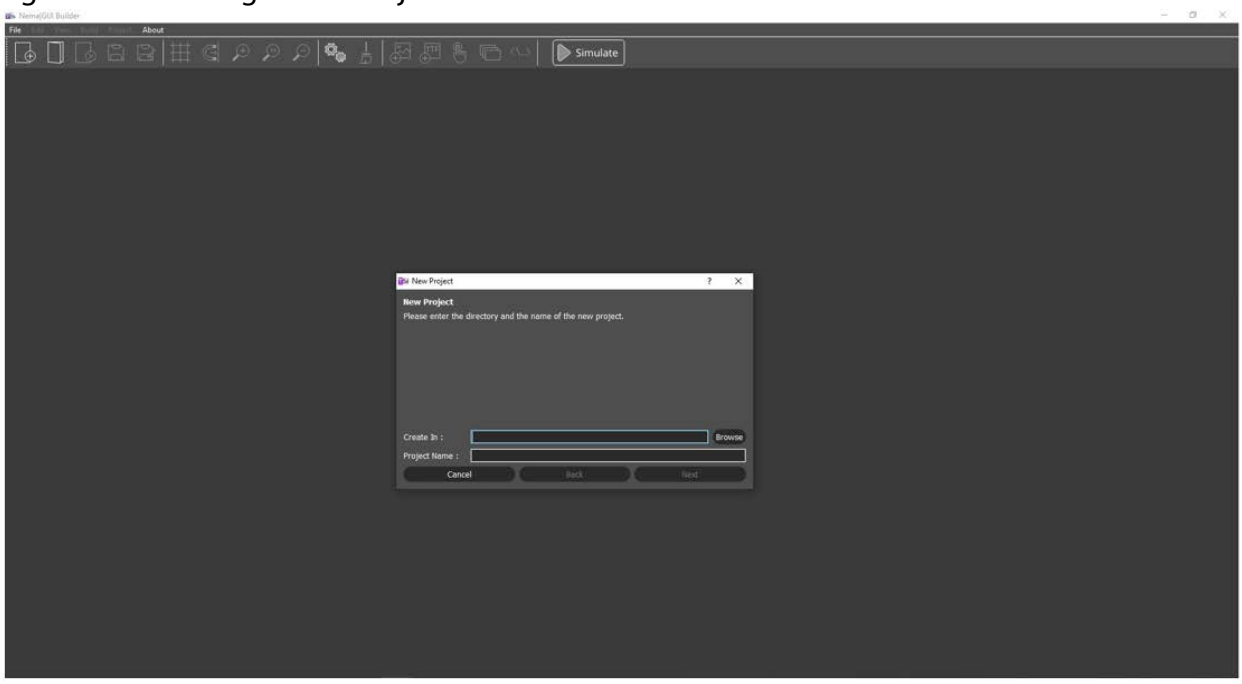
SECTION

4

Basic Functionality

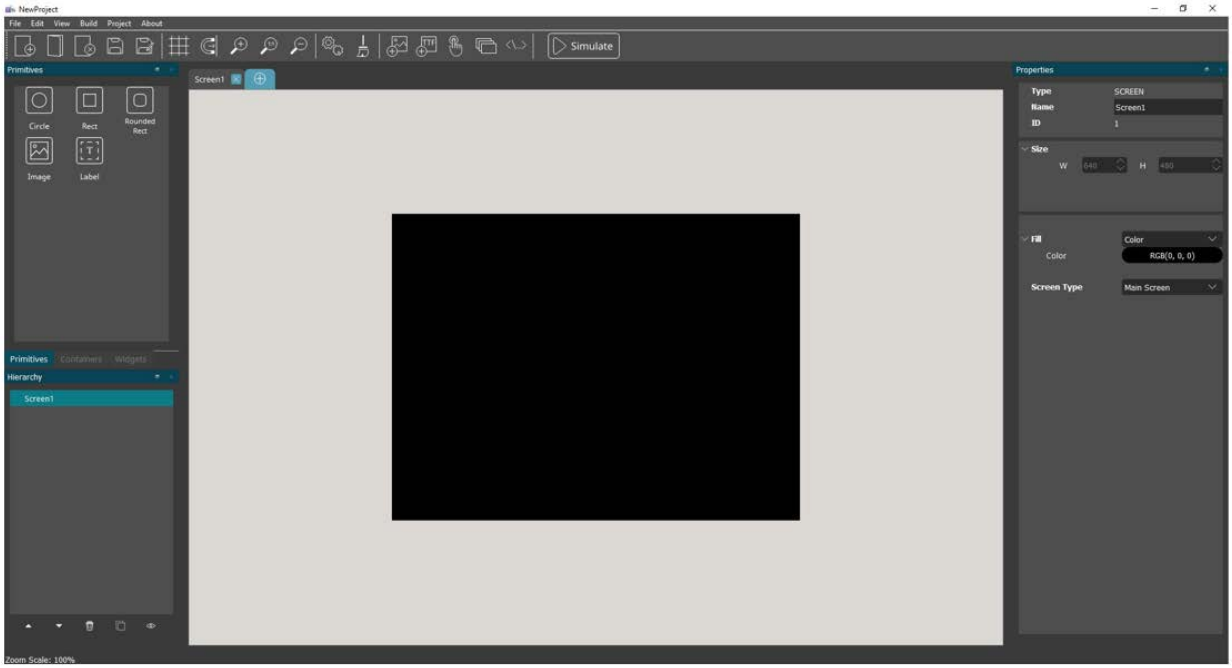
Once the user has opened the application, the GUI design can begin after completing a few steps. The first one is to create a new project (by selecting **New Project** under the **File** menu). At this point the project wizard is invoked (Figure 4-1) and asks the user to enter a project directory and a project name. The wizard will then allocate the entered directory for the needs of the project and all the project related files (assets, generated files etc.) will be saved within this location.

Figure 4-1: Creating a New Project



Subsequently to entering the project's directory the wizard asks the user to enter the projects resolution (resolution of the target screen). After completing these steps, the user can start designing the GUI in the **Design Area** (Figure 4-2 on page 16).

Figure 4-2: Creating a New Project From Scratch

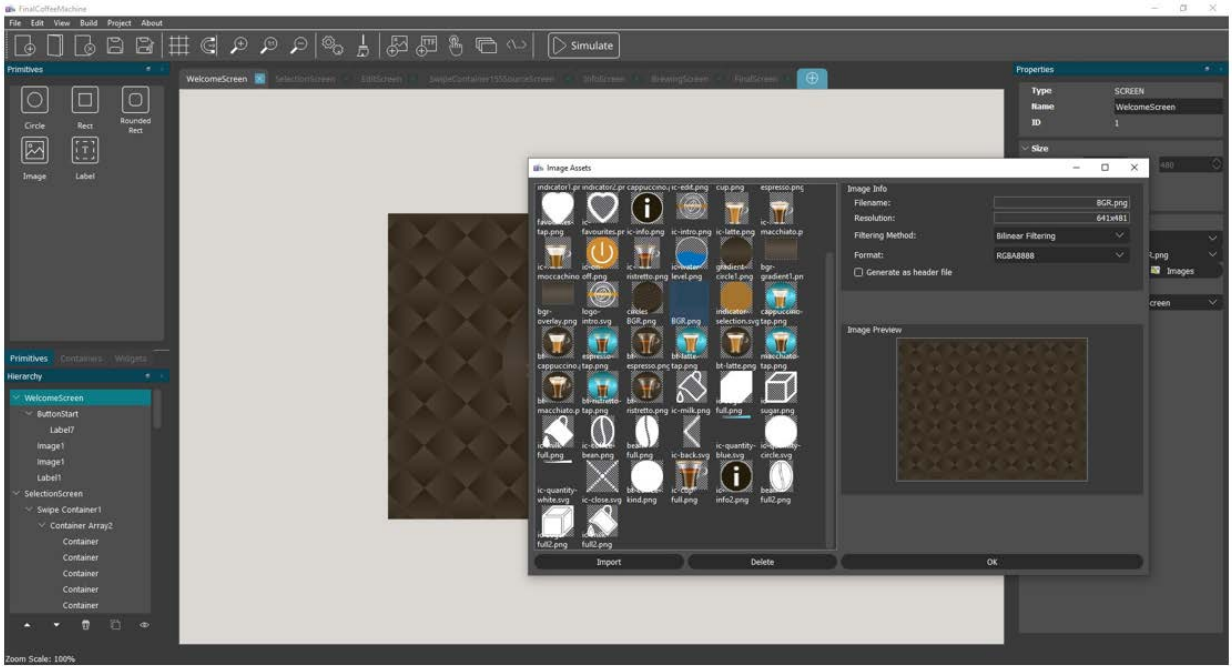


At this point, the user can see a blank screen as there are no items added in it. More screens can be added by selecting the **(+)** icon in the design area. Screens can either be configured as Primary or Secondary screens. This affects the way they are displayed at application runtime.

Primary screens have fixed resolution and when they are displayed they occupy the whole area of the framebuffer. Secondary screens have variable resolution so that they can be used along with a **Window** item or they can be displayed as pop-up screens.

The background of the screen can be edited by the Properties window. This can either be a plain color or an image. Images must be added in project's assets before they can be used in the project. This is performed by clicking **Import** in the Properties window, or by selecting **Assets/Images** under the **Project** menu. This will pop-up a form that allows the user to inspect and modify the project's images (Figure 4-3 on page 17).

Figure 4-3: Importing Images in a Project



In addition, the user can zoom in/out the design area. The zooming operation is performed by pressing the **Ctrl** key and scrolling the mouse wheel, while the mouse cursor is within the design area, or alternatively by using the keyboard hotkeys:

- Ctrl + = or Ctrl + + (zoom in), • Ctrl + - (zoom out)
- Ctrl + 0 (reset zoom to 100%)

The zoom scale is displayed at the status bar (at the bottom of the designed area).

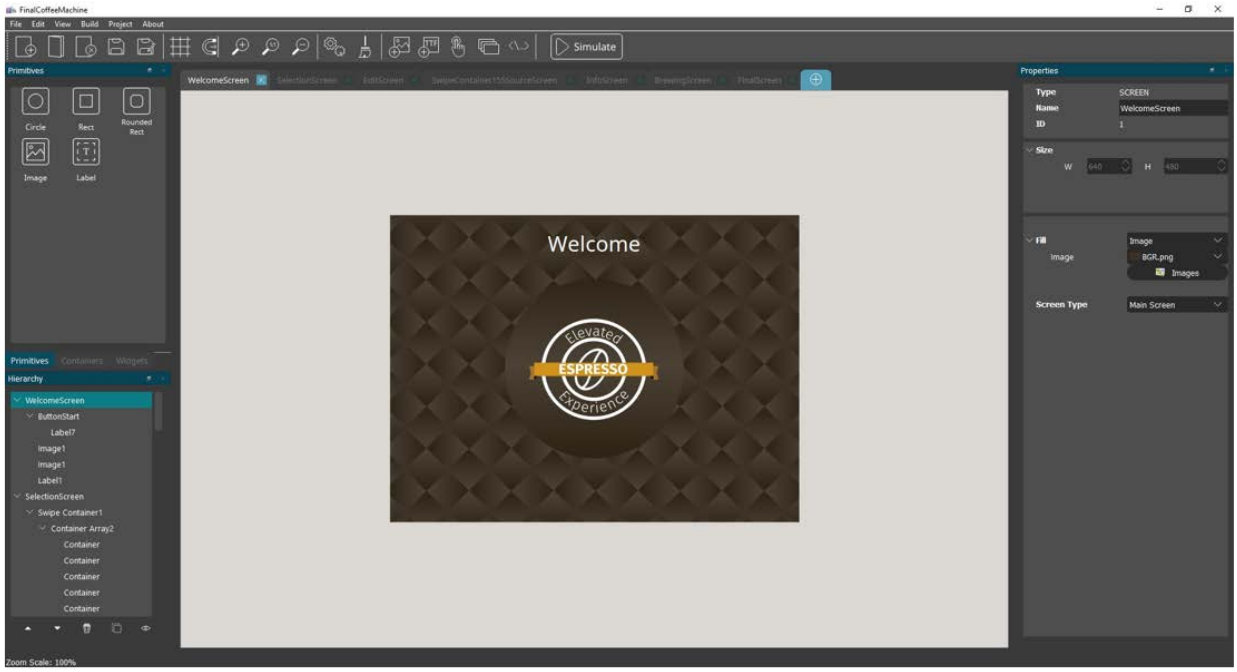
The imported images can now be used to customize image compatible graphics items (images, containers etc.). After setting the screen background and adding a container in the **Design Area**, the application should look as in Figure 4-4 on page 18. Note that in this figure the **Grid** is also visible.

The Grid is a feature that eases the alignment of graphics items. It can be shown or hidden by checking the respective check box under the **View** menu. In addition, graphics items can be snapped to grid by selecting the corresponding option while the grid parameters (width, height, horizontal and vertical offset) can be modified under the same menu. This guarantees that the items will be aligned on the desired line of the grid, thus avoiding the manual alignment by setting each of the coordinates of every item. Note that snapping is available only for the top-left corner of each item.

Graphics items support generic capabilities such as copy, paste, cut, delete, bring to front and send to back. Whenever such actions take place in **Containers**, the same action takes place for the contained graphics items. The default graphics items, that Nema GUI Builder offers, include some complex items.

These items consist of a set of basic items (e.g., sliders that are made of a progress bar and an indicator container). Manipulating the basic items can be performed by the **Hierarchy** window or by double clicking on them in the **Design Area**. The user can select these items in the **Hierarchy** window and edit their properties in the **Properties** window. In addition, supported drag-and-drop operations in such items can also be performed in the **Hierarchy** window (by dragging the name of the desired item and dropping it over the name of the desired parent item). Selecting also sub-items, can be achieved by double-clicking consecutively on them (until the get selected).

Figure 4-4: Design Area After the Addition of Graphic Items



In this way the functionality of complex items is extended as they come with some default features (so that the user does not have to worry about them at first hand) and they can be customized to the demands of each application (e.g., dropping a circle inside a slider's container, changes the way the slider's indicator can look like).

Furthermore, a project can be saved in the designated location (project directory) as *.tsg file. This will save:

- The project's structure (in xml format)
- The Assets (images and fonts)
- The Events (explained in more detail in following section)

A saved project with these features (structure, assets, and events) can then be opened by the application for further modifications. In addition, Nema GUI Builder is configured to auto-save the current project silently every 2 minutes. In this way, one can recover the projects content when this action is necessary (e.g., data loss or corrupted files). The backup files are located inside the project directory in the folder Backup. In order to recover the project from the Backup files, the user must copy the backup files to the project sources manually.

When a project has been saved, the user must be very cautious when modifying the saved files outside the Nema GUI Builder. This could break associations between these files by breaking the project's structure or even by making the project incompatible with Nema GUI Builder.

SECTION

5

Project Assets

At the current state of the application, images and fonts compose the assets of the project. This is a very important feature as they allow the customization of graphics items that contain images or text (that are ubiquitous in GUI design).

5.1 Images

As explained in *Section 4 Basic Functionality on page 15*, images that are necessary for the GUI must be imported to it before becoming available for graphics item customization. Images can be imported by the corresponding menu under the **Project** menu. In addition, graphics items that support images have a respective button in their **Properties** window in order to ease the user when importing new images. Current supported formats are PNG, JPG and SVG image formats.

After importing an image to the assets, the user can modify the target format that is going to be used during runtime. The imported images must be converted into a format suitable for low power devices. Such formats require direct pixel mapping in the memory subsystem of the device. Nema GUI Builder currently supports the following formats:

- RGBA8888 (32 bits-per-pixel)
- RGBA5650 (16 bpp) • RGBA5551 (16 bpp)
- RGBA4444 (16 bpp)
- L8 (luminance-only 8bpp)
- A8 (transparency-only 8bpp)
- Think Silicon's proprietary and patented formats:
  - TSC4 (4 bpp)
  - TSC6 (6 bpp)
  - TSC6a (6 bpp with alpha channel support).

The default format for newly imported images is set to RGBA8888 format.

Note that the format of an associated image asset to an item with opacity (opacity value different than 255) must support opacity, otherwise these items will not be displayed properly during the application runtime. Think Silicon's formats (TSC4 and TSC6) as well as RGBA5650 do not support opacity. When an image asset's format is A8, the user can also select a default color, in order to colorize the displayed image.

Table 5-1 summarizes the memory requirements for each format.

Table 5-1: Memory Bytes Per Pixel for Various Image Formats

Format	Bytes/Pixel
RGBA8888	4
RGBA5650	2
RGBA5551	2
RGBA4444	2
RGBA3320	1
TSC4	0.5
TSC6	0.75
TSC6A	0.75
L8	1
A8	1

Besides the target format that affects the application memory requirements, the user is also free to select the texture filtering method (the way that image pixels are rendered on the output screen pixels). This method can either be "point-sampling filtering" (also known as "nearest-neighbor filtering") or "bilinear filtering". The first one offers high performance versus poor rendering quality while the latter one trades-off performance to rendering quality.

Depending on the nature of each image (not all the images of a GUI contain high level of details) the users have the freedom to tune the application to their actual requirements. Figure 5-1and Figure 5-2 on page 22 illustrates the difference between these two texture filtering methods.

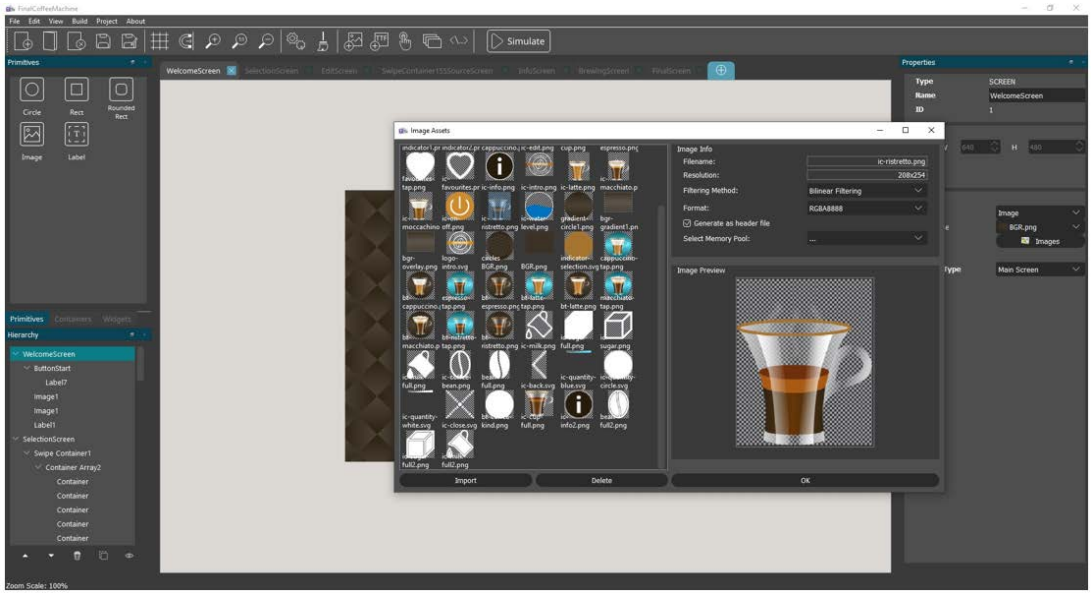
Figure 5-1: Point-Sampling Filtering

Figure 5-2: Bilinear Filtering

At the runtime of a deployed project, the generated assets (images and fonts) are usually stored in a file system (e.g., an SD card). These stored files are then loaded to the main memory of the system so that the application can start executing. Nevertheless there exist systems that are not equipped with a file system. In this case, the images can be generated as header file so that they can be included in the

source code of the generated project. For instance, the image **mostlysunny.svg** of Figure 5-3 will be generated as header file.

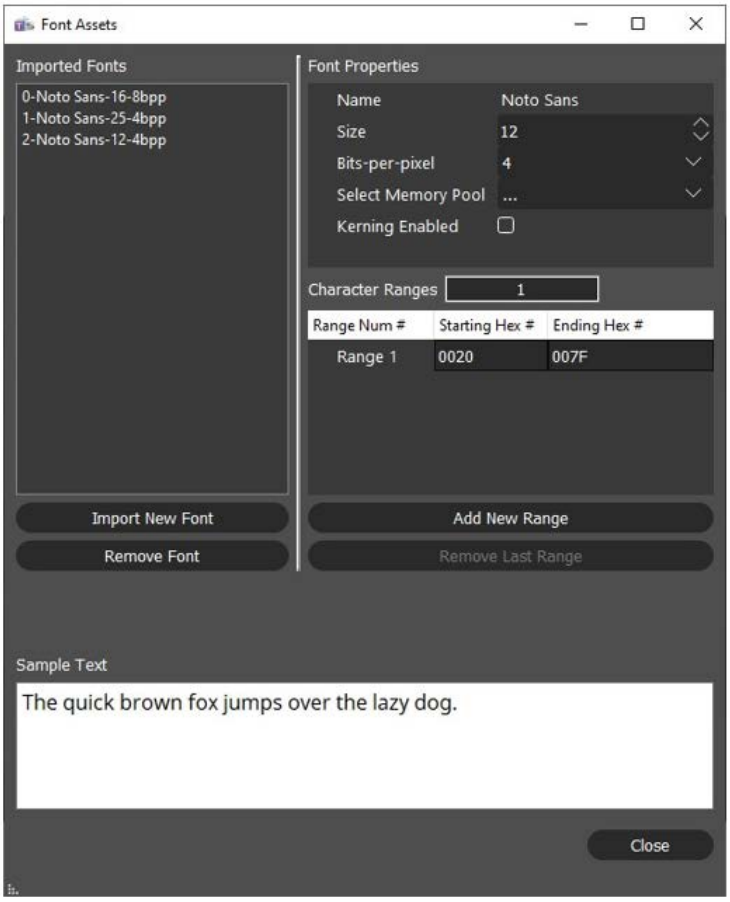
Figure 5-3: Option of Generating Image Assets as Header Files



5.2 Fonts

Fonts are necessary for customizing any graphics items with text support such as labels, digital meters etc. Nema GUI Builder includes the NotoSans_Regular-12 (font family: NotoSans-Regular, font size: 12) as default font. Nevertheless, the end user can import at any time new fonts from the respective **Asset** menu. Only true-type-fonts are currently supported and consequently the user can only import such files (.ttf files). Figure 5-4 on page 24 shows the form used for this purpose.

Figure 5-4: Form Used for Adding a New Font



When importing a font, its size, bits-per-pixel, range count as well as the start and end values for each range need to be set. By using the default values, the imported font will have size 12, 8 bits-per-pixel and one range that spans from value 32 to 127; this is the range for the ASCII character set. In order to include characters beyond this range, the start and end values can be modified. However, is must be noted that the more characters are included the bigger the memory footprint will be. Therefore, the user is advised to make use of multiple ranges that contain exactly the number of character needed by the application. Each range can contain characters that belong to the Unicode character set: 0 up to 10FFFF (hexadecimal value). In addition, the size and bits-per pixel parameters affect directly the memory size of the generated fonts as well. Make sure that these parameters are fine tuned before generating the project code, so that their impact in memory consumption will be as small as possible.

SECTION

6

Project Properties

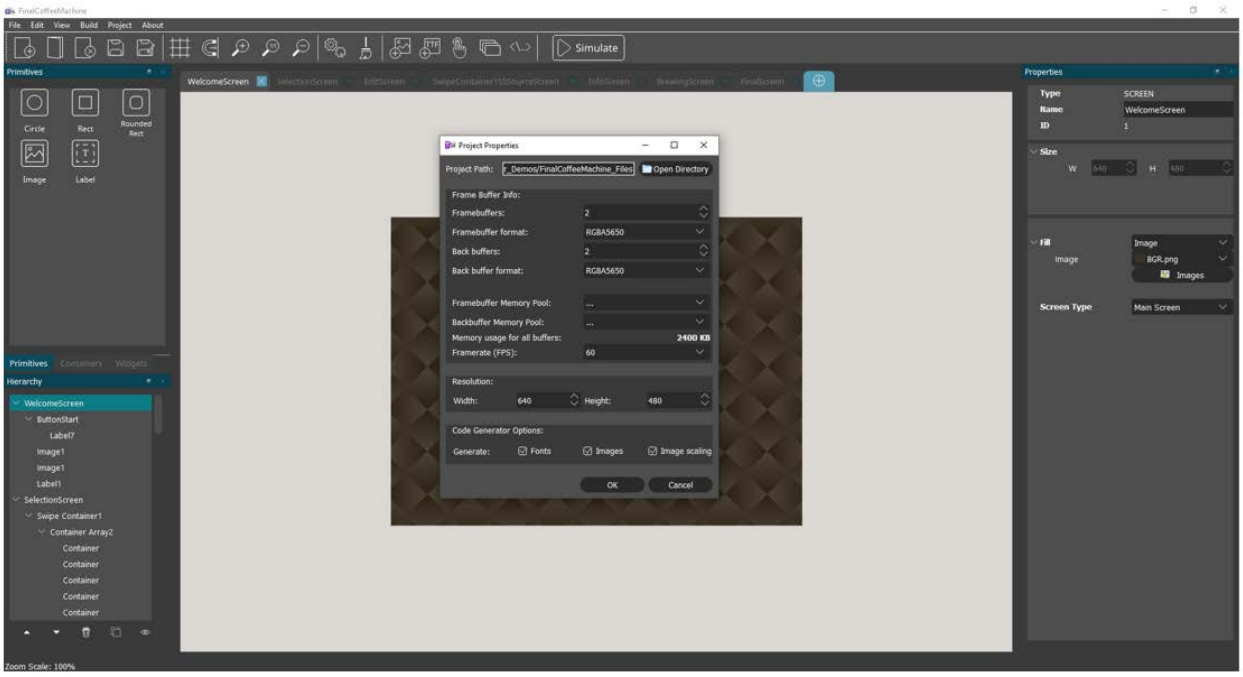
Each project created in Nema GUI Builder has some specific properties, that affect the code generation process as well as the generated code. These are:

- How many framebuffers will be used (single, double or triple buffering)
- The frame-buffer format (RGBA8888, RGBA5650, TSC4, TSC6)
- How many back buffers will be used (up to two). Two back buffers are necessary when performing animations such as screen transitions using a non linear animation effect. If the animation buffers are less than two, screen transitions will be performed using linear effect and show/hide animations will be performed instantly, without animation effect.
- The back-buffers format (RGBA8888, RGBA5650, TSC4, TSC6) To keep the memory usage to minimum, the default format of these buffers is TSC4.
- The memory pool that will be used for the framebuffers and the back buffers.
- The animations frame rate (this sets the animation timer period used in the generated code)
- The project's resolution and the code generator options (whether the code generator will generate the fonts, images and if the generated images will be scaled to the minimum possible resolution as identified by Nema GUI Builder).

In addition, the user can see the current project path and navigate to it by pressing the respective option (**Open Directory**).

Last but not least, the users can also adapt some settings relative to the code generation. These concern the generation of images and fonts and whether the images should be scaled or not. Image scaling aims to minimize the memory that the generated images will consume. In order to achieve this, a large image can be scaled down to the resolution of the graphic item that it is assigned. If the resolution of the imported image is smaller than the resolution of the respective graphics item, the original resolution is kept. The resolution of the generated images, can be inspected in the code generation report (generated along with a project).

Figure 6-1: Project Properties



SECTION

7

Screen Groups

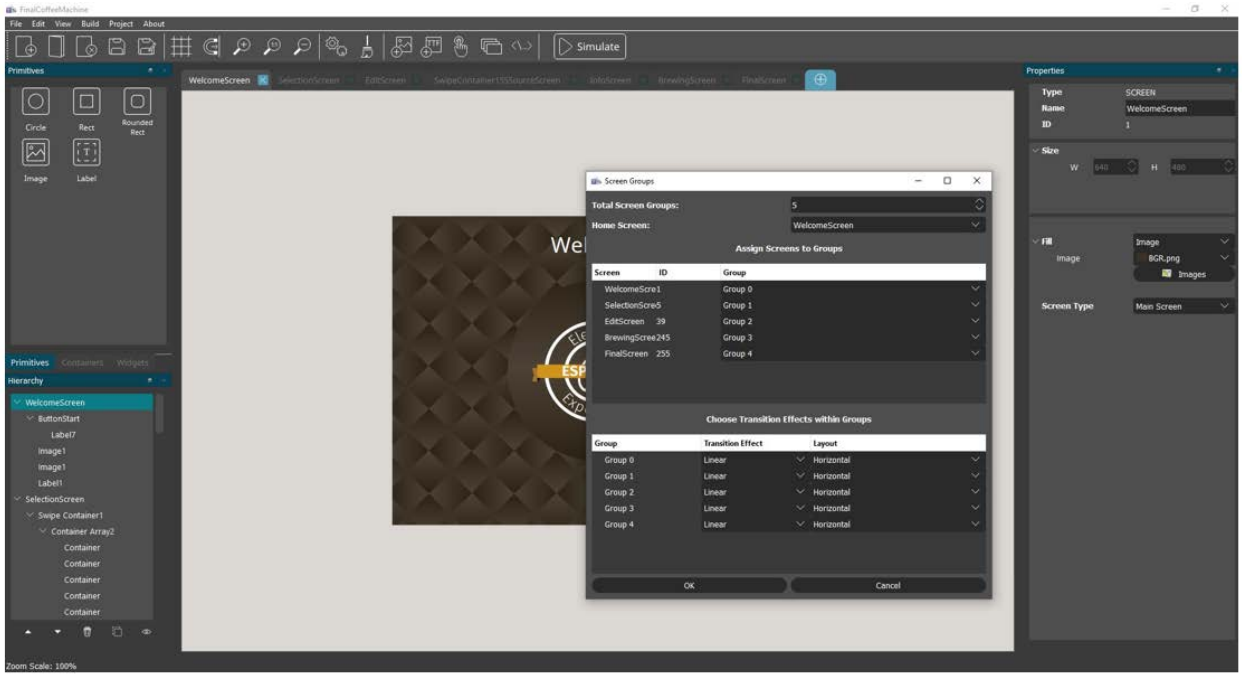
Modern applications usually consist of several sub-applications with similar but different user interfaces. For instance, complete smart-watch UI, can include graphical interfaces for sub-apps such as weather forecast, music player, make a call via a connected smartphone and many more.

Having this in mind, Nema GUI Builder organizes the designed screens of a project into one or several groups. By adopting this technique, screens are organized in an efficient way that makes the design of complex applications easier.

Each group has its own transition effect; effect during scene rendering while swiping from one screen to another. Swiping operations are allowed only for screens of the same group. In addition, screen groups have a layout. The layout can be either horizontal or vertical and it defines the orientation of swiping operations. For instance, when swiping a screen that belongs to a screen group with horizontal layout, the respective screen transition will be controlled by the cursor movement (mouse, touch point etc.) dx on the horizontal axis.

Changing the current screen group can be achieved by displaying a screen of a different group (e.g., by pressing a button). In this case the applications' current group will be set according to the target screen.

Figure 7-1: Screen Groups



SECTION

8

Event Manager

Nema GUI Builder utilizes the **Event Manager** for managing the events associated to a project. It can be found under the **Project** menu. This maximizes the user's interaction during runtime and allows them to inspect, add or remove events according to the needs of the project in a user-friendly way. Through the **Event Manager**, the user can add events by setting following parameters:

- The Trigger (e.g., a button is released)
- The Source item (if applicable)
- The Action: what should happen when the event is triggered

Depending on the **Action** that should be executed, the user can add more data to an event. For instance a **Screen Transition** action needs to know the duration and animation effect, while a **Set Value** action should know the value (absolute value or percentage) that will be set. These attributes are displayed when the user creates an event or inspects an existing one.

Many actions are predefined, nevertheless, the user can also create custom events and tailor their functionality to the project requirements. This can be performed by selecting a **Custom** action when creating a new event (or attaching a new action to an existing event). In the generated code, **Actions** are handled as callback functions. Therefore the user needs to complete these callback functions with the desired code. These functions can be found in the **custom_callbacks.c** file among the generated files.

For ease of use, custom actions are divided in four categories:

- One-Shot
- Periodic
- Transition
- Periodic Transition

The **One-Shot** callback function is executed instantly when the event is triggered.

The **Periodic** actions are performed in a periodic basis. For example, the callback function is executed periodically according to the defined period (e.g., a **Digital Meter** that needs to set its value every 10 seconds).

The **Transition** is a special case of custom action. This action can be used to change an attribute of the target item (e.g., opacity) in a continuous way. The **Transition** has a specific duration, defined by the user and during runtime, it keeps track of its progress (discussed in more detail in the next section).

The **Periodic Transitions** are essentially **Transitions** that are performed periodically. The user must define both the duration and the period of this action.

After a custom action has been created, this can be afterwards edited by changing its corresponding attributes (**Type**, **Duration** and **Period**) in the **Event Manager**.

Figure 8-1 and Figure 8-2 on page 31 depicts how the user can manage events in the **Event Manager**.

Figure 8-1: The Event Manager Form

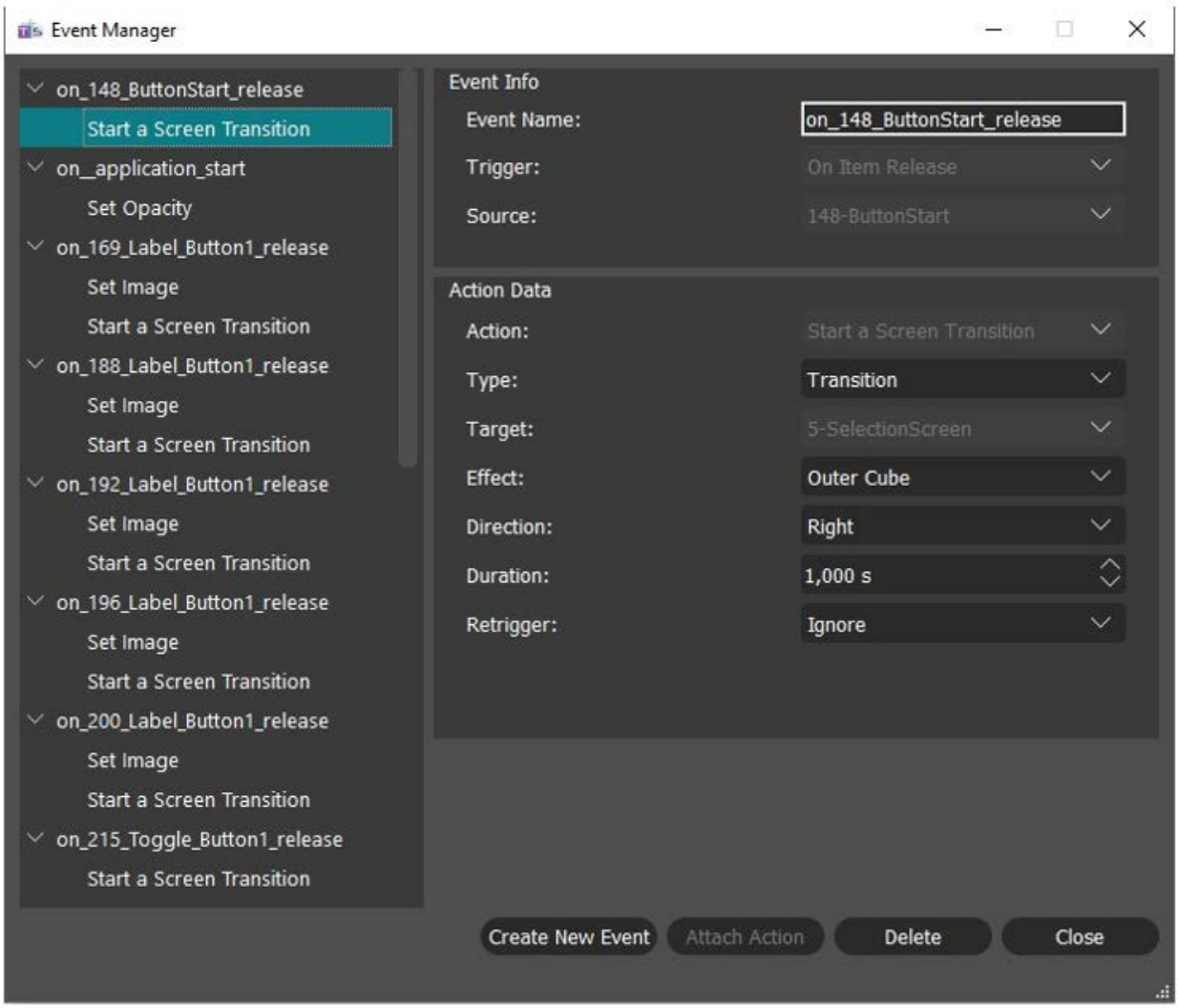
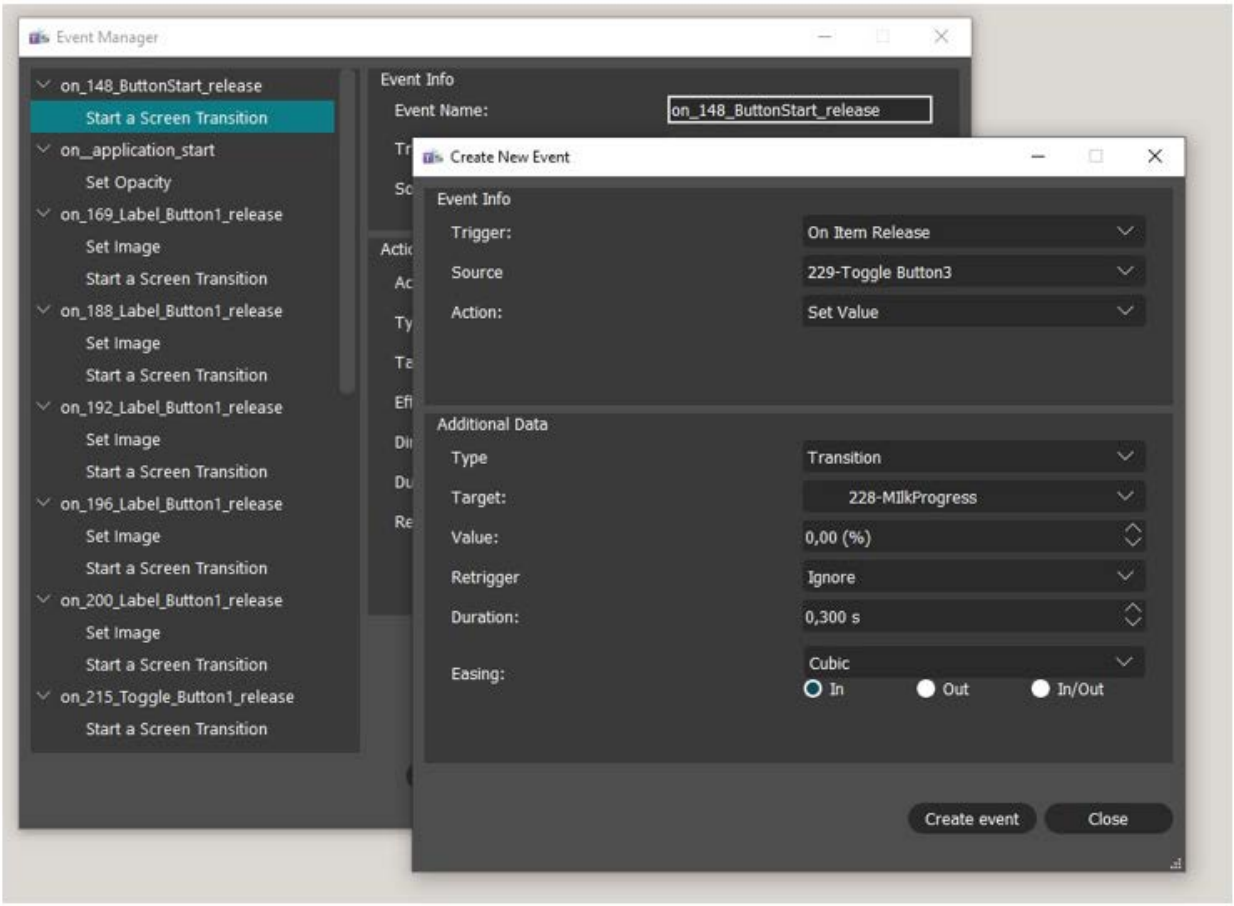


Figure 8-2: Adding a New Event



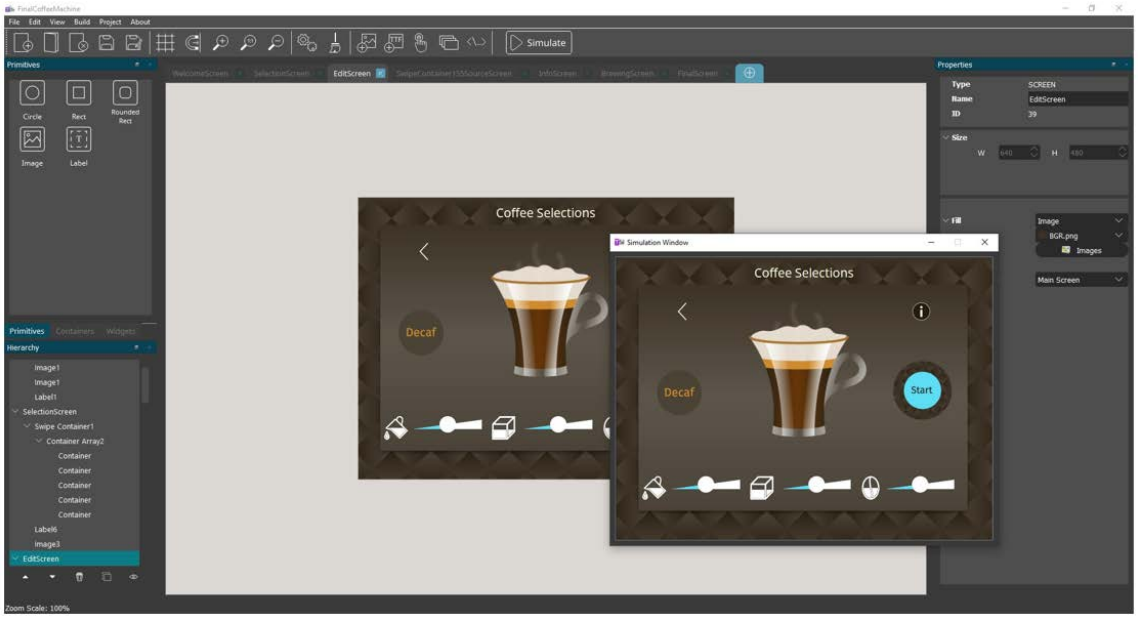
SECTION

9

Simulation Window

The current project can be simulated by selecting the Simulate option under the Build menu. This will invoke the simulation window (Figure 9-1) in which the user can simulate the project at its current status and observe whether its behavior is the desired one or not. The simulator supports all user interactions except the custom ones (custom events). Screen transitions, show/hide animation effects, scrollable graphics items and the rest of the visual features can be inspected at no time in the simulation window.

Figure 9-1: Simulation Window



The **Simulate** option has some limitations, but the generated code is not affected by them. These limitations are:

- The simulation window is always rendered in RBGA8888 color format, and thus the differences when using different image or framebuffer format cannot be observed there..

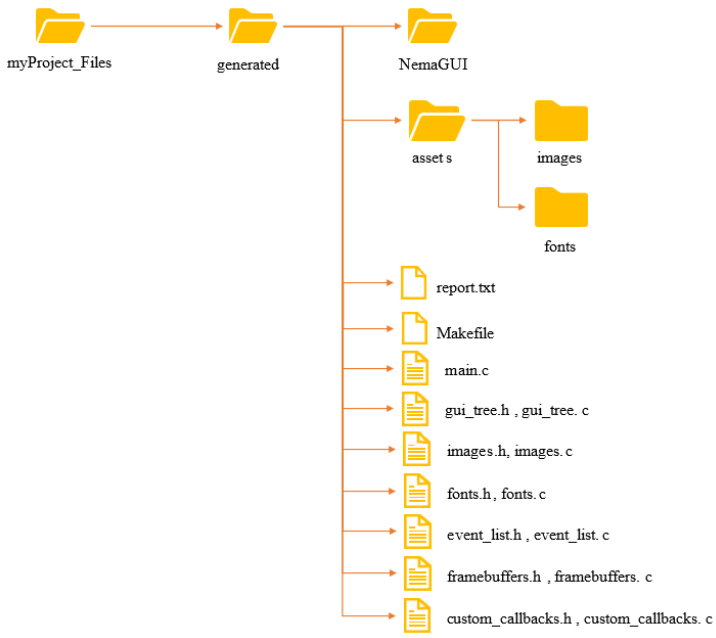
SECTION

10

Generated Code and Custom Callbacks

At code generation, a folder generated will be created inside the project's file directory. The generated files (C language) are located inside the generated folder. In this folder, one can find the generated assets (images and fonts) the NemaGUI folder that contains the API responsible for the runtime of the generated application. More specifically, it contains the header files of the necessary modules (e.g., main loop, gestures, screen transitions etc.) and data structs (graphics items data structs, events data structs etc.) that are necessary for an application in order to run on an embedded device. Besides the header files that expose the interface to the end user, NemaGUI API is provided as a library and the generated code is linked against the library, as indicated inside the generated Makefile (link against libNemaGUI). More details regarding NemaGUI API can be found in *Section 13 Nema GUI API Reference on page 40* and *Section 14 Widget References on page 114* that contain its documentation. In addition, project specific files are also generated inside the generated folder, as depicted in Figure 10-1.

Figure 10-1: Generated Code Directory Structure



Such files are:

- **report.txt,** contains information regarding the memory consumption of the generated images, fonts and framebuffers.
- **Makefile**, a default Makefile for building the generated code.
- **main.c**, the application entry point (initializes NemaGUI API, Nema GFX library and enters the application's main loop).
- **gui_tree.h** and **gui_tree.c**, contain the generated graphics items.
- **fonts.h** and **fonts.c**, contain the code that loads the generated fonts.
- **images.h** and **images.c**, contain the code that loads the generated images.
- **event_list.h** and **event_list.c**, contain the generated event list.
- **framebuffers.h** and **framebuffers.c,** contain the code that creates the framebuffer(s), back buffers (if available) memory objects along with the display layers.
- **custom_callbacks.h** and **custom_callbacks.c**, are the files that the user needs to fill in with custom code.

The **event_list.c** and **custom_callbacks.c** are the files related to the events the an application contains. More specifically, **event_list.c** contains the events that the Event Manager's contains, translated in C code.

The **custom_callbacks.c** is the file that the user needs to edit, in order to define the custom functionality. When the tool generates this file, it defines the respective callback function for each event that is associated to a custom Action.

These functions are named according to the name given by the user (when a custom action was created).

The user should fill the body of these functions and must not edit their names (as these names are used in the respective header file and the **event_list.c**)

By default, these functions contain a pointer to the related event **ng_event_base_t** *event and a void pointer void *data.

An example of a generated function is the following:

```
void my_custom_callback(ng_event_base_t* event, void *data){
ng_transition_t *transition = NG_TRANSITION(event);
}
```

In the above snippet, one can observe that the event pointer is casted to a **ng_transition_t** type. This is due to the fact that the callback is assigned to a transition event (the code generator casts the base event type to the desired type automatically). In this way, the user can now access the attributes of the **ng_transition_t** inside the function body. For instance the opacity of an object can be set based on the progress (transition attribute) of the event. The second argument is currently not used. For more information about the data structs used in the event mechanism, see *Section 13 Nema GUI API Reference on page 40*.

Events can be triggered using various triggers such as when the application starts-up, when a button is pressed, when a screen is entered and more. A special case of an event trigger is the custom trigger. When a custom trigger is used, it means that the event will be triggered manually by the user. To trigger an event, one needs to run its start function. The following snippet shows how an event can be manually triggered.

```
#include "ng_globals.h" //access the event list
void my_custom_trigger(){
        ng_event_base_t *my_event = NG_EVENT_LIST[8];
        my_event->start();
}
```

In the previous snippet, one can observe that the **ng_globals.h** must be included in the file that the custom trigger (start function) will be called. Including this file gives access to the necessary **NG_EVENT_LIST** array that contains the generated events. The developer needs to identify the index of the event that will be manually triggered by inspecting the generated event_list.c file. Once the index of the desired event is identified, the respective element of the event list can then be retrieved (the index in this example is 8) and its start function can then run.

SECTION

11

Examples

Nema GUI Builder comes along with four example projects that aim to work as "hello world" applications. They can be found inside the installation directory, in the examples folder. The goal of these examples is to familiarize the users with the Nema GUI Builder environment and the way it handles the GUI development process. They can be instantly simulated in order to evaluate their behavior at runtime. The examples are:

- **Animated screens**: Contains six screens that are divided into two screen groups. It depicts the functionality of swiping inside a screen group, along with how a transition from one group to another is performed (button-triggered transition).
- **Gauge**: This example contains a button that is connected to a gauge and a digital meter. Clicking the button will set the value of the gauge and the digital meter using a transition event.
- **Mixer**: The functionality and custom style (modified compared to the default one) of sliders is illustrated in this example. In addition, interactive (swipable) gauges are used in order to create rotating objects.
- **Coffee Machine**: A demo application that depicts how a coffee machine UI is designed in Nema GUI Builder.

In all the examples, the events used to perform their functionality can be inspected in the Event Manager. It is recommended not to modify these examples as their goal is to act as guides for GUI development projects that should be available at any time. Modifying these examples and saving the changes will overwrite them and their initial structure cannot be recovered.

SECTION

12

Project Deployments

The source code of a project created in Nema GUI Builder can be generated simply by pressing the **Generate** button. This will create the generated folder inside the project's directory and the source code will be placed there. The user can then modify the generated code, by completing the code of the custom callbacks (if applicable). The next step is to compile the project's source code for the target platform.

12.1 Project deployment on Linux PC

In order to compile and run the project on a Linux PC, the generated source code must be compiled and linked to the Nema GFX libraries. All the necessary files (libraries, header files) have been placed inside **<PATH_TO_Nema_GUI_Builder>/NemaGFX_SDK/**.

Inside the generated code's directory, there exists a Makefile that is going to be used for the creation of the executable file. Prior to running make, the following environment parameters (shell variables) have to be configured.

- PLATFORM
- NEMAGFX_SDK_PATH
- LD_LIBRARY_PATH

They can be set by exporting them inside the shell terminal from which the make command will be executed, as follows:

```
export PLATFORM=sw_linux_sdl
export NEMAGFX_SDK_PATH=<PATH_TO_Nema_GUI_Builder>/NemaGFX_SDK
exportLD_LIBRARY_PATH=<PATH_TO_Nema_GUI_Builder>/NemaGFX_SDK/lib
```

For instance, assuming that Nema GUI Builder is located inside the home directory on a Linux PC, these parameters can be set as:

```
export PLATFORM=sw_linux_sdl
export NEMAGFX_SDK_PATH=/home/Nema_GUI_Builder/NemaGFX_SDK export
LD_LIBRARY_PATH=/home/Nema_GUI_Builder/NemaGFX_SDK/lib/
```

The following figures, illustrate how the Gauge example (that comes along with Nema GUI Builder) is compiled and linked to **libNemaGFX.so**.

Figure 12-1: Setting Environment Variables

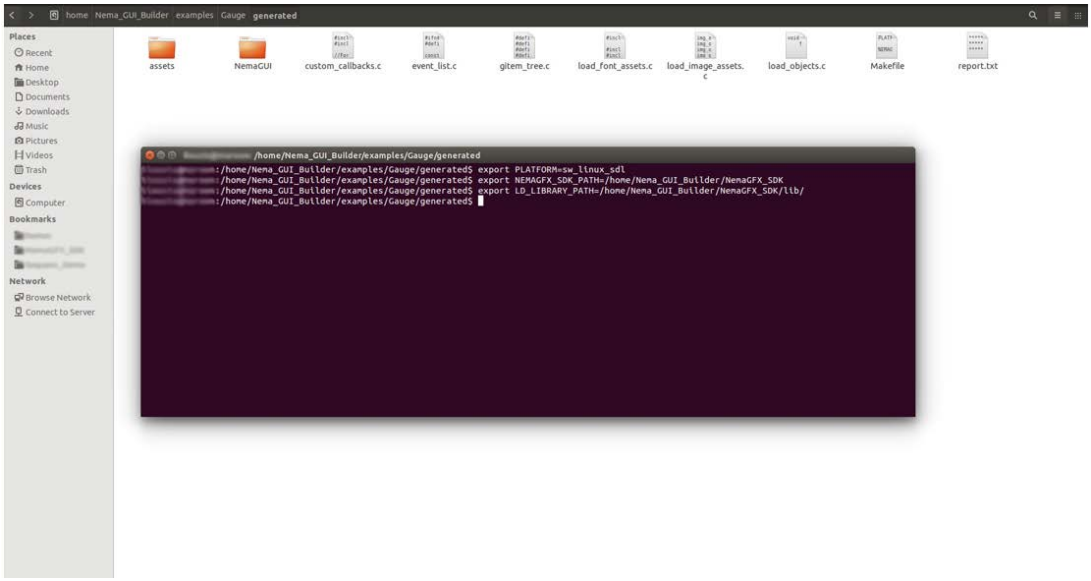


Figure 12-2: Running Make to Create Executable File

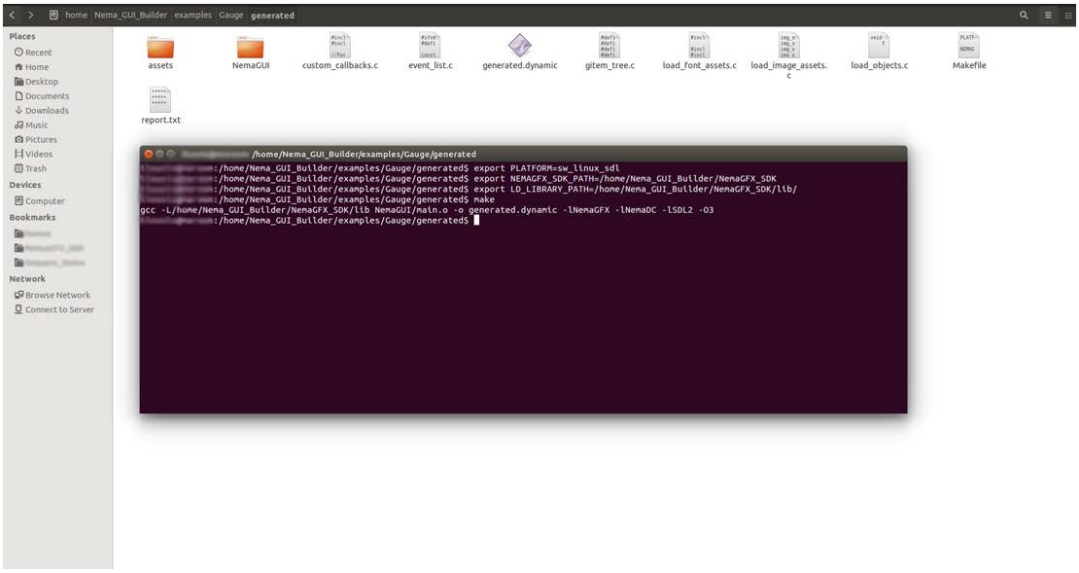
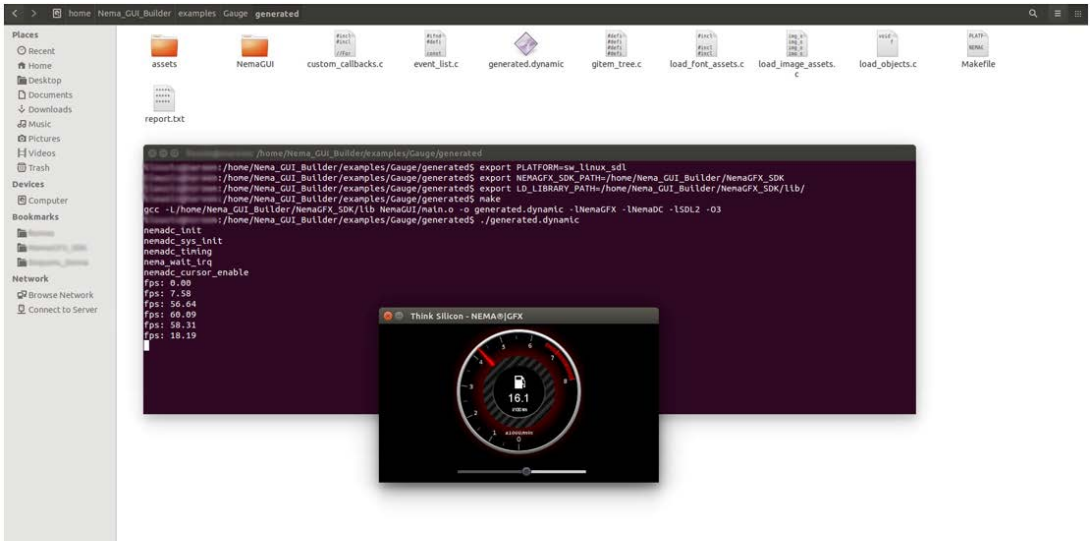


Figure 12-3: Running the Executable File



As it can be seen in Figure 12-3, running the executable file (generated.dynamic) will display the application window. The frames per second (fps) are displayed in the terminal window.

12.2 Dependencies

The application window utilizes the Simple Direct Media Layer 2 library (SDL2) (\url{https://www.libsdl.org/}), and therefore it is necessary that this library is installed on the host machine.

On an Ubuntu machine, the SDL2 library can be installed by executing the following command:

```
sudo apt-get install libsdl2-dev
```

12.3 Limitations

The software implementation of the Nema GFX API is designed for use in embedded systems. This indicates that, in its current version, a few features are not implemented, mainly for performance reasons. The following limitations apply to running Nema GUI Builder generated applications without hardware acceleration (Nema GPU):

- TSC4, TSC6 and TSC6a formats are not supported. Framebuffers, back-buffers and imported images should be configured in a different format, otherwise they will not be displayed at all.
- Bilinear filtering in textures is not supported. Generated textures that contain the bilinear filtering attribute are rendered using the point-sampling method.

SECTION

13

Nema GUI API Reference

This section provides an overview of the Nema GUI API which is the API for handling the run-time of a generated project. It contains the software modules and data structs that enable an application to run on an embedded device. Nema GUI consists of the graphics item's (widgets) data structs (described in the next section) along with the modules that handle their behavior and interactions. Nema GUI consumes approximately 25KB of memory with all its features included (all widgets, event types, animations etc.) By narrowing down the feature set used in an application, this amount can be further reduced.

13.1 Files

Here is a list of all files with brief descriptions:

13.1.1 ng_animation.h

Animations

```
#include "ng_typedefs.h"
#include "ng_tree.h"
#include "ng_gitem.h"
#include "nema_utils.h"
```

Data Structures

```
struct ng_animation_data_t
```
More...

Macros

```
#define NG_DIRECTION_LEFT
#define NG_DIRECTION_RIGHT
#define NG_DIRECTION_TOP
#define NG_DIRECTION_BOTTOM
#define NG_SHOW
```

```
#define NG_HIDE
#define NG_ANIMATION_DATA
```

13.1.1.1 Functions

```
void ng_animation_callback(ng_event_base_t *event, void *data)
```

Callback function executed continuously to update an animation.

```
void ng_animation_draw(tree_node_t *node, int x_min, int y_min,
int x_max, int y_max)
```

Draws the animations back buffer (if available) in the framebuffer.

```
bool ng_animation_init(ng_animation_data_t *data)
```

Draws an animated tree node (along with its children) inside a back buffer (if available)

```
void ng_animation_fade(ng_animation_data_t *data)
```

Fade animation function.

```
void ng_animation_fly(ng_animation_data_t *data)
```

Fly animation function.

```
void ng_animation_fade_zoom(ng_animation_data_t *data)
```

Fade-zoom animation function.

```
void ng_animation_cube_face(ng_animation_data_t *data)
```

Cube face animation function.

```
void ng_animation_flip(ng_animation_data_t *data)
```

Flip animation function.

13.1.1.2 Detailed Description

Animations

This files includes the data needed for the implementation of animations (show/hide). Such animations are performed using back buffers. If there are no back buffers available, show and hide effects will be applied instantly.

13.1.1.3 Macro Definition Documentation

```
#define NG_ANIMATION_DATA
```

Type caster for casting a void pointer to **ng_animation_data_t** pointer struct

```
#define NG_DIRECTION_BOTTOM
```

Animation moves to the bottom

#define NG_DIRECTION_LEFT

Animation moves to the left

#define NG_DIRECTION_RIGHT

Animation moves to the right

#define NG_DIRECTION_TOP

Animation moves to the top

#define NG_HIDE

Animation should perform a "hide"

#define NG_SHOW

Animation should perform a "show"

13.1.1.4 Function Documentation

void ng_animation_callback (ng_event_base_t * event, void * data)

Callback function executed continuously to update an animation.

Parameter	Description
*data	Pointer to **ng_animation_data_t** data struct tuple (type casting from void is performed internally)

Return

void

void ng_animation_cube_face (ng_animation_data_t * data)

Cube face animation function.

Parameter	Description
*data	Pointer to the animation data

Return

void

void ng_animation_draw (tree_node_t * node, int x_min, int y_min, int x_max, int y_max)

Draws the animations back buffer (if available) in the framebuffer.

Parameter	Description
*node	Pointer to the animated tree node
x_min	minimum x position
y_min	minimum y position
x_max	maximum x position
y_max	maximum y position

Return

void

```
void ng_animation_fade (  ng_animation_data_t * data )
```

Fade animation function.

Parameter	Description
*data	Pointer to the animation data

Return

void

```
void ng_animation_fade_zoom (  ng_animation_data_t * data )
```

Fade-zoom animation function.

Parameter	Description
*data	Pointer to the animation data

Return

void

```
void ng_animation_flip (  ng_animation_data_t * data )
```

Flip animation function.

Parameter	Description
*data	Pointer to the animation data

Return

void

```
void ng_animation_fly (  ng_animation_data_t * data )
```

Fly animation function.

Parameter	Description
*data	Pointer to the animation data

Return

void

```
bool ng_animation_init (  ng_animation_data_t * data )
```

Draws an animated tree node (along with its children) inside a back buffer (if available).

Parameter	Description
*data	Pointer to the animation data

Return

true if the drawing was performed inside the back buffer, otherwise false

13.1.2 ng_callbacks.h

Callback functions.

```
#include "ng_event.h"
```

13.1.2.1 Functions

```
void ng_animate_uint32(ng_event_base_t *event, void *data)
```

Animates a **uint32_t** variable from an initial value to a final value using an easing function.

```
void ng_animate_float(ng_event_base_t *event, void *data)
```

Animates a float variable from an initial value to a final value using an easing function.

```
void ng_set_uint32(ng_event_base_t *event, void *data)
```

Sets **uint32_t** variable.

```
void ng_set_float(ng_event_base_t *event, void *data)
```

Sets float variable.

```
void ng_set_ptr(ng_event_base_t *event, void *data)
```

Sets void pointer.

`void ng_update_gitem(ng_event_base_t *event, void *data)`

Utility callback for updating a gitem.

`void ng_set_int_int(ng_event_base_t *event, void *data)`

Sets two integer variables e.g., (w, h)

`void ng_animate_int_int_pair(ng_event_base_t *event, void *data)`

Animates two integer variables e.g., (w, h) from an initial value to a final value using an easing function.

`void ng_animate_int_int(ng_event_base_t *event, void *data)`

Animates an integer variable from an initial value to a final value using an easing function.

`void ng_set_tree_node(ng_event_base_t *event, void *data)`

Sets a **tree_node_t** pointer.

`void ng_set_node_to_node(ng_event_base_t *event, void *data)`

Sets a **tree_node_t** pointer to another tree node.

`void ng_set_percent(ng_event_base_t *event, void *data)`

Sets a percentage value [0.f, 1.f].

13.1.2.2 Detailed Description

Callback functions.

Generic callback functions are used by the event mechanism, for setting or animating various parameters (e.g., position, color, numerical value etc.) Any callback function must have the signature **void function_name(ng_event_base_t *event, void *data)**. The void *data argument should then be casted inside the callback function body to an appropriate data type

13.1.2.3 Function Documentation

`void ng_animate_float ( ng_event_base_t * event, void * data )`

Animates a float variable from an initial value to a final value using an easing function.

Parameter	Description
*data	Pointer to **ng_git_float_float_ez_t** data struct tuple (type casting from void is performed internally)

Return

void

```
void ng_animate_int_int (  ng_event_base_t * event,  void * data )
```

Animates an integer variable from an initial value to a final value using an easing function.

Parameter	Description
*data	Pointer to **ng_git_int_int_ez_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_animate_int_int_pair (  ng_event_base_t * event,  void * data
)
```

Animates two integer variables e.g., (w, h) from an initial value to a final value using an easing function.

Parameter	Description
*data	Pointer to **ng_git_int_int_pair_ez_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_animate_uint32 (  ng_event_base_t * event,  void * data )
```

Animates a **uint32_t** variable from an initial value to a final value using an easing function.

Parameter	Description
*data	Pointer to **ng_git_uint32_uint32_ez_t** data struct tuple (type casting from void is performed internally)

Return

void

```
void ng_set_float (  ng_event_base_t * event,  void * data )
```

Sets float variable.

Parameter	Description
*data	Pointer to **ng_git_float_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_int_int (  ng_event_base_t * event,  void * data )
```

Sets two integer variables e.g., (w, h).

Parameter	Description
*data	Pointer to **ng_git_int_int_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_node_to_node (  ng_event_base_t * event,  void * data )
```

Sets a **tree_node_t** pointer to another tree node.

Parameter	Description
*data	Pointer to **ng_node_node_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_percent (  ng_event_base_t * event,  void * data )
```

Sets a percentage value [0.f, 1.f].

Parameter	Description
*data	Pointer to **ng_git_float_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_ptr (  ng_event_base_t * event,  void * data )
```

Sets void pointer.

Parameter	Description
*data	Pointer to **ng_git_ptr_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_tree_node (  ng_event_base_t * event,  void * data )
```

Sets a **tree_node_t** pointer.

Parameter	Description
*data	Pointer to **ng_tree_node_ptr_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_set_uint32 (  ng_event_base_t * event,  void * data )
```

Sets uint32_t variable.

Parameter	Description
*data	Pointer to **ng_git_uint32_t** tuple (type casting from void is performed internally)

Return

void

```
void ng_update_gitem (  ng_event_base_t * event,  void * data )
```

Utility callback for updating a gitem.

Parameter	Description
*data	Pointer to **ng_gitptr_t** tuple (type casting from void is performed internally)

Return

void

13.1.3 ng_display.h

Display

```
#include "nema_core.h"
#include "nema_transitions.h"
#include "ng_gitem.h" #include "ng_tree.h"
```

Macros

```
#define DISPLAY_SCREEN
#define DISPLAY_SCREEN_TRANSITION
#define DISPLAY_POPUP
```

13.1.3.1 Functions

`void ng_display_screen_node_to_fb(img_obj_t *fb_img, tree_node_t *screen_node, int x_off, int y_off)`

Draws a screen tree node (screen gitem along with its children) to a designated buffer.

`void ng_display_screen_clear(int wait)`

Clears the current framebuffer.

`void ng_display_bind_transition_buffers(void)`

Binds the transition buffers (if available) to the GPU.

`void ng_display(void)`

Updates the display (redraws current screen)

`void ng_display_init(void)`

Initializes the display module (framebuffer(s), command lists)

`void ng_display_set_event(ng_event_base_t *event)`

Sets the event used for performing a screen transition.

`void ng_display_set_mode(int mode)`

Sets the mode of the display.

`int ng_display_get_mode()`

Gets the current display mode.

`void ng_display_set_popup(tree_node_t *node)`

Sets the pop-up tree node to be diplayed along with the display mode (DISPLAY_POPUP)

`void ng_display_set_clear(bool clear)`

Controls if the display should perform a "clear screen" before updating it.

`bool ng_back_buffer_is_locked(int index)`

Checks if the buckbuffer "index" is locked (currently not available)

`void ng_back_buffer_lock(int index)`

Locks the back buffer "index".

`void ng_back_buffer_unlock(int index)`

Unlocks the back buffer "index".

13.1.3.2 Detailed Description

Display

The display module, provides functions for updating the framebuffer according to the current context. This can either be displaying a simple screen, a screen transition or a pop-up window on top of a main screen.

13.1.3.3 Macro Definition Documentation

`#define DISPLAY_POPUP`

Display mode for a pop-up

`#define DISPLAY_SCREEN`

Display mode for displaying a screen (default mode)

`#define DISPLAY_SCREEN_TRANSITION`

Display mode when performing a screen transition

13.1.3.4 Function Documentation

`bool ng_back_buffer_is_locked ( int index )`

Checks if the buckbuffer "index" is locked (currently not available).

Parameter	Description
index	Index of the back buffer to check

Return

bool True if the back buffer is locked, otherwise false.

`void ng_back_buffer_lock ( int index )`

Locks the back buffer "index".

Parameter	Description
index	Index of the back buffer to lock

```
void ng_back_buffer_unlock (  int index )
```

Unlocks the back buffer "index".

Parameter	Description
index	Index of the buck buffer to unlock

```
int ng_display_get_mode
```

Gets the current display mode.

Return

int This should be **DISPLAY_SCREEN**, **DISPLAY_SCREEN_TRANSITION** or **DISPLAY_POPUP**.

```
void ng_display_screen_clear (  int wait )
```

Clears the current framebuffer.

Parameter	Description
wait	if this is equal to zero, the command list is submitted without waiting for an interrupt

```
void ng_display_screen_node_to_fb (  img_obj_t * fb_img,  tree_node_t
* screen_node,  int x_off,  int y_off )
```

Draws a screen tree node (screen gitem along with its children) to a designated buffer.

Parameter	Description
*fb_img	Pointer to the designated buffer
*screen_node	Pointer of the screen tree node that needs to be drawn
x_off	x offset
y_off	y offset

```
void ng_display_set_clear (  bool clear )
```

Controls if the display should perform a "clear screen" before updating it.

Parameter	Description
clear	If true, the display will clear its current content before updating it. Otherwise it will draw the new content on top of the old one

```
void ng_display_set_event (  ng_event_base_t * event )
```

Sets the event used for performing a screen transition.

Parameter	Description
event	Pointer to the screen transition event data struct (casted to **ng_transition_t** struct internally)

```
void ng_display_set_mode (  int mode )
```

Sets the mode of the display.

Parameter	Description
mode	This can eitherbe **DISPLAY_SCREEN**, **DISPLAY_SCREEN_TRANSITION** or **DISPLAY_POPUP**

```
void ng_display_set_popup (  tree_node_t * node )
```

Sets the pop-up tree node to be displayed along with the display mode (**DISPLAY_POPUP**)

Parameter	Description
node	pop-up node to be displayed

13.1.4 ng_draw.h

High level drawing functions for performing hierarchical drawing of tree nodes (along with their children)

```
#include "ng_gitem.h" #include "ng_tree.h"
```

13.1.4.1 Functions

```
void ng_draw_tree_node(tree_node_t *node, int x_off, int y_off,
int x_min, int y_min, int x_max, int y_max)
```

Draws a specific tree node (its graphics item) in the framebuffer.

void ng_draw_tree(tree_node_t *node, int x_off, int y_off, int x_min, int y_min, int x_max, int y_max)

Recursive function, similar to **ng_draw_tree_node** but draws recursively every child tree node of the initial node.

void ng_draw_to_buffer(tree_node_t *node, int x_off, int y_off, int x_min, int y_min, int x_max, int y_max)

Draws a tree node and its children nodes inside the bound framebuffer (used by the animations module)

13.1.4.2 Detailed Description

High level drawing functions for performing hierarchical drawing of tree nodes (along with their children)

13.1.4.3 Function Documentation

void ng_draw_to_buffer (tree_node_t * node, int x_off, int y_off, int x_min, int y_min, int x_max, int y_max)

Draws a tree node and its children nodes inside the bound framebuffer (used by the animations module)

Parameter	Description
node	The initial tree node whose graphics item needs to be drawn
x_off	Horizontal offset whithin the framebuffer (absolute coordinate)
y_off	Vertical offset whithin the framebuffer (absolute coordinate)
x_min	Minimum x coordinate (needed for clipping)
y_min	Minimum y coordinate (needed for clipping)
x_max	Maximum x coordinate (needed for clipping)
y_max	Maximum y coordinate (needed for clipping)

void ng_draw_tree (tree_node_t * node, int x_off, int y_off, int x_min, int y_min, int x_max, int y_max)

Recursive function, similar to **ng_draw_tree_node** but draws recursively every child tree node of the initial node.

Parameter	Description
node	The initial tree node whose graphics item needs to be drawn (usually a tree node that contains a screen)
x_off	Horizontal offset within the framebuffer (absolute coordinate)
y_off	Vertical offset within the framebuffer (absolute coordinate)
x_min	Minimum x coordinate (needed for clipping)
y_min	Minimum y coordinate (needed for clipping)
x_max	Maximum x coordinate (needed for clipping)
y_max	Maximum y coordinate (needed for clipping)

```
void ng_draw_tree_node ( tree_node_t * node, int x_off, int y_off,
int x_min, int y_min, int x_max, int y_max )
```

Draws a specific tree node (its graphics item) in the framebuffer.

Parameter	Description
node	The tree node whose graphics item needs to be drawn
x_off	Horizontal offset within the framebuffer (absolute coordinate)
y_off	Vertical offset within the framebuffer (absolute coordinate)
x_min	Minimum x coordinate (needed for clipping)
y_min	Minimum y coordinate (needed for clipping)
x_max	Maximum x coordinate (needed for clipping)
y_max	Maximum y coordinate (needed for clipping)

13.1.5 ng_draw_prim.h

Primitive drawing (rectangle, circle, rounded rectangle, quadrilateral and image)

```
#include "nema_sys_defs.h"
```

13.1.5.1 Functions

```
void ng_fill_rect(int x, int y, int w, int h, uint32_t color, int
override_blend)
```

Fills a rectangular area with a color (that can contain opacity)

void ng_draw_primitive_rect(int x, int y, int w, int h, uint32_t color, int pen_width, int override_blend)

Draws a rectangle (outline) with a certain color (that can contain opacity) and pen width.

void ng_draw_primitive_rounded_rect(int x, int y, int w, int h, uint32_t color, int radius, int override_blend)

Draws a rounded rectangle with a certain color (that can contain opacity) and radius.

void ng_fill_primitive_rounded_rect(int x, int y, int w, int h, uint32_t color, int radius, int override_blend)

Fills a rounded rectangle with a certain color (that can contain opacity) and radius.

void ng_draw_primitive_circle(int x, int y, int r, uint32_t color, int override_blend)

Draws a circle with a specific color (that contains the opacity) and radius.

void ng_fill_primitive_circle(int x, int y, int r, uint32_t color, int override_blend)

Fills a circle with a specific color (that can contain opacity) and radius.

void ng_blit_rect_fit(img_obj_t *img, int x, int y, int w, int h, int override_blend, uint8_t opacity)

Blits a source image by fitting it into a rectangular area.

void ng_blit_quad_fit(img_obj_t *img, float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, int override_blend, uint8_t opacity)

Blits a source image by fitting it into a quadrilateral area.

void ng_fill_quad(float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, int override_blend, uint32_t color)

Fills a quadrilateral area with a certain color (that can contain opacity)

13.1.5.2 Detailed Description

Primitive drawing (rectangle, circle, rounded rectangle, quadrilateral and image) Drawing functions used commonly used across several graphics items (by their drawing functions)

13.1.5.3 Function Documentation

```
void ng_blit_quad_fit ( img_obj_t * img, float x0, float y0, float
x1, float y1, float x2, float y2, float x3, float y3, int over-
ride_blend, uint8_t opacity )
```

Blits a source image by fitting it into a quadrilateral area.

Parameter	Description
img	Pointer to the source image
x0	x position of the first vertex of the quadrilateral
y0	y position of the first vertex of the quadrilateral
x1	x position of the second vertex of the quadrilateral
y1	y position of the second vertex of the quadrilateral
x2	x position of the third vertex of the quadrilateral
y2	y position of the third vertex of the quadrilateral
x3	x position of the fourth vertex of the quadrilateral
y3	y position of the fourth vertex of the quadrilateral
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function
opacity	Opacity [0, 255]

```
void ng_blit_rect_fit ( img_obj_t * img, int x, int y, int w, int
h, int override_blend, uint8_t opacity )
```

Blits a source image by fitting it into a rectangular area.

Parameter	Description
img	Pointer to the source image
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
w	Rectangle's width
h	Rectangle's height

Parameter	Description
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function
opacity	Opacity [0, 255]

```
void ng_draw_primitive_circle ( int x, int y, int r, uint32_t
color, int override_blend )
```

Draws a circle with a specific color (that contains the opacity) and radius.

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
r	Radius
color	Circle's color
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

```
void ng_draw_primitive_rect ( int x, int y, int w, int h,
uint32_t color, int pen_width, int override_blend )
```

Draws a rectangle (outline) with a certain color (that can contain opacity) and pen width.

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
w	Rectangle's width
h	Rectangle's height
color	Rectangle's color
pen_width	Pen width
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

```
void ng_draw_primitive_rounded_rect ( int x, int y, int w, int h,
uint32_t color, int radius, int override_blend )
```

Draws a rounded rectangle with a certain color (that can contain opacity) and radius.

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
w	Rectangle's width
h	Rectangle's height
color	Rectangle's color
radius	Radius
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

```
void ng_fill_primitive_circle ( int x, int y, int r, uint32_t
color, int override_blend )
```

Fills a circle with a specific color (that can contain opacity) and radius.

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
r	Radius
color	Circle's color
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

```
void ng_fill_primitive_rounded_rect ( int x, int y, int w, int h,
uint32_t color, int radius, int override_blend )
```

Fills a rounded rectangle with a certain color (that can contain opacity) and radius.

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
w	Rectangle's width
h	Rectangle's height
color	Rectangle's color
radius	Radius
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

```
void ng_fill_quad ( float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, int override_blend, uint32_t color )
```

Fills a quadrilateral area with a cirtain color (that can contain opacity)

Parameter	Description
x0	x position of the first vertex of the quadrilateral
y0	y position of the first vertex of the quadrilateral
x1	x position of the second vertex of the quadrilateral
y1	y position of the second vertex of the quadrilateral
x2	x position of the third vertex of the quadrilateral
y2	y position of the third vertex of the quadrilateral
x3	x position of the fourth vertex of the quadrilateral
y3	y position of the fourth vertex of the quadrilateral
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function
color	Color (contains opacity information)

```
void ng_fill_rect ( int x, int y, int w, int h, uint32_t color, int override_blend )
```

Fills a rectangular area with a color (that can contain opacity)

Parameter	Description
x	Absolute x position inside the framebuffer
y	Absolute y position inside the framebuffer
w	Rectangle's width
h	Rectangle's height
color	Rectangle's color
override_blend	If zero, the parameters needed for blending (blending mode, const color, texture color) are implicitly calculated within this function, otherwise they need to be configured manually before calling this function

13.1.6 ng_event.h

```
#include "ng_gitem.h"
#include "ng_typedefs.h"
#include "ng_animation.h"
```

Data Structures

`union ng_act_ptr`

Union that groups together pointers to all possible actions that are supported in the event mechanism. More...

`struct __ng_event_base_t`

Event base struct definition. More...

Macros

```
#define EV_TRIGGER_NULL
#define EV_TRIGGER_PRESS
#define EV_TRIGGER_RELEASE
#define EV_TRIGGER_HOLD
#define EV_TRIGGER_DRAG
#define EV_TRIGGER_VALUE_CHANGED
#define EV_TRIGGER_STATE_CHANGED
#define EV_TRIGGER_SCREEN_ENTERED
#define EV_TRIGGER_APP_START
#define EV_TRIGGER_CUSTOM
#define EV_TRIGGER_TIMER
#define EV_RETRIGGER_IGNORE
#define EV_RETRIGGER_PAUSE
#define EV_RETRIGGER_RESUME
#define EV_RETRIGGER_PAUSE_TOGGLE
#define EV_RETRIGGER_REVERSE
#define EV_RETRIGGER_RESET
#define EV_RETRIGGER_FINISH
```

```
#define EV_RETRIGGER_RESTART
#define EV_RETRIGGER_STOP
#define EV_STATUS_STOPPED
#define EV_STATUS_RUNNING
#define EV_STATUS_PAUSED
#define EV_STATUS_MASK
#define EV_STATUS_REVERSED
#define EV_STATUS_FIRST_RUN
#define EV_STATUS_LAST_RUN
#define EVENT_BASE_STRUCT
```

Define that contains all the **ng_event_base_t** data struct attributes that are common among all different event types (oneshot, periodic, transition, periodic transition)

```
#define ng_event_is_stopped
#define ng_event_is_running
#define ng_event_is_paused
#define NG_EVENT
#define PROGRESS
#define DURATION
#define PERIOD
#define START_TIME
#define NG_CALLBACK_DATA
```

13.1.6.1 Typedefs

```
typedef void(*  act_gitptr_float_f )(struct _gitem_base_t *gitem,
float value)typedef
```

```
void(* act_gitptr_float_f) (struct _gitem_base_t *gitem, float
value) More...
```

```
typedef void(*  act_gitptr_ptr_f )(struct _gitem_base_t *gitem,
void *ptr)typedef void(* act_gitptr_ptr_f) (struct _gitem_base_t
*gitem, void *ptr) More...
```

```
typedef void(*  act_gitptr_int_f )(struct _gitem_base_t *gitem,
int value)typedef void(* act_gitptr_int_f) (struct _gitem_base_t
*gitem, int value) More...
```

```
typedef void(*  act_gitptr_uint_f )(struct _gitem_base_t *gitem,
uint32_t value)typedef
```

```
void(* act_gitptr_uint_f) (struct _gitem_base_t *gitem, uint32_t
value) More...
```

```
typedef void(*  act_gitptr_f )(struct _gitem_base_t
*gitem)typedef void(* act_gitptr_f) (struct _gitem_base_t *gitem)
More...
```

```
typedef void(*  act_gitptr_int_int_f )(struct _gitem_base_t
*gitem, int a, int b)typedef
```

```
void(* act_gitptr_int_int_f) (struct _gitem_base_t *gitem, int a,
int b) More...
```

```
typedef void(*  act_nodeptr_f )(struct _tree_node_t *node)typedef
void(* act_nodeptr_f) (struct _tree_node_t *node) More...

typedef void(*  act_nodeptr_nodeptr_f )(struct _tree_node_t
*node0, struct _tree_node_t *node1)typedef void(*
act_nodeptr_nodeptr_f) (struct _tree_node_t *node0, struct
_tree_node_t *node1) More...

typedef void(*  act_animptr_f )(ng_animation_data_t *data)typedef
void(*

act_animptr_f) (ng_animation_data_t *data) More...

typedef void(*  act_void_f )(void)typedef void(* act_void_f)
(void) More...

typedef void(*  handler_f )(struct _ng_event_base_t *event,
uint32_t trigger)typedef

void(* handler_f) (struct _ng_event_base_t *event, uint32_t
trigger) More...

typedef void(*  start_f )(struct _ng_event_base_t *event)typedef
void(* start_f) (struct _ng_event_base_t *event) More...

typedef void(*  stop_f )(struct _ng_event_base_t *event, bool
force_finish)typedef

void(* stop_f) (struct _ng_event_base_t *event, bool
force_finish) More...

typedef void(*  pause_toggle_f )(struct _ng_event_base_t *event,
bool pause)typedef

void(* pause_toggle_f) (struct _ng_event_base_t *event, bool
pause) More...
```

13.1.6.2 Functions

```
void ng_event_init(void)
```

Assigns events to graphics items (as generated) and creates the application's timer.

```
void ng_event_handle(ng_event_base_t *event, uint32_t
trigger_event)
```

Handles an event according to the trigger that triggered it.

```
void ng_event_run_callback(ng_event_base_t *event, int
status_flags)
```

Runs the callback function of the event.

```
void ng_event_set_status(ng_event_base_t *event, uint32_t status)
```

Set the status (stopped, running, paused) of an event.

```
bool ng_event_check_retrigger_flag(ng_event_base_t *event, int
flag)
```

Checks if a retrigger flag of an event is set or not.

```
bool ng_event_check_status_flag(ng_event_base_t *event, int flag)
```

Checks the status of an event.

```
void ng_event_set_status_flag(ng_event_base_t *event, int flag)
```

Sets a status flag of an event.

```
void ng_event_unset_status_flag(ng_event_base_t *event, int flag)
```

Unset a status flag of an event.

```
void ng_event_flip_status_flag(ng_event_base_t *event, int flag)
```

Flips (inverts) a status flag of an event.

13.1.6.3 Macro Definition Documentation

```
#define DURATION
```

Readability helper

```
#define EVENT_BASE_STRUCT
```

Define that contains all the **ng_event_base_t** data struct attributes that are common among all different event types (oneshot, periodic, transition, periodic transition) These attributes are:

uint32_t trigger: Pointer to an event assigned to the graphics item

uint32_t re-trigger: Pointer to the draw function

gitem_base_t *src_gitem: Pointer to the source graphics item **callback_f** callback: Function pointer to callback function union **ng_act_ptr** (*action): Function pointer to action function void ***action_data**: Pointer to the action's data

int affected_screen_id: ID of the screen that is affected by the event

uint32_t status: Event status

handler_f handler: Function pointer the event handler function **start_f** start: Function pointer to the event start function **stop_f** stop: Function pointer to the evetn stop function

pause_toggle_f pause_toggle: Function pointer to the event pause-toggle function **ng_event_base_t** *next: Pointer to the next event (used by graphics items that accept multiple events)

```
#define EV_RETRIGGER_FINISH
```

Go to final state (transition)

```
#define EV_RETRIGGER_IGNORE
```

Ignore the re-trigger (continue execution normally)

#define EV_RETRIGGER_PAUSE

Pause a running event

#define EV_RETRIGGER_PAUSE_TOGGLE

Ignore the re-trigger (continue execution normally)

#define EV_RETRIGGER_RESET

Reset the event to its initial state

#define EV_RETRIGGER_RESTART

On re-trigger, restart periodic/transitional event

#define EV_RETRIGGER_RESUME

Resume a paused event

#define EV_RETRIGGER_REVERSE

Return to the initial state by running the event in reverse order (**cur_progress** to 0.f)

#define EV_RETRIGGER_STOP

Go to final state and stop (periodic transition)

#define EV_STATUS_FIRST_RUN

Indicates the first run of the event

#define EV_STATUS_LAST_RUN

Indicates the last run of the event

#define EV_STATUS_MASK

Helper mask

#define EV_STATUS_PAUSED

The event is paused

#define EV_STATUS_REVERSED

The event is performing a backwards transition (from progress: p1 to 0)

#define EV_STATUS_RUNNING

The event is running

#define EV_STATUS_STOPPED

The event is stopped

#define EV_TRIGGER_APP_START

Application start-up trigger

#define EV_TRIGGER_CUSTOM

Custom trigger

#define EV_TRIGGER_DRAG

Drag (swipe) trigger

#define EV_TRIGGER_HOLD

Hold trigger (reserved for future use)

#define EV_TRIGGER_NULL

Reserved

#define EV_TRIGGER_PRESS

Press trigger

#define EV_TRIGGER_RELEASE

Release trigger

#define EV_TRIGGER_SCREEN_ENTERED

Screen entered trigger

#define EV_TRIGGER_STATE_CHANGED

State changed trigger

#define EV_TRIGGER_TIMER

Timer trigger

#define EV_TRIGGER_VALUE_CHANGED

Value changed trigger

#define NG_CALLBACK_DATA

Readability helper

#define NG_EVENT

Type caster from a derived event data structs (transition, periodic etc.) to the base **ng_event_base_t** data struct

#define PERIOD

Readability helper

#define PROGRESS

Readability helper

#define START_TIME

Readability helper

#define ng_event_is_paused

Checks if an event is paused (returns true/false)

#define ng_event_is_running

Checks if a screen transition is running (returns true/false)

#define ng_event_is_stopped

Checks if a screen transition is stopped (returns true/false)

13.1.6.4 Typedef Documentation

typedef void(* act_animptr_f)(ng_animation_data_t *data)

Typedef function pointer that takes a pointer to **ng_animation_data_t** as argument

typedef void(* act_gitptr_f)(struct _gitem_base_t *gitem)

Typedef function pointer that takes a pointer to a **gitem_base_t** as argument

typedef void(* act_gitptr_float_f)(struct _gitem_base_t *gitem, float value)

Typedef function pointer that takes a pointer to a **gitem_base_t** and a float as arguments

typedef void(* act_gitptr_int_f)(struct _gitem_base_t *gitem, int value)

Typedef function pointer that takes a pointer to a **gitem_base_t** and a int as arguments

typedef void(* act_gitptr_int_int_f)(struct _gitem_base_t *gitem, int a, int b)

Typedef function pointer that takes a pointer to a **gitem_base_t** and two int as arguments

typedef void(* act_gitptr_ptr_f)(struct _gitem_base_t *gitem, void *ptr)

Typedef function pointer that takes a pointer to a **gitem_base_t** and a void pointer as arguments

typedef void(* act_gitptr_uint_f)(struct _gitem_base_t *gitem, uint32_t value)

Typedef function pointer that takes a pointer to a **gitem_base_t** and a **uint32_t** as arguments

typedef void(* act_nodeptr_f)(struct _tree_node_t *node)

Typedef function pointer that takes a **tree_node_t** pointer as argument

```
typedef void(* act_nodeptr_nodeptr_f )(struct _tree_node_t
*node0, struct _tree_node_t *node1)
```

Typedef function pointer that takes two **tree_node_t** pointers as arguments

```
typedef void(* act_void_f )(void)
```

Typedef function pointer that takes no arguments

```
typedef void(* handler_f )(struct _ng_event_base_t *event,
uint32_t trigger)
```

Function pointer to an event handler

```
typedef void(* pause_toggle_f )(struct _ng_event_base_t *event,
bool pause)
```

Function pointer to an event pause-toggle function

```
typedef void(* start_f )(struct _ng_event_base_t *event)
```

Function pointer to an event start function

```
typedef void(* stop_f )(struct _ng_event_base_t *event, bool
force_finish)
```

Function pointer to an event stop function

13.1.6.5 Function Documentation

```
bool ng_event_check_retrigger_flag (  ng_event_base_t * event,  int
flag )
```

Checks if a re-trigger flag of an event is set or not.

Parameter	Description
*event	Pointer to the event
flag	Flag to be checked

Return

bool True is the flag is set, otherwise false.

```
bool ng_event_check_status_flag ( ng_event_base_t * event, int flag )
```

Checks the status of an event.

Parameter	Description
*event	Pointer to the event
flag	Flag to be checked

Return

bool True is the flag is set, otherwise false.

```
void ng_event_flip_status_flag (  ng_event_base_t * event, int flag )
```

Flips (inverts) a status flag of an event.

Parameter	Description
*event	Pointer to the event
flag	Flag to be flipped

```
void ng_event_handle (ng_event_base_t * event, uint32_t trigger_event
)
```

Handles an event according to the trigger that triggered it.

Parameter	Description
*event	Pointer to the event that needs to be handled
trigger_event	Triggers that caused the event

```
void ng_event_run_callback (  ng_event_base_t * event,  int
status_flags )
```

Runs the callback function of the event.

Parameter	Description
*event	Pointer to the event whose callback needs to run
status_flags	Allows the callback to run with specific flags

```
void ng_event_set_status ( ng_event_base_t * event, uint32_t status )
```

Set the status (stopped, running, paused) of an event.

Parameter	Description
*event	Pointer to the event
status	Status to be set

```
void ng_event_set_status_flag (  ng_event_base_t * event,  int flag )
```

Sets a status flag of an event.

Parameter	Description
*event	Pointer to the event
flag	Flag to be set

```
void ng_event_unset_status_flag ( ng_event_base_t * event, int flag )
```

Unset a status flag of an event.

Parameter	Description
*event	Pointer to the event
flag	Flag to be unset

13.1.7 ng_event_oneshot.h

This file contains the event handler of a one-shot event.

```
#include "ng_event.h"
```

13.1.7.1 Functions

```
void ng_oneshot_handler(ng_event_base_t *event, uint32_t trigger)
```

Function for handling the execution of a one-shot event.

13.1.7.2 Detailed Description

This file contains the event handler of a one-shot event.

13.1.7.3 Function Documentation

```
void ng_oneshot_handler ( ng_event_base_t * event, uint32_t trigger )
```

Function for handling the execution of a one-shot event.

Parameter	Description
*event	Pointer to the event that needs to be handled
trigger	The trigger that initiated the execution of the event

13.1.8 ng_event_periodic.h

Periodic event type.

```
#include "ng_event.h"
#include "ng_utils.h"
#include "ng_globals.h"
```

Data Structures

`struct ng_periodic_t`

More...

Macros

`#define NG_PERIODIC`

13.1.8.1 Functions

`void ng_periodic_handler(ng_event_base_t *event, uint32_t trigger)`

Handler function of a periodic event.

`void ng_periodic_start(ng_event_base_t *event)`

Start function for starting a periodic event.

`void ng_periodic_stop(ng_event_base_t *event, bool force_finish)`

Stop function for stopping a periodic transition.

`void ng_periodic_pause_toggle(ng_event_base_t *event, bool pause)`

Function for pausing or resuming a periodic event.

13.1.8.2 Detailed Description

Periodic event type.

Periodic is derived from the base event type **ng_event_base_t** and contains additional attributes regarding its timing as well as specific functions for controlling it (handler start stop **pause_toggle**).

13.1.8.3 Macro Definition Documentation

`#define NG_PERIODIC`

Type caster from base **ng_event_base_t** struct to derived **ng_periodic_t** struct

13.1.8.4 Function Documentation

`void ng_periodic_handler (ng_event_base_t * event, uint32_t trigger)`

Handler function of a periodic event.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be handled (casted internally to **ng_periodic_t**)
trigger	The trigger that initiated the execution of the event

`void ng_periodic_pause_toggle ( ng_event_base_t * event, bool pause )`

Function for pausing or resuming a periodic event.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be paused/resumed (casted internally to **ng_periodic_t**)
pause	If true, the periodic event will explicitly pause, otherwise if the periodic event is paused, it will resume its execution

`void ng_periodic_start (  ng_event_base_t * event )`

Start function for starting a periodic event.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be started (casted internally to **ng_periodic_t**)

`void ng_periodic_stop (  ng_event_base_t * event, bool force_finish )`

Stop function for stopping a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be stopped (casted internally to **ng_periodic_t**)
force_finish	unused (needed for function's signature)

13.1.9 ng_event_periodic_transition.h

Periodic transition event type.

#include "ng_event.h" #include "ng_utils.h"

Data Structures

`struct ng_periodic_transition_t`

More...

Macros

`#define NG_PERIODIC_TRANSITION`

13.1.9.1 Functions

`void ng_periodic_transition_handler(ng_event_base_t *event, uint32_t trigger)`

Handler function of a periodic transition.

`void ng_periodic_transition_start(ng_event_base_t *event)`

Start function for starting a periodic transition.

`void ng_periodic_transition_stop(ng_event_base_t *event, bool force_finish)`

Stop function for stopping a periodic transition.

`void ng_periodic_transition_pause_toggle(ng_event_base_t *event, bool pause)`

Function for pausing or resuming a periodic transition.

13.1.9.2 Detailed Description

Periodic transition event type.

Periodic transition is derived from the base event type **ng_event_base_t** and contains additional attributes regarding its timing as well as specific functions for controlling it (handler start stop **pause_toggle**).

13.1.9.3 Macro Definition Documentation

```
#define NG_PERIODIC_TRANSITION
```

Type caster from base **ng_event_base_t** struct to derived **ng_periodic_transition_t** struct

13.1.9.4 Function Documentation

```
void ng_periodic_transition_handler (  ng_event_base_t * event,
uint32_t trigger )
```

Handler function of a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be handled (casted internally to **ng_periodic_transition_t**)
trigger	The trigger that initiated the execution of the event

```
void ng_periodic_transition_pause_toggle (  ng_event_base_t * event,
bool pause )
```

Function for pausing or resuming a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be paused/resumed (casted internally to **ng_periodic_transition_t**)
pause	if true, the periodic transitions will explicitly pause, otherwise if the periodic transition is paused, it will resume its execution

```
void ng_periodic_transition_start (  ng_event_base_t * event )
```

Start function for starting a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be started (casted internally to **ng_periodic_transition_t**)

```
void ng_periodic_transition_stop (  ng_event_base_t * event, bool
force_finish )
```

Stop function for stopping a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be stopped (casted internally to **ng_periodic_transition_t**)
force_finish	if this is true, the periodic transition will go to its final state (progress = 1.f) and stop, otherwise it will reset to its initial state (progress = 0.f) and stop

13.1.10 ng_event_transition.h

Transition event type.

```
#include "ng_event.h"
#include "ng_utils.h"
```

Data Structures

```
struct ng_transition_t
```

More...

Macros

```
#define NG_TRANSITION
```

13.1.10.1 Functions

```
void ng_transition_handler(ng_event_base_t *event, uint32_t
trigger)
```

Handler function of a transition.

```
void ng_transition_start(ng_event_base_t *event)
```

Start function for starting a periodic transition.

```
void ng_transition_stop(ng_event_base_t *event, bool
force_finish)
```

Stop function for stopping a transition.

```
void ng_transition_pause_toggle(ng_event_base_t *event, bool
pause)
```

Function for pausing or resuming a transition.

```
void ng_transition_revert(ng_event_base_t *event)
```

Reverts the transition progress once (do not use this function to re-revert a transition)

```
void ng_transition_revert_force(ng_event_base_t *event, int set)
```

Reverts the transition progress.

13.1.10.2 Detailed Description

Transition event type.

Transition is derived from the base event type **ng_event_base_t** and contains additional attributes regarding its timing as well as specific functions for controlling it (handler start stop **pause_toggle**).

13.1.10.3 Macro Definition Documentation

#define NG_TRANSITION

Type caster from base **ng_event_base_t** struct to derived **ng_transition_t** struct

13.1.10.4 Function Documentation

```
void ng_transition_handler (  ng_event_base_t * event,  uint32_t
trigger )
```

Handler function of a transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be handled (casted internally to **ng_transition_t**)
trigger	The trigger that initiated the execution of the event

```
void ng_transition_pause_toggle (  ng_event_base_t * event,  bool
pause )
```

Function for pausing or resuming a transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be paused/resumed (casted internally to **ng_transition_t**)
pause	if true, the transitions will explicitly pause, otherwise if the transition is paused, it will resume its execution

```
void ng_transition_revert (  ng_event_base_t * event )
```

Reverts the transition progress once (do not use this function to re-revert a transition)

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the transition that needs to be reverted (casted internally to **ng_transition_t**)

```
void ng_transition_revert_force (  ng_event_base_t * event, int set )
```

Reverts the transition progress.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the transition that needs to be reverted (casted internally to **ng_transition_t**)
set	if this is equal to zero, the transition's final progress is 1.f, otherwise the final progress is 0.f

```
void ng_transition_start (  ng_event_base_t * event )
```

Start function for starting a periodic transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be started (casted internally to **ng_transition_t**)

```
void ng_transition_stop ( ng_event_base_t * event, bool force_finish)
```

Stop function for stopping a transition.

Parameter	Description
*event	Pointer to the base struct **ng_event_base_t** of the event that needs to be stopped (casted internally to **ng_transition_t**)
force_finish	if this is true, the periodic transition will go to its final state (progress = 1.f) and stop, otherwise it will reset to its initial state (progress = 0.f) and stop

13.1.11 ng_gestures.h

```
#include "ng_tree.h"
#include "nema_event.h"
```

Data Structures

struct gitem_gestures_t

More...

Macros

#define GESTURE_FUNC_ABORT_PRESS

Gesture's abort function signature.

#define GESTURE_FUNC_RELEASE

Gesture's release function signature.

#define GESTURE_FUNC_PRESS

Gesture's press function signature.

#define GESTURE_FUNC_SWIPE

Gesture's swipe function signature.

13.1.11.1 Functions

typedef GESTURE_FUNC_RELEASE(release_gesture_func_t)

Gesture's release function definition.

typedef GESTURE_FUNC_PRESS(press_gesture_func_t)

Gesture's press function definition.

typedef GESTURE_FUNC_SWIPE(swipe_gesture_func_t)

Gesture's swipe function definition.

typedef GESTURE_FUNC_ABORT_PRESS(abort_gesture_func_t)

Gesture's abort function definition.

tree_node_t* ng_gestures_press(nema_event_t *event, int event_press_x, int event_press_y)

Function executed on mouse/finger press.

void ng_gestures_release(nema_event_t *event)

Function executed on mouse/finger release.

void ng_gestures_swipe(nema_event_t *event, int mouse_dx, int mouse_dy)

Function executed on mouse/finger swipe/drag.

13.1.11.2 Function Documentation

```
bool ng_gestures_is_inside_popup (  int x,  int y )
```

Parameter	Description
x	Gesture's horizontal position
y	Gesture's vertical position

Return

bool True if the gesture was performed inside a pop-up, otherwise false.

```
tree_node_t * ng_gestures_press (  nema_event_t * event,  int
event_press_x, int event_press_y )
```

Function executed on mouse/finger press.

Parameter	Description
*event	Pointer to the press event as forwarded by the main loop
event_press_x	The x position (absolute coordinate) of the press event
event_press_y	The y position (absolute coordinate) of the press event

Return

tree_node_t* The tree node that accepted the "press" if its graphics item supports press, otherwise NULL.

```
void ng_gestures_release (  nema_event_t * event )
```

Function executed on mouse/finger release.

Parameter	Description
*event	Pointer to the release event as forwarded by the main loop

```
void ng_gestures_swipe (  nema_event_t * event,  int mouse_dx,  int
mouse_dy )
```

Function executed on mouse/finger swipe/drag.

Parameter	Description
*event	Pointer to the release event as forwarded by the main loop
mouse_dx	Horizontal difference in pixels to the previously captured event
mouse_dy	Vertical difference in pixels to the previously captured event

13.1.12 ng_gitem.h

```
#include "nema_core.h"
#include "nema_utils.h"
#include "nema_font.h"
#include "ng_typedefs.h"
```

Data Structures

struct __gitem_base_t

Graphics item base struct definition. More...

struct attr_text_t

Text attributes data struct. More...

Macros

```
#define GITEMF_ALWAYS
#define GITEMF_PRESS
#define GITEMF_RELEASE
#define GITEMF_HOLD
#define GITEMF_DRAG
#define GITEMF_FILL_COLOR
#define GITEMF_FILL_IMAGE
#define GITEMF_FILL_OUTLINE
#define GITEMF_HIGHLIGHTED
#define GITEMF_CHECKED
#define GITEMF_STOP_RECUR
#define GITEMF_HIDDEN
#define GITEMF_ANIMATED
#define GITEMF_PRESS_SCALE
#define GITEMF_CONTAINS_WINDOW
```

#define BASE_STRUCT

Define that contains all the **gitem_base_t** data struct attributes that are common among all different graphics items (widgets)

```
#define NG_GITEM
#define DRAW_FUNC
#define ID
#define X
#define Y
#define W
```

```
#define H
#define COLOR
#define EVENT
#define IMAGE
#define CUR_VAL
#define MAX_VAL
#define MIN_VAL
#define FLAGS
#define GESTURES
#define PEN_WIDTH
#define TEXT_COLOR
#define SEC_COLOR
#define SEC_IMAGE
#define ANGLE
#define MAX_ANGLE
#define MIN_ANGLE
#define STEP
#define X_ROT
#define Y_ROT
#define NEEDLE
#define HOUR
#define MINUTE
#define SECOND
#define RADIUS
#define SUFFIX
#define PAGE_COUNT
#define SPACING
#define CUR_STATE
#define STATE_COUNT
#define CUR_PAGE
#define INT_PRECISION
#define DEC_PRECISION
#define TIME_FORMAT
```

13.1.12.1 Enumerations

enum gitem_type_e

```
GITEM_MAIN_SCREEN, GITEM_CONTAINER, GITEM_CIRCLE, GITEM_RECT,
GITEM_ROUNDED_RECT, GITEM_IMAGE, GITEM_LABEL, GITEM_LABEL_BUTTON,
GITEM_CHECKBOX, GITEM_RADIO_BUTTON, GITEM_HORIZONTAL_SLIDER,
GITEM_VERTICAL_SLIDER, GITEM_DIGITAL_METER, GITEM_WINDOW_SCREEN,
GITEM_ICON, GITEM_TABLE, GITEM_HORIZONTAL_PROGRESS_BAR, GITEM_GAUGE,
GITEM_NEEDLE, GITEM_ICON_BUTTON, GITEM_WINDOW, GITEM_CIRCULAR_PROG-
RESS, GITEM_WATCH_FACE, GITEM_DIGITAL_CLOCK, GITEM_VERTICAL_PRO-
GRESS_BAR, GITEM_SWIPE_WINDOW, GITEM_TOGGLE_BUTTON, GITEM_TYPE_COUNT
```

Enumerator that contains the various graphics items types. Each type is documented in the next section (widgets)

13.1.12.2 Functions

`float ng_gitem_get_value(gitem_base_t *git)`

Gets the current value of the graphics item (if supported)

`void ng_gitem_set_flag(gitem_base_t *git, uint32_t flag)`

Sets a flag of a graphics item.

`void ng_gitem_unset_flag(gitem_base_t *git, uint32_t flag)`

Unsets a graphics item's flag.

`void ng_gitem_set_visible(gitem_base_t *git)`

Makes a graphics item visible (if was previously hidden)

`void ng_gitem_set_hidden(gitem_base_t *git)`

Hides a graphics item.

`void ng_gitem_set_alpha(gitem_base_t *git, uint32_t alpha)`

Set the opacity (alpha channel) of a graphics item.

`void ng_gitem_set_color(gitem_base_t *git, uint32_t rgba)`

Sets the color of a graphics item.

`void ng_gitem_set_position(gitem_base_t *git, int x, int y)`

Sets the position of a graphics item.

`void ng_gitem_set_x(gitem_base_t *git, int x)`

Sets the x-position of a graphics item.

`void ng_gitem_set_y(gitem_base_t *git, int y)`

Sets the y-position of a graphics item.

`void ng_gitem_set_size(gitem_base_t *git, int w, int h)`

Sets the size of a graphics item.

13.1.12.3 Macro Definition Documentation

`#define ANGLE`

Readability helper

`#define BASE_STRUCT`

Define that contains all the **gitem_base_t** data struct attributes that are common among all different graphics items (widgets).

These attributes are:

ng_event_base_t *event: Pointer to an event assigned to the graphics item

draw_f *draw: Pointer to the draw function

uint32_t flags: Graphics item's flags (as defined earlier)

int x: Horizontal (x) offset (with respect to its parent)
int y: Vertical (y) offset (with respect to its parent)
uint16_t w: Width
uint16_t h: Height

int id: Unique identification number

gitem_type_e type: Enumerator that indicates the type of the graphics

items uint32_t color: Base color (contains the item's opacity)

#define COLOR

Readability helper

#define CUR_PAGE

Readability helper

#define CUR_STATE

Readability helper

#define CUR_VAL

Readability helper

#define DEC_PRECISION

Readability helper

#define DRAW_FUNC

Draw function definition

#define EVENT

Readability helper

#define FLAGS

Readability helper

#define GESTURES

Readability helper

#define GITEMF_ALWAYS

Reserved

#define GITEMF_ANIMATED

The graphics item is animated (displayed using a back buffer)

#define GITEMF_CHECKED

The graphics item is checked

#define GITEMF_CONTAINS_WINDOW

Applicable to "Screen", indicates whether the screen contains a window or not

#define GITEMF_DRAG

The graphics item accepts mouse drag (swipe) events

#define GITEMF_FILL_COLOR

The graphics item draw function fills a color in the item's geometry

#define GITEMF_FILL_IMAGE

The graphics item draw function blits a texture in the item's geometry

#define GITEMF_FILL_OUTLINE

The graphics item draw function draw's the outline of the item

#define GITEMF_HIDDEN

The graphics item is hidden

#define GITEMF_HIGHLIGHTED

The graphics item is highlighted

#define GITEMF_HOLD

The graphics item accepts mouse hold events (reserved for future use)

#define GITEMF_PRESS

The graphics item accepts mouse press events

#define GITEMF_PRESS_SCALE

The graphics item scales itself (size) when pressed

#define GITEMF_RELEASE

The graphics item accepts mouse release events

#define GITEMF_STOP_RECUR

Stop a recursion flag

#define H

Readability helper

#define HOUR

Readability helper

#define ID

Readability helper

#define IMAGE

Readability helper

#define INT_PRECISION

Readability helper

#define MAX_ANGLE

Readability helper

#define MAX_VAL

#define MINUTE

Readability helper

#define MIN_ANGLE

Readability helper

#define MIN_VAL

Readability helper

#define NEEDLE

Readability helper

#define NG_GITEM

Type caster from a derived gitem data struct to the base **gitem_base_t**

#define PAGE_COUNT

Readability helper

#define PEN_WIDTH

Readability helper

#define RADIUS

Readability helper

#define SECOND

Readability helper

#define SEC_COLOR

Readability helper

#define SEC_IMAGE

#define SPACING

Readability helper

#define STATE_COUNT

Readability helper

#define STEP

Readability helper

#define SUFFIX

Readability helper

#define TEXT_COLOR

Readability helper

#define TIME_FORMAT

Readability helper

#define W

Readability helper

#define X

Readability helper

#define X_ROT

Readability helper

#define Y

Readability helper

#define Y_ROT

Readability helper

13.1.12.4 Function Documentation

float ng_gitem_get_value (gitem_base_t * git)

Gets the current value of the graphics item (if supported)

Parameter	Description
git	Pointer to the graphics item that its value should be returned

Return

float Value of the graphics item. If the item does not support "value", returns 0.f.

```
void ng_gitem_set_alpha (  gitem_base_t * git,  uint32_t alpha )
```

Set the opacity (alpha channel) of a graphics item.

Parameter	Description
git	Pointer to the graphics item
alpha	Opacity value

```
void ng_gitem_set_color (  gitem_base_t * git,  uint32_t rgba )
```

Sets the color of a graphics item.

Parameter	Description
git	Pointer to the graphics item
rgba	Color value

```
void ng_gitem_set_flag (  gitem_base_t * git,  uint32_t flag )
```

Sets a flag of a graphics item.

Parameter	Description
git	Pointer to the graphics item
flag	Flag to be set

```
void ng_gitem_set_hidden (  gitem_base_t * git )
```

Hides a graphics item.

Parameter	Description
git	Pointer to the graphics item

```
void ng_gitem_set_position (  gitem_base_t * git,  int x,  int y )
```

Sets the position of a graphics item.

Parameter	Description
git	Pointer to the graphics item
x	Horizontal offset
y	Vertical offset

```
void ng_gitem_set_size (  gitem_base_t * git,  int w,  int h )
```

Sets the size of a graphics item.

Parameter	Description
git	Pointer to the graphics item
w	Width
h	Height

```
void ng_gitem_set_visible (  gitem_base_t * git )
```

Makes a graphics item visible (if was previously hidden)

Parameter	Description
git	Pointer to the graphics item

```
void ng_gitem_set_x (  gitem_base_t * git,  int x )
```

Sets the x-position of a graphics item.

Parameter	Description
git	Pointer to the graphics item
x	Horizontal offset

```
void ng_gitem_set_y (  gitem_base_t * git,  int y )
```

Sets the y-position of a graphics item.

Parameter	Description
git	Pointer to the graphics item
y	Vertical position

```
void ng_gitem_unset_flag (  gitem_base_t * git,  uint32_t flag )
```

Unsets a graphics item's flag.

Parameter	Description
git	Pointer to the graphics item
flag	Flag to be unset

13.1.13 ng_globals.h

```
#include "nema_core.h"
#include "nema_font.h"
#include "nema_dc_hal.h"
#include "nema_dc.h"
#include "nema_easing.h"
#include "nema_transitions.h"
#include "nema_utils.h"
#include "ng_gitem.h"
#include "ng_tuples.h"
#include "ng_gestures.h"
#include "ng_tree.h"
#include "ng_progress_bar.h"
#include "ng_utils.h"
#include "ng_animation.h"
#include "ng_callbacks.h"
#include "ng_event.h"
#include "ng_event_transition.h"
#include "ng_event_oneshot.h"
#include "ng_event_periodic.h"
#include "ng_event_periodic_transition.h"
#include "ng_button.h"
#include "ng_checkbox.h"
#include "ng_circle.h"
#include "ng_circular_progress.h"
#include "ng_container.h"
#include "ng_digimeter.h"
#include "ng_gauge.h"
#include "ng_image.h"
#include "ng_label.h"
#include "ng_needle.h"
#include "ng_radio_button.h"
#include "ng_rect.h"
#include "ng_rounded_rect.h"
#include "ng_screen.h"
#include "ng_slider.h"
#include "ng_slider_hor.h"
#include "ng_slider_ver.h"
#include "ng_watchface.h"
#include "ng_icon.h"
#include "ng_window.h"
#include "ng_toggle_button.h"
#include "ng_swipe_window.h"
#include "ng_digital_clock.h"
```

Macros

```
#define NG_LAYOUT_HOR
#define NG_LAYOUT_VER
#define SCREEN_TRANSITION_PAUSED
#define SCREEN_TRANSITION_RUNNING
#define SCREEN_TRANSITION_STOPPED
#define DOING_SCREEN_TRANSITION
```

13.1.13.1 Functions

`void ng_globals_set_resolution(int resx, int resy)`

Sets the resolution of the applciation (NG_RESX and NG_RESY variables)

`void ng_globals_register_screen_transition_event(ng_event_base_t *event)`

Registers the screen transition event to the API. By default, this event is the first event of the generated event list.

`void ng_globals_register_event_list(ng_event_base_t **event_list, int list_size, int temp_animations)`

Registers the generated event list to the API.

`void ng_globals_register_screen_groups(int group_count, int popup_count, int *screens_per_group, tree_node_t ***nodes_per_group, nema_transition_t *effect_per_group, uint8_t *layout_per_group, tree_node_t **popup_nodes, int cur_group, int cur_screen, tree_node_t **cur_group_nodes)`

Registers the generated screen groups to the API.

`void ng_globals_register_framebuffers(int frame_buffer_count, img_obj_t *frame_buffers, int back_buffer_count, img_obj_t *back_buffers, nemadc_layer_t *layers)`

Registers the framebuffers to the API.

`int ng_globals_sanity_check()`

Performs a sanity check, that the generated project's parameters have been properly registered to the API.

13.1.13.2 Macro Definition Documentation

`#define DOING_SCREEN_TRANSITION`

Checks if a screen transition is stopped (returns true/false)

`#define NG_LAYOUT_HOR`

Horizontal Layout

`#define NG_LAYOUT_VER`

Vertical Layout

`#define SCREEN_TRANSITION_PAUSED`

Checks if the screen transition event is paused (returns true/false)

`#define SCREEN_TRANSITION_RUNNING`

Checks if the screen transition event is running (returns true/false)

```
#define SCREEN_TRANSITION_STOPPED
```

Checks if the screen transition event is stopped (returns true/false)

13.1.13.3 Function Documentation

```
void ng_globals_register_event_list ( ng_event_base_t ** event_list,
int list_size,  int temp_animations )
```

Registers the generated event list to the API.

Parameter	Description
**event_list	List (array) with pointers to the generated events
list_size	Size of the event list
temp_animations	Maximum count of temporary animations (e.g., swipe window animation)

```
void ng_globals_register_framebuffers (  int frame_buffer_count,
img_obj_t * frame_buffers,  int back_buffer_count,  img_obj_t *
back_buffers, nemadc_layer_t * layers )
```

Registers the framebuffers to the API.

Parameter	Description
frame_buffer_count	Count of the (front) framebuffers
frame_buffers	Array that contains the framebuffers
back_buffer_count	Count of the backbuffers
back_buffers	Array that contains the backbuffers
layers	Array that contains the display controller layers

```
void ng_globals_register_screen_groups (  int group_count,  int
popup_count, int * screens_per_group,  tree_node_t ***
nodes_per_group,  nema_transition_t * effect_per_group,  uint8_t *
layout_per_group,  tree_node_t ** popup_nodes, int cur_group,  int
cur_screen,  tree_node_t ** cur_group_nodes )
```

Registers the generated screen groups to the API.

Parameter	Description
group_count	Total count of the screen groups
popup_count	Total count the pop-ups

Parameter	Description
*screens_per_group	Array that contains how many screens belong to each screen group
***nodes_per_group	Array that contains pointers to arrays that contain the tree nodes of each screen group
*effect_per_group	Transition effect per screen group
*layout_per_group	Layout (horizontal-vertical) per screen group (application specific)
**popup_nodes	Array that contains the pointers of all the pop-up tree nodes
cur_group	Index of the current screen group
cur_screen	Index of the current screen
**cur_group_nodes	Array with pointers to the tree nodes of the current screen group

```
void ng_globals_register_screen_transition_event (  ng_event_base_t
* event )
```

Registers the screen transition event to the API. By default, this event is the first event of the generated event list.

Parameter	Description
event	Pointer to the event that will be used for the screen transitions

```
int ng_globals_sanity_check
```

Performs a sanity check, that the generated project's parameters have been properly registered to the API.

Return

int Zero if everything has been registered correctly, otherwise a positive number that indicates the error.

```
void ng_globals_set_resolution (  int resx,  int resy )
```

Sets the resolution of the application (NG_RESX and NG_RESY variables)

Parameter	Description
resx	Horizontal resolution
resy	Vertical resolution

13.1.14 ng_main_loop.h

NemaGUI main loop function.

13.1.14.1 Functions

```
void ng_main_loop(const int run_once)
```

The applications main loop function.

13.1.14.2 Detailed Description

NemaGUI main loop function.

This file must be included inside the main function's file. It contains the main loop function of a NemaGUI application.

13.1.14.3 Function Documentation

```
void ng_main_loop (  const int run_once )
```

The applications main loop function.

Parameter	Description
run_once	If this is zero, the application will enter an endless loop, otherwise the main loop will run for one iteration.

13.1.15 ng_screen_trans.h

```
#include "ng_globals.h"
#include "ng_event_transition.h"
#include "ng_utils.h"
#include "nema_transitions.h"
```

Macros

```
#define SCREEN_TRANSITION_DURATION_SECS
```

13.1.15.1 Functions

```
void ng_screen_trans_initialize(ng_event_base_t *event,
tree_node_t *to_screen, tree_node_t *from_screen,
nema_transition_t effect, int go_right, float initial_progress)
```

Initializes a screen transition.

```
void ng_screen_trans_swipe(float progress_diff)
```

Function executed when swiping during a screen transition.

```
void ng_screen_trans_resume(ng_event_base_t *event, float
duration, int abort)
```

Resumes an active screen transition after mouse/finger release (uses the timer)

```
void ng_screen_trans_pause(ng_event_base_t *event)
```

Pauses the screen transition's event.

```
void ng_screen_trans_swipable(ng_event_base_t *event, void *data)
```

Callback function assigned to the screen transition event.

```
void ng_callback_show_screen(ng_event_base_t *event, void *data)
```

Callback function executed when the screen transition's event is triggered by the timer.

```
void ng_callback_set_screen(ng_event_base_t *event, void *data)
```

Callback function for setting instantly the current screen.

13.1.15.2 Macro Definition Documentation

```
#define SCREEN_TRANSITION_DURATION_SECS
```

Defines the maximum duration of the screen transition's event

13.1.15.3 Function Documentation

```
void ng_callback_set_screen ( ng_event_base_t * event, void * data )
```

Callback function for setting instantly the current screen.

Parameter	Description
*event	Pointer to the event with this action
*data	Pointer to the screen's tree node that needs to be set as current screen (casted to **ng_tree_node_ptr_t** internally)

```
void ng_callback_show_screen ( ng_event_base_t * event, void * data )
```

Callback function executed when the screen transition's event is triggered by the timer.

Parameter	Description
*event	Pointer to the screen transition's event
*data	Transition data (casted to **ng_node_effect_direction_t** data struct internally)

```
void ng_screen_trans_initialize (  ng_event_base_t * event,
tree_node_t * to_screen,  tree_node_t * from_screen,
nema_transition_t effect,  int go_right, float initial_progress )
```

Initializes a screen transition.

Parameter	Description
*event	Pointer to the event struct that controls the transition
*to_screen	Pointer to the final screen's tree node
*from_screen	Pointer to the initial screen's tree node
effect	Transition effect
go_right	Parameter for the direction of the transition
initial_progress	Screen transition's event initial progress

```
void ng_screen_trans_pause (  ng_event_base_t * event )
```

Pauses the screen transition's event.

Parameter	Description
*event	Pointer to the screen transition's event

```
void ng_screen_trans_resume (  ng_event_base_t * event,  float dura-
tion,  int abort )
```

Resumes an active screen transition after mouse/finger release (uses the timer)

Parameter	Description
*event	Pointer to the event of the screen transition
duration	Remaining duration
abort	If this different than zero, the screen transition will return to its initial screen

```
void ng_screen_trans_swipable (  ng_event_base_t * event,  void *
data )
```

Callback function assigned to the screen transition event.

Parameter	Description
*event	Pointer to the screen transition event
data	Screen transition data (casted to **ng_node_effect_direction_t** type internal)

```
void ng_screen_trans_swipe (  float progress_diff )
```

Function executed when swiping during a screen transition.

Parameter	Description
progress_diff	Progress difference of the screen transition's event

13.1.16 ng_timer.h

Timer module.

```
#include "nema_core.h"
#include "ng_event.h"
```

13.1.16.1 Functions

```
int ng_timer_create()
```

Creates the periodic timer needed by the application.

```
void ng_timer_set_period(int ms)
```

Sets the period of the timer.

```
void ng_timer_start()
```

Starts the timer.

```
void ng_timer_stop()
```

Stops the timer.

```
void ng_timer_handler()
```

Function executed each time the timer ticks. Runs all the running events.

```
int ng_timer_get()
```

Gets the timer ID.

```
bool ng_timer_is_running()
```

Checks if the timer is currently running.

`int ng_timer_get_period()`

Gets the period of the timer.

`float ng_timer_get_frequency()`

Gets the timer frequency in Hz.

13.1.16.2 Detailed Description

Timer module.

Provides the functions needed to create a control the application timer. An Nema-GUI application requires one periodic timer.

13.1.16.3 Function Documentation

`int ng_timer_create`

Creates the periodic timer needed by the application.

Return

int The timer's ID.

`int ng_timer_get`

Gets the timer ID.

Return

int Timer ID.

`float ng_timer_get_frequency`

Gets the timer frequency in Hz.

Return

float Timer frequency in Hz

`int ng_timer_get_period`

Gets the period of the timer.

Return

int Timer period in ms.

`bool ng_timer_is_running`

Checks if the timer is currently running.

Return

bool true if the timer is running, otherwise false.

```
void ng_timer_set_period (  int ms )
```

Sets the period of the timer.

Parameter	Description
ms	Period in ms

13.1.17 ng_tree.h

This file contains the necessary data and functions for controlling a project's tree struct.

```
#include "ng_typedefs.h"
#include "nema_sys_defs.h"
```

Data Structures

```
struct tree_node_t
```

More...

Macros

```
#define NG_TREE_NODE
#define PARENT_NODE
#define FIRST_CHILD_NODE
#define NEXT_NODE
```

13.1.17.1 Functions

```
void ng_tree_set_current_screen(tree_node_t *node)
```

Sets the current screen.

```
void ng_tree_set_current_popup(tree_node_t *node)
```

Sets the current pop-up to be displayed.

```
tree_node_t* ng_tree_get_current_screen()
```

Returns the tree node of the current screen.

```
tree_node_t* ng_tree_get_node_under_cursor(tree_node_t *node,
uint32_t gesture, int x, int y, int x_off, int y_off, int
*click_x, int *click_y)
```

Recursive function, gets the tree node under the cursor (mouse, finger etc)

13.1.17.2 Detailed Description

This file contains the necessary data and functions for controlling a project's tree struct.

13.1.17.3 Macro Definition Documentation

`#define FIRST_CHILD_NODE`

Readability helper

`#define NEXT_NODE`

Readability helper

`#define NG_TREE_NODE`

Type caster for casting a void pointer to **tree_node_t** pointer struct

`#define PARENT_NODE`

Readability helper

13.1.17.4 Function Documentation

`tree_node_t * ng_tree_get_current_screen`

Returns the tree node of the current screen.

Return

tree_node_t* Node of the current screen.

```
tree_node_t * ng_tree_get_node_under_cursor (  tree_node_t * node,
uint32_t gesture,  int x,  int y,  int x_off,  int y_off,  int *
click_x,  int * click_y )
```

Recursive function, gets the tree node under the cursor (mouse, finger etc)

Parameter	Description
node	Node to be checked in the current iteration (recursion)
gesture	Gesture that was initiated by the cursor
x	Cursor x
y	Cursor y
x_off	Horizontal offset with respect to the parent node
y_off	Vertical offset with respect to the parent node

Parameter	Description
click_x	Node's absolute x position
click_y	Node's absolute y position

Return

tree_node_t* Returns the tree node that supports the specific gesture.

```
void ng_tree_set_current_popup (  tree_node_t * node )
```

Sets the current pop-up to be displayed.

Parameter	Description
*node	Pointer to a tree node that contains the pop-up

```
void ng_tree_set_current_screen (  tree_node_t * node )
```

Sets the current screen.

Parameter	Description
*node	Pointer to the node that contains the new screen

13.1.18 ng_tuples.h

Tuples are core data structs used by NemaGUI API.

```
#include "ng_typedefs.h"
#include "ng_gitem.h"
#include "ng_animation.h"
#include "nema_core.h"
#include "nema_easing.h"
#include "nema_utils.h"
```

Data Structures

```
struct ng_point_t
```
More...

```
struct ng_git_uint32_t
```
More...

```
struct ng_git_uint32_uint32_ez_t
```
More...

```
struct ng_git_ptr_t
```
More...

```
struct ng_git_float_t
```

More...

struct ng_git_float_float_ez_t
More...

struct ng_git_int_int_t
More...

struct ng_git_int_int_ez_t
More...

struct ng_git_int_int_pair_ez_t
More...

struct ng_node_effect_direction_t
More...

struct ng_node_node_t
More...

struct ng_gitptr_t
More...

struct ng_tree_node_ptr_t
More...

Detailed Description

Tuples are core data structs used by NemaGUI API.

A major use of these data structs is in the event handling mechanism. Callback functions accept a void pointer in their arguments (signature constraint). Depending on the specific action that should take place during an event this void pointer is casted inside the implementation of each callback to specific data structs, most of which are defined here.

13.1.19 ng_typedefs.h

Typedefs

typedef struct _gitem_base_t gitem_base_t typedef struct _gitem_base_t gitem_base_t typedef struct _ng_event_base_t ng_event_base_t typedef struct _ng_event_base_t ng_event_base_t

typedef float(* easing_f)(float val)typedef float(* easing_f) (float val)

More...

typedef void(* draw_f)(struct _gitem_base_t *gitem, int x_off, int y_off)typedef void(* draw_f) (struct _gitem_base_t *gitem, int x_off, int y_off)

More...

```
typedef void(*  callback_f )(struct _ng_event_base_t *event, void
*data)typedef void(* callback_f) (struct _ng_event_base_t *event,
void *data)
```

More...

Detailed Description

Typedef Documentation

```
typedef void(* callback_f )(struct _ng_event_base_t *event, void
*data)
```

Typedef to callback function pointer (used by the g_event_base_t struct)

```
typedef void(* draw_f )(struct _gitem_base_t *gitem, int x_off,
int y_off)
```

Typedef to draw function pointer (used by the gitem_base_t struct)

```
typedef float(* easing_f )(float val)
```

Typedef to easing function pointer

13.1.20 ng_utils.h

```
#include "nema_core.h"
#include "nema_utils.h"
```

Macros

```
#define SAFE_CAST
#define CLAMP
#define NG_ARRAY_MAX_SIZE
#define NG_LOAD_ARRAY
```

Macro Definition Documentation

```
#define NG_ARRAY_MAX_SIZE
```

Maximum size (length) for arrays used internally

13.2 Directories

Here is a list of all directories with brief descriptions:

13.3 Data Structures

Here is a list of all data structures with brief descriptions:

13.3.1 __gitem_base_t

Data Structure

Graphics item base struct definition.

`#include <ng_gitem.h>`

Data Fields

`BASE_STRUCT`

Attributes as defined in the description of **BASE_STRUCT** (event pointer, draw function pointer, gestures pointer, flags, x, y, w, h, id, type, color)

Detailed Description

Graphics item base struct definition.

13.3.2 __ng_event_base_t

Data Structure

Event base struct definition.

`#include <ng_event.h>`

Data Fields

`EVENT_BASE_STRUCT`

Attributes as defined in the description of **EVENT_BASE_STRUCT**

Detailed Description

Event base struct definition.

13.3.3 attr_text_t

Data Structure

Text attributes data struct.

`#include <ng_gitem.h>`

Data Fields

`uint8_t index`

Current index that helps identifying the graphics item's current text and font

`nema_font_t ** fonts`

Pointer to an array of font

pointer char ** texts

Pointer to an array of strings

uint32_t alignment

Text alignment (bitwise operator)

Detailed Description

Text attributes data struct.

13.3.4 gitem_gestures_t

Data Structure

```
#include <ng_gestures.h>
```

Data Fields

press_gesture_func_t * press

Function pointer to press function

release_gesture_func_t * release

Function pointer to release function

swipe_gesture_func_t * swipe

Function pointer to swipe function

abort_gesture_func_t * abort

Function pointer to abort function

Data struct that contains function pointers to gestures

Detailed Description

Data struct that contains function pointers to gestures

13.3.5 ng_act_ptr

Union

Union that groups together pointers to all possible actions that are supported in the event mechanism.

```
#include <ng_event.h>
```

Data Fields

act_gitptr_float_f act_gitptr_float

Function pointer to a **act_gitptr_float_f**

function act_gitptr_ptr_f act_gitptr_ptr

Function pointer to a **act_gitptr_ptr_f** function

act_gitptr_int_f act_gitptr_int

Function pointer to a **act_gitptr_int_f** function

act_gitptr_uint_f act_gitptr_uint

Function pointer to a **act_gitptr_uint_f**

function act_gitptr_f act_gitptr

Function pointer to a **act_gitptr_f** function

act_gitptr_int_int_f act_gitptr_int_int

Function pointer to a **act_gitptr_int_int_f** function

act_nodeptr_f act_nodeptr

Function pointer to a **act_nodeptr_f** function

act_nodeptr_nodeptr_f act_nodeptr_nodeptr

Function pointer to a **act_nodeptr_nodeptr_f**

function act_animptr_f act_animptr

Function pointer to a **act_animptr_f** function

act_void_f act_void

Function pointer to a **act_void_f** function

13.3.6 ng_animation_data_t

Data Structure

#include <ng_animation.h>

Data Fields

tree_node_t * node

Pointer to the animated tree node

easing_f ez_func

Pointer to easing

function int back_buffer_index

Index to the back buffer used (if available)

void * ext_data

Pointer to extra data (ng_point_t is currently supported)

int action

Action to be performed (show or hide)

Data struct that contains the animation data

Detailed Description

Data struct that contains the animation data

13.3.7 ng_git_float_float_ez_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item float val_0

First float

value float val_1

Second float value

easing_f easing

Pointer to easing function

Data struct that contains a pointer to a **gitem_base_t**, two float values and a pointer to an easing function

Detailed Description

Data struct that contains a pointer to a **gitem_base_t**, two float values and a pointer to an easing function

13.3.8 ng_git_float_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item float val

Floating point value

Data struct that contains a pointer to a **gitem_base_t** and a float value

Detailed Description

Data struct that contains a pointer to a **gitem_base_t** and a float value

13.3.9 ng_git_int_int_ez_t

Data Structure

#include <ng_tuples.h>

Data Fields

gitem_base_t * git

Pointer to a graphics

item int a

First int

value int b

First int

value easing_f easing

Pointer to an easing function

Data struct that contains a pointer to a **gitem_base_t**, two int values and a pointer to an easing function

Detailed Description

Data struct that contains a pointer to a **gitem_base_t**, two int values and a pointer to an easing function

13.3.10 ng_git_int_int_pair_ez_t

Data Structure

#include <ng_tuples.h>

Data Fields

gitem_base_t * git

Pointer to a graphics

item int a0

First value of the first

pair int a1

Second value of the first

pair int b0

First value of the second

pair int b1

Second value of the second

pair easing_f easing

Pointer to an easing function

Data struct that contains a pointer to a **gitem_base_t**, two pairs of int values and a pointer to an easing function

Detailed Description

Data struct that contains a pointer to a **gitem_base_t**, two pairs of int values and a pointer to an easing function

13.3.11 ng_git_int_int_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item int a

First int

value int b

Second int value

Data struct that contains a pointer to a **gitem_base_t** and two int values

Detailed Description

Data struct that contains a pointer to a **gitem_base_t** and two int values

13.3.12 ng_git_ptr_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item void * ptr

Void to a graphics item

Data struct that contains a pointer to a **gitem_base_t** and a void pointer

Detailed Description

Data struct that contains a pointer to a **gitem_base_t** and a void pointer

13.3.13 ng_git_uint32_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item uint32_t val uint32_t value

Data struct that contains a pointer to a **gitem_base_t** and a **uint32_t** value

Detailed Description

Data struct that contains a pointer to a **gitem_base_t** and a **uint32_t** value

13.3.14 ng_git_uint32_uint32_ez_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

gitem_base_t * git

Pointer to a graphics

item uint32_t val_0

First **uint32_t** value

uint32_t val_1

Second **uint32_t** value

easing_f easing

Pointer to easing function

Data struct that contains a pointer to a **gitem_base_t**, two **uint32_t** values and a pointer to an easing function

Detailed Description

Data struct that contains a pointer to a **gitem_base_t**, two **uint32_t** values and a pointer to an easing function

13.3.15 ng_gitptr_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

```
gitem_base_t * git
```

Pointer to a graphics item

Utility data struct that contains a **gitem_base_t** pointer

Detailed Description

Utility data struct that contains a **gitem_base_t** pointer

13.3.16 ng_node_effect_direction_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

```
tree_node_t * node
```

Pointer to a tree node

```
nema_transition_t effect
```

Transition effect

```
int direction
```

Direction (see the defines in ng_animation.h)

Data struct that contains a pointer to a **tree_node_t**, a transition effect and a direction value (according to the defines in **ng_animation.h**)

Detailed Description

Data struct that contains a pointer to a **tree_node_t**, a transition effect and a direction value (according to the defines in **ng_animation.h**)

13.3.17 ng_node_node_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

tree_node_t * node0

Pointer to the first tree node

tree_node_t * node1

Pointer to the second tree node

Data struct that contains two pointers to a **tree_node_t** data structs

Detailed Description

Data struct that contains two pointers to a **tree_node_t** data structs

13.3.18 ng_periodic_t

Data Structure

```
#include <ng_event_periodic.h>
```

Data Fields

EVENT_BASE_STRUCT

Inherited attributes from **ng_event_base_t** data

struct float start_time

Start time of the periodic

transition float period

Period in seconds

Data struct that contains a periodic event's data

Detailed Description

Data struct that contains a periodic event's data

13.3.19 ng_periodic_transition_t

Data Structure

```
#include <ng_event_periodic_transition.h>
```

Data Fields

EVENT_BASE_STRUCT

Inherited attributes from **ng_event_base_t** data

struct float start_time

Start time of the periodic

transition float duration

Duration in seconds (must be less or equal to period)

float progress

Progress [0.f, 1.f] float period

Period in seconds

Data struct that contains a periodic transition's data

Detailed Description

Data struct that contains a periodic transition's data

13.3.20 ng_point_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

int x

x coordinate

int y

y coordinate

Data struct that contains point (x,y coordinates)

Detailed Description

Data struct that contains point (x,y coordinates)

13.3.21 ng_transition_t

Data Structure

```
#include <ng_event_transition.h>
```

Data Fields

EVENT_BASE_STRUCT

Inherited attributes from **ng_event_base_t** data

struct float start_time

Start time of the periodic

transition float duration

Duration in seconds

float progress

Progress [0.f, 1.f]

Data struct that contains a transition's data

Detailed Description

Data struct that contains a transition's data

13.3.22 ng_tree_node_ptr_t

Data Structure

```
#include <ng_tuples.h>
```

Data Fields

tree_node_t * node

Pointer to a tree node

Utility data struct that contains a **tree_node_t** pointer

Detailed Description

Utility data struct that contains a **tree_node_t** pointer

13.3.23 tree_node_t

Data Structure

```
#include <ng_tree.h>
```

Data Fields

gitem_base_t * this_

Pointer to the graphics item that the node contains struct

_tree_node_t * parent

Pointer to the parent node struct

_tree_node_t * first_child

Pointer to the first child of the node struct

`_tree_node_t * next`

Pointer to the next node (in the same hierarchy level)

Data struct that defines a tree node

Detailed Description

Data struct that defines a tree node

SECTION

14

Widget References

This section provides interface of each widgets supported by NemaGUI API. The more widgets are used in an application, the more memory is required in order to store them. The following table summarizes the memory needed for each instance of a widget (graphics item).

Table 14-1: Widgets Reference

Widget	Bytes/Widget
Screen	60
Circle	56
Rectangle	60
Rounded Rectangle	60
Image	60
Label	84
Container	64
Table	56
Container Array	56
Window	56
Swipe Window	84
Label Button	72
Toggle Button	64
Icon Button	72
Radio Button	72
Checkbox	72

Table 14-1: Widgets Reference *(Continued)*

Widget	Bytes/Widget
Horizontal Slider	68
Vertical Slider	68
Digital Meter	92
Icon	56
Horizontal Progress Bar	76
Vertical Progress Bar	76
Gauge	100
Circular Progress	76
Watch-face	76
Digital Clock	72

14.1 Files

Here is a list of all files with brief descriptions:

14.1.1 ng_button.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_button_t
```

More...

Macros

```
#define NG_BUTTON
```

14.1.1.1 Functions

```
DRAW_FUNC(ng_button_draw)
```

Draw function.

```
void ng_button_set_primary_image(gitem_base_t *git, void
*asset_ptr)
```

Sets the primary image asset.

```
void ng_button_set_secondary_image(gitem_base_t *git, void
*asset_ptr)
```

Sets the secondary image asset.

```
void ng_button_set_secondary_color(gitem_base_t *git, uint32_t
rgba)
```

Sets the secondary color.

14.1.1.2 Macro Definition Documentation

```
#define NG_BUTTON
```

Type caster from base **gitem_base_t** struct to derived **gitem_button_t** struct

14.1.1.3 Function Documentation

```
DRAW_FUNC (  ng_button_draw )
```

Draw function.

Parameter	Description
*git	Pointer to image's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_button_set_primary_image (  gitem_base_t * git,  void *
asset_ptr )
```

Sets the primary image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_button_set_secondary_color (  gitem_base_t * git,  uint32_t
rgba )
```

Sets the secondary color.

Parameter	Description
*git	Pointer to target gitem (gitem_base_t data struct)
rgba	Color value

R

Return

void

```
void ng_button_set_secondary_image (  gitem_base_t * git,  void *
asset_ptr )
```

Sets the secondary image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

14.1.2 ng_checkbox.h

```
#include "ng_gitem.h"
#include "ng_tree.h"
#include "ng_gestures.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_checkbox_t
```

More...

Macros

```
#define NG_CHECKBOX
```

14.1.2.1 Functions

```
DRAW_FUNC(ng_checkbox_draw)
```

Draw function.

```
void ng_checkbox_set_image(gitem_base_t *git, void *asset_ptr)
```

Sets the foreground image asset.

```
void ng_checkbox_set_secondary_color(gitem_base_t *git, uint32_t
rgba)
```

Sets the secondary color.

14.1.2.2 Macro Definition Documentation

```
#define NG_CHECKBOX
```

Type caster from base **gitem_base_t** struct to derived **gitem_checkbox_t** struct

14.1.2.3 Function Documentation

```
DRAW_FUNC (  ng_checkbox_draw )
```

Draw function.

Parameter	Description
*git	Pointer to checkbox's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_checkbox_set_image (  gitem_base_t * git,  void * asset_ptr )
```

Sets the foreground image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_checkbox_set_secondary_color (  gitem_base_t * git,  uint32_t
rgba )
```

Sets the secondary color.

Parameter	Description
*git	Pointer to target gitem (gitem_base_t data struct)
rgba	Color value

Return

void

14.1.3 ng_circle.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

`struct gitem_circle_t`

More...

Functions

`DRAW_FUNC(ng_circle_draw)`

Draw function.

14.1.3.1 Function Documentation

`DRAW_FUNC ( ng_circle_draw )`

Draw function.

Parameter	Description
*git	Pointer to checkbox's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

14.1.4 ng_circular_progress.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

`struct gitem_circular_progress_t`

More...

Macros

`#define NG_CIRCULAR_PROGRESS`

14.1.4.1 Functions

`DRAW_FUNC(ng_circular_progress_draw)`

Draw function.

`void ng_circular_progress_set_percent(gitem_base_t *git, float percent)`

Sets the current value (percent) of the progress bar.

`void ng_circular_progress_set_background_image(gitem_base_t *git, void *asset_ptr)`

Sets the background image asset.

`void ng_circular_progress_set_foreground_image(gitem_base_t *git, void *asset_ptr)`

Sets the foreground image asset.

14.1.4.2 Macro Definition Documentation

`#define NG_CIRCULAR_PROGRESS`

Type caster from base **gitem_base_t** struct to derived **gitem_circular_progress_t** struct

14.1.4.3 Function Documentation

`DRAW_FUNC ( ng_circular_progress_draw )`

Draw function.

Parameter	Description
*git	Pointer to circular progress's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_circular_progress_set_background_image ( gitem_base_t * git,
void * asset_ptr )
```

Sets the background image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_circular_progress_set_foreground_image ( gitem_base_t * git,
void * asset_ptr )
```

Sets the foreground image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_circular_progress_set_percent (  gitem_base_t * git,  float
percent )
```

Sets the current value (percent) of the progress bar.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
percent	Percent value in range [0.f , 1.f]. If it is beyond the acceptable range, it is automatically clamped

Return

void

14.1.5 ng_container.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

`struct gitem_container_t`

More...

Macros

```
#define NG_CONTAINER
```

14.1.5.1 Functions

`DRAW_FUNC(ng_container_draw)`

Draw function.

`void ng_container_set_image(gitem_base_t *git, void *asset_ptr)`

Sets the foreground image asset.

14.1.5.2 Macro Definition Documentation

#define NG_CONTAINER

Type caster from base **gitem_base_t** struct to derived **gitem_container_t** struct

14.1.5.3 Function Documentation

DRAW_FUNC (ng_container_draw)

Draw function.

Parameter	Description
*git	Pointer to circular progress's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_container_set_image (  gitem_base_t * git,  void * asset_ptr
)
```

Sets the foreground image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

14.1.6 ng_digimeter.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
#include "ng_draw_prim.h"
```

Data Structures

```
struct gitem_digimeter_t
```

More...

Macros

```
#define NG_DIGIMETER
```

14.1.6.1 Functions

```
DRAW_FUNC(ng_digimeter_draw)
```

Draw function.

```
void ng_digimeter_set_value(gitem_base_t *git, float value)
```

Sets the current value of the digital meter.

```
void ng_digimeter_set_percent(gitem_base_t *git, float percent)
```

Sets the current value of the digital meter by percent (value = percent*(**max_val**-**min_val**) + **min_val**). Percent must be within [0.f, 1.f].

```
void ng_digimeter_count_up(gitem_base_t *git)
```

Count up, increase the digital meter's value by its step.

```
void ng_digimeter_count_down(gitem_base_t *git)
```

Count down, decrease the digital meter's value by its step.

```
void ng_digimeter_set_text_color(gitem_base_t *git, uint32_t
rgba)
```

Sets the digital meter's text color.

14.1.6.2 Macro Definition Documentation

```
#define NG_DIGIMETER
```

Type caster from base **gitem_base_t** struct to derived **gitem_digimeter_t** struct

14.1.6.3 Function Documentation

```
DRAW_FUNC (  ng_digimeter_draw )
```

Draw function.

Parameter	Description
*git	Pointer to digimeter's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_digimeter_count_down (  gitem_base_t * git )
```

Count down, decrease the digital meter's value by its step.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)

Return

void

```
void ng_digimeter_count_up (  gitem_base_t * git )
```

Count up, increase the digital meter's value by its step.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)

Return

void

```
void ng_digimeter_set_percent (  gitem_base_t * git,  float percent )
```

Sets the current value of the digital meter by percent (value = percent*(**max_val**-**min_val**) + **min_val**). Percent must be within [0.f, 1.f].

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
percent	Percentage in range [0.f, 1.f]. If it is beyond the acceptable range, it is automatically clamped

Return

void

```
void ng_digimeter_set_text_color (  gitem_base_t * git,  uint32_t
rgba )
```

Sets the digital meter's text color.

Parameter	Description
*git	Pointer to target gitem (gitem_base_t data struct)
rgba	Color value

Return

void

```
void ng_digimeter_set_value (  gitem_base_t * git,  float value )
```

Sets the current value of the digital meter.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
value	Value in range [**min_value** , **max_value**]. If it is beyond the acceptable range, it is automatically clamped

Return

void

14.1.7 ng_digital_clock.h

```
#include "ng_gitem.h"
```

```
#include "ng_utils.h"
#include "ng_draw_prim.h"
```

Data Structures

struct gitem_digital_clock_t

More...

Macros

```
#define NG_TIME_HH_MM_SS
#define NG_TIME_HH_MM
#define NG_TIME_H_MM
#define NG_TIME_HH
#define NG_TIME_H
#define NG_TIME_MM
#define NG_TIME_SS
#define NG_DIGITAL_CLOCK
```

14.1.7.1 Functions

DRAW_FUNC(ng_digital_clock_draw)

Draw function.

void ng_digital_clock_update(gitem_base_t *git)

Updates the clock's text according to the current time. System's wall time is used by default.

14.1.7.2 Macro Definition Documentation

#define NG_DIGITAL_CLOCK

Type caster from base **gitem_base_t** struct to derived **gitem_digital_-clock_t** struct

#define NG_TIME_H

Time format eg. "9"

#define NG_TIME_HH

Time format eg. "09"

#define NG_TIME_HH_MM

Time format eg. "09:45"

#define NG_TIME_HH_MM_SS

Time format eg. "09:45:18"

#define NG_TIME_H_MM

Time format eg. "9:45"

#define NG_TIME_MM

Time format eg. "45"

#define NG_TIME_SS

Time format eg. "18"

14.1.7.3 Function Documentation

DRAW_FUNC (ng_digital_clock_draw)

Draw function.

Parameter	Description
git	Pointer to digital clocks's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

void ng_digital_clock_update (gitem_base_t * git)

Updates the clock's text according to the current time. System's wall time is used by default.

In order to use a different time update method (not the system's wall time), the define **WALL_TIME_CLOCKS** (defined in the compiler flags of the generated Makefile) needs to be undefined and the time needs to be updated inside the #else segment of this function.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)

Return

void

14.1.8 ng_gauge.h

```
#include "ng_gitem.h"
#include "ng_gestures.h"
#include "ng_utils.h"
```

Data Structures

`struct gitem_gauge_t`

More...

Macros

```
#define NG_GAUGE
```

14.1.8.1 Functions

`DRAW_FUNC(ng_gauge_draw)`

Draw function.

`void ng_gauge_set_value(gitem_base_t *git, float value)`

Sets the current value of thegauge.

`void ng_gauge_set_percent(gitem_base_t *git, float percent)`

Sets the current value of the gauge by percent (value = percent*(max_val-min_val) + min_val). Percent must be within [0.f, 1.f].

`void ng_gauge_set_image(gitem_base_t *git, void *asset_ptr)`

Sets the image asset.

14.1.8.2 Macro Definition Documentation

`#define NG_GAUGE`

Type caster from base **gitem_base_t** struct to derived **gitem_gauge_t** struct

14.1.8.3 Function Documentation

`DRAW_FUNC ( ng_gauge_draw )`

Draw function.

Parameter	Description
*git	Pointer to gauge's base gitem (**gitem_base_t** data struct)

Parameter	Description
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_gauge_set_image (  gitem_base_t * git,  void * asset_ptr )
```

Sets the image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_gauge_set_percent (  gitem_base_t * git,  float percent )
```

Sets the current value of the gauge by percent (value = percent*(**max_val**-**min_val**) + **min_val**). Percent must be within [0.f, 1.f].

Parameter	Description
*git	Pointer to target gitem (gitem_base_t data struct)
percent	Percentage value

Return

void

```
void ng_gauge_set_value (  gitem_base_t * git,  float value )
```

Sets the current value of the gauge.

Parameter	Description

*gitPointer to target gitem (gitem_base_t data struct)

valueValue

Return

void

14.1.9 ng_icon.h

#include "ng_gitem.h"

#include "ng_utils.h"

Data Structures

struct gitem_icon_t

More...

14.1.10 ng_image.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_image_t
```

More...

Macros

```
#define NG_IMAGE
```

Functions

```
DRAW_FUNC(ng_image_draw)
```

Draw function.

```
void ng_image_set_asset(gitem_base_t *git, void *asset_ptr)
```

Set image asset.

14.1.10.1 Macro Definition Documentation

```
#define NG_IMAGE
```

Type caster from base **gitem_base_t** struct to derived **gitem_image_t** struct

14.1.10.2 Function Documentation

```
DRAW_FUNC (  ng_image_draw )
```

Draw function.

Parameter	Description
*git	Pointer to image's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_image_set_asset (  gitem_base_t * git,  void * asset_ptr )
```

Set image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

14.1.11 ng_label.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_label_t
```

More...

Macros

```
#define NG_LABEL
```

Functions

```
DRAW_FUNC(ng_label_draw)
```

Draw function.

```
void ng_label_set_text_color(gitem_base_t *git, uint32_t rgba)
```

Sets the label's text color.

14.1.11.1 Macro Definition Documentation

```
#define NG_LABEL
```

Type caster from base **gitem_base_t** struct to derived **gitem_label_t** struct

14.1.11.2 Function Documentation

```
DRAW_FUNC (  ng_label_draw )
```

Draw function.

Parameter	Description
*git	Pointer to label's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_label_set_text_color (  gitem_base_t * git,  uint32_t rgba )
```

Sets the label's text color.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
rgba	Color value

Return

void

14.1.12 ng_needle.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_needle_t
```

More...

Macros

```
#define NG_NEEDLE
```

Functions

```
DRAW_FUNC(ng_needle_draw)
```

Draw function.

```
void ng_needle_set_image(gitem_base_t *git, void *asset_ptr)
```

Sets the image asset.

14.1.12.1 Macro Definition Documentation

```
#define NG_NEEDLE
```

Type caster from base **gitem_base_t** struct to derived **gitem_needle_t** struct

14.1.12.2 Function Documentation

```
DRAW_FUNC (  ng_needle_draw )
```

Draw function.

Parameter	Description
*git	Pointer to needle's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_needle_set_image (  gitem_base_t * git,  void * asset_ptr )
```

Sets the image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

14.1.13 ng_progress_bar.h

```
#include "ng_gitem.h"
```

```
#include "ng_utils.h"
```

Data Structures

```
struct gitem_progress_bar_t
```

More...

Macros

```
#define NG_PROGRESS_BAR
```

Functions

```
DRAW_FUNC(ng_horizontal_progress_bar_draw)
```

Draw function (horizontal progress bar)

```
DRAW_FUNC(ng_vertical_progress_bar_draw)
```

Draw function (vertical progress bar)

```
void ng_progress_bar_set_percent(gitem_base_t *git, float
percent)
```

Sets the current value (percent) of the progress bar.

```
void ng_progress_bar_set_background_image(gitem_base_t *git, void
*asset_ptr)
```

Sets the background image asset.

```
void ng_progress_bar_set_foreground_image(gitem_base_t *git, void
*asset_ptr)
```

Sets the foreground image asset.

```
void ng_progress_bar_set_foreground_color(gitem_base_t *git,
uint32_t rgba)
```

Sets the foreground color.

14.1.13.1 Macro Definition Documentation

```
#define NG_PROGRESS_BAR
```

Type caster from base **gitem_base_t** struct to derived **gitem_progress_bar_t** struct

14.1.13.2 Function Documentation

```
DRAW_FUNC (  ng_horizontal_progress_bar_draw )
```

Draw function (horizontal progress bar)

Parameter	Description
*git	Pointer to needle's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
DRAW_FUNC (  ng_vertical_progress_bar_draw )
```

Draw function (vertical progress bar)

Parameter	Description
*git	Pointer to progress bar's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_progress_bar_set_background_image (  gitem_base_t * git,
void * asset_ptr )
```

Sets the background image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to background image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_progress_bar_set_foreground_color (  gitem_base_t * git,
uint32_t rgba )
```

Sets the foreground color.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
rgba	Foreground color value

Return

void

```
void ng_progress_bar_set_foreground_image (  gitem_base_t * git,
void * asset_ptr )
```

Sets the foreground image asset.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
*asset_ptr	Pointer to foreground image asset (casted to **img_obj_t** internally)

Return

void

```
void ng_progress_bar_set_percent ( gitem_base_t * git, float percent)
```

Sets the current value (percent) of the progress bar.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
percent	Value in range [0.f , 1.f]. If it is beyond the acceptable range, it is automatically clamped

Return

void

14.1.14 ng_radio_button.h

```
#include "ng_gitem.h"
#include "ng_gestures.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_radio_button_t
```

More...

Macros

```
#define NG_RADIO_BUTTON
```

Functions

```
DRAW_FUNC(ng_radio_button_draw)
```

Draw function.

```
void ng_radio_button_toggle(tree_node_t *node)
```

Toggles all radio buttons inside a table.

```
void ng_radio_button_set_secondary_color(gitem_base_t *git,
uint32_t rgba)
```

Sets the secondary color.

14.1.14.1 Macro Definition Documentation

```
#define NG_RADIO_BUTTON
```

Type caster from base **gitem_base_t** struct to derived **gitem_radio_button_t** struct

14.1.14.2 Function Documentation

```
DRAW_FUNC (  ng_radio_button_draw )
```

Draw function.

Parameter	Description
*git	Pointer to radio button's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_radio_button_set_secondary_color (  gitem_base_t * git,
uint32_t rgba )
```

Sets the secondary color.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
rgba	Secondary (pressed) color value

Return

void

```
void ng_radio_button_toggle (  tree_node_t * node )
```

Toggles all radio buttons inside a table.

Parameter	Description
*node	Pointer to the radio button's parent tree node

Return

void

14.1.15 ng_rect.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_rect_t
```

More...

Macros

```
#define NG_RECT
```

Functions

```
DRAW_FUNC(ng_rect_draw)
```

Draw function.

14.1.15.1 Macro Definition Documentation

```
#define NG_RECT
```

Type caster from base **gitem_base_t** struct to derived **gitem_rect_t** struct

14.1.15.2 Function Documentation

```
DRAW_FUNC (  ng_rect_draw )
```

Draw function.

Parameter	Description
*git	Pointer to needle's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

14.1.16 ng_rounded_rect.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_rounded_rect_t
```

More...

Macros

```
#define NG_ROUNDED_RECT
```

Functions

```
DRAW_FUNC(ng_rounded_rect_draw)
```

Draw function.

14.1.16.1 Macro Definition Documentation

```
#define NG_ROUNDED_RECT
```

Type caster from base **gitem_base_t** struct to derived **gitem_rect_t** struct

14.1.16.2 Function Documentation

```
DRAW_FUNC (  ng_rounded_rect_draw )
```

Draw function.

Parameter	Description
*git	Pointer to needle's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

14.1.17 g_screen.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

Data Structures

```
struct gitem_screen_t
```

More...

Macros

```
#define NG_SCREEN
```

14.1.17.1 Functions

```
DRAW_FUNC(ng_screen_draw)
```

Draw function.

```
void ng_screen_set_image(gitem_base_t *git, void *asset_ptr)
```

Set image asset.

14.1.17.2 Macro Definition Documentation

```
#define NG_SCREEN
```

Type caster from base **gitem_base_t** struct to derived **gitem_screen_t** struct

14.1.17.3 Function Documentation

```
DRAW_FUNC (  ng_screen_draw )
```

Draw function.

Parameter	Description
*git	Pointer to screen's base gitem (gitem_base_t data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_screen_set_image (  gitem_base_t * git,  void * asset_ptr )
```

Set image asset.

Parameter	Description
*git	Pointer to **gitem_base_t** struct
*asset_ptr	Pointer to image asset (casted to **img_obj_t** internally)

Return

void

14.1.18 ng_slider.h

```
#include "ng_gitem.h"
```

Data Structures

```
struct gitem_slider_t
```

More...

Macros

```
#define NG_SLIDER
```

14.1.18.1 Functions

```
void ng_slider_set_percent(gitem_base_t *git, float percent)
```

Sets the current value (percent) of the slider.

```
void ng_slider_horizontal_set_indicator_x(gitem_base_t *git, int
x)
```

Sets the horizontal slider's indicator horizontal position.

```
void ng_slider_vertical_set_indicator_y(gitem_base_t *git, int y)
```

Sets the vertical slider's indicator vertical position.

14.1.18.2 Macro Definition Documentation

```
#define NG_SLIDER
```

Type caster from base **gitem_base_t** struct to derived **gitem_slider_t** struct

14.1.18.3 Function Documentation

```
void ng_slider_horizontal_set_indicator_x (gitem_base_t * git, int x)
```

Sets the horizontal slider's indicator horizontal position.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
x	Coordinate x (relative)

Return

void

```
void ng_slider_set_percent (  gitem_base_t * git,  float percent )
```

Sets the current value (percent) of the slider.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
percent	Value in range [0.f , 1.f]. If it is beyond the acceptable range, it is automatically clamped

Return

void

```
void ng_slider_vertical_set_indicator_y ( gitem_base_t * git, int y )
```

Sets the vertical slider's indicator vertical position.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)
y	Coordinate y (relative)

Return

void

14.1.19 ng_slider_hor.h

```
#include "ng_gitem.h"
#include "ng_utils.h"
```

14.1.20 ng_swipe_window.h

```
#include "ng_gitem.h"
#include "ng_globals.h"
#include "ng_event_transition.h"
#include "ng_tuples.h"
#include "ng_tree.h"
```

Data Structures

```
struct gitem_swipe_window_t
```

More...

Macros

```
#define NG_SWIPE_WINDOW
```

14.1.20.1 Macro Definition Documentation

```
#define NG_SWIPE_WINDOW
```

Type caster from base **gitem_base_t** struct to derived **gitem_swipe_window_t** struct

14.1.21 ng_toggle_button.h

Toggle button widget interface.

```
#include "ng_gitem.h"
```

Data Structures

```
struct gitem_toggle_button_t
```

More...

Macros

```
#define NG_TOGGLE_BUTTON
```

Functions

DRAW_FUNC(ng_toggle_button_draw)

Draw function.

Detailed Description

Toggle button widget interface.

14.1.21.1 Macro Definition Documentation

#define NG_TOGGLE_BUTTON

Type caster from base **gitem_base_t** struct to derived **gitem_toggle_button_t** struct

14.1.21.2 Function Documentation

DRAW_FUNC (ng_toggle_button_draw)

Draw function.

Parameter	Description
*git	Pointer to toggle button's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

14.1.22 ng_watchface.h

#include "ng_gitem.h"

Data Structures

struct gitem_watchface_t

More...

Macros

#define NG_WATCHFACE

Functions

DRAW_FUNC(ng_watchface_draw)

Draw function.

```
void ng_watchface_update(gitem_base_t *git)
```

Updates the watchface's hand angles according to the current time. System's wall time is used by default.

14.1.22.1 Macro Definition Documentation

```
#define NG_WATCHFACE
```

Type caster from base **gitem_base_t** struct to derived **gitem_watchface_t** struct

14.1.22.2 Function Documentation

```
DRAW_FUNC (  ng_watchface_draw )
```

Draw function.

Parameter	Description
*git	Pointer to watchface's base gitem (**gitem_base_t** data struct)
x_off	Horizontal offset from its parent item
y_off	Vertical offset from its parent item

Return

void

```
void ng_watchface_update (  gitem_base_t * git )
```

Updates the watchface's hand angles according to the current time. System's wall time is used by default.

In order to use a different time update method (not the system's wall time), the define **WALL_TIME_CLOCKS** (defined in the compiler flags of the generated Makefile) needs to be undefined and the time needs to be updated inside the #else segment of this function.

Parameter	Description
*git	Pointer to target gitem (**gitem_base_t** data struct)

Return

void

14.1.23 ng_window.h

```
#include "ng_gitem.h"
#include "ng_gestures.h"
```

Data Structures

```
struct gitem_window_t
```

More...

Macros

```
#define NG_WINDOW
```

Functions

```
void ng_window_set_source(tree_node_t *window, tree_node_t
*source)
```

Sets the source screen that the window displays.

14.1.23.1 Macro Definition Documentation

```
#define NG_WINDOW
```

Type caster from base **gitem_base_t** struct to derived **gitem_window_t** struct

14.1.23.2 Function Documentation

```
void ng_window_set_source (  tree_node_t * window,  tree_node_t *
source )
```

Sets the source screen that the window displays.

Parameter	Description
*window	Pointer to the window's tree node
*source	Pointer to the source screen's tree node

Return

void

14.2 Directories

Here is a list of all directories with brief descriptions:

14.2.1 File List

gitems ng_button.h
ng_checkbox.h
ng_circle.h
ng_circular_progress.h
ng_container.h
ng_digimeter.h
ng_digital_clock.h
ng_gauge.h ng_icon.h
ng_image.h
ng_label.h
ng_needle.h
ng_progress_bar.h
ng_radio_button.h
ng_rect.h
ng_rounded_rect.h
ng_screen.h
ng_slider.h
ng_slider_hor.h
ng_swipe_window.h

ng_toggle_button.h
Toggle button widget interface.

ng_watchface.h
ng_window.h

14.3 Data Structures

Here is a list of all data structures with brief descriptions:

14.3.1 gitem_button_t

Data Structure

#include <ng_button.h>

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct uint32_t secondary_color

Secondary color (color when button is pressed)

img_obj_t * primary_image

Pointer to primary image asset (image displayed when button is released)

img_obj_t * secondary_image

Pointer to secondary image asset (image displayed when button is pressed)

uint16_t pen_width

Pen width (when button is outlined)

Detailed Description

Button widget data struct. Acts as parent widget for label button and icon button

14.3.2 gitem_checkbox_t

Data Structure

```
#include <ng_checkbox.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct uint32_t secondary_color

Secondary color (color when checkbox is pressed)

img_obj_t * background_image

Pointer to background image asset

img_obj_t * foreground_image

Pointer to foreground image asset (e.g., a checkmark)

uint16_t pen_width

Pen width (for painting the checkbox outline)

Detailed Description

Checkbox widget data struct

14.3.3 gitem_circle_t

Data Structure

```
#include <ng_circle.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data struct

Detailed Description

Circle widget data struct

14.3.4 gitem_circular_progress_t

Data Structure

```
#include <ng_circular_progress.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct img_obj_t * background_image

Pointer to background image

asset img_obj_t * foreground_image

Pointer to foreground image asset (displayed on top of the background)

float value

Current value (0.f up to 1.f)

float max_angle

Maximum angle (span starts from min_angle up to max_angle)

float min_angle

Minimum angle (span starts from min_angle up to max_angle)

Detailed Description

Circular progress widget data struct

14.3.5 gitem_container_t

Data Structure

```
#include <ng_container.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct img_obj_t * image

Pointer to image asset

uint16_t pen_width

Pen width (when container is outlined)

Detailed Description

Container widget data struct

14.3.6 gitem_digimeter_t

Data Structure

```
#include <ng_digimeter.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct uint32_t text_color

Text color

float value

Current value

float max_value

Maximum value

float min_value

Minimum value

float step

Step (increase/decrease by)

nema_font_t * font

Pointer to font asset

uint32_t alignment

Horizontal/Vertical alignment

(bitfields) uint8_t dec_precision

Decimal digits

uint8_t int_precision

Integer digits. If this is bigger than zero, padding with zeros takes place on empty digits)

char * suffix

Suffix string

Detailed Description

Digital meter widget data struct

14.3.7 gitem_digital_clock_t

Data Structure

```
#include <ng_digital_clock.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct uint32_t text_color

Text color

nema_font_t * font

Pointer to font asset

uint32_t alignment

Horizontal/Vertical

alignment (bitfields) uint32_t time_format

Time format

Detailed Description

Digital clock data struct

14.3.8 gitem_gauge_t

Data Structure

```
#include <ng_gauge.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from gitem_base_t data

struct img_obj_t * image

Pointer to image asset

gitem_base_t * needle

Pointer to its "needle child

item" float value

Current value

float max_value

Maximum value

float min_value

Minimum value

float angle

Needle's current angle

float max_angle

Maximum angle

float min_angle

Minimum angle

int x_rot

Rotation x relative

coordinate int y_rot

Rotation y relative coordinate

uint16_t pen_width

Pen width (used when gauge is outlined)

Detailed Description

Gauge widget data struct

14.3.9 gitem_icon_t

Data Structure

```
#include <ng_icon.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct Icon widget data

struct (placeholder struct)

Detailed Description

Icon widget data struct (placeholder struct)

14.3.10 gitem_image_t

Data Structure

```
#include <ng_image.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct img_obj_t * image

Pointer to image asset

Detailed Description

Image widget data struct

14.3.11 gitem_label_t

Data Structure

```
#include <ng_label.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct attr_text_t * text

Pointer to its text attributes (string, font, index)

uint32_t text_color

Text color

Detailed Description

Label widget data struct

14.3.12 gitem_needle_t

Data Structure

```
#include <ng_needle.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct img_obj_t * image

Pointer to image asset

float angle

Current angle

int x_rot

Pivot x relative

coordinate int y_rot

Pivot y relative coordinate **uint16_t**

pen_width Pen width

Needle widget data struct

Detailed Description

Needle widget data struct

14.3.13 gitem_progress_bar_t

Data Structure

#include <ng_progress_bar.h>

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct uint32_t foreground_color

Foreground color

img_obj_t * background_image

Pointer to background image asset

img_obj_t * foreground_image

Pointer to foreground image

asset float value

Current value [0.f, 1.f]

uint16_t pen_width

Pen width

Detailed Description

Progress bar widget data struct

14.3.14 gitem_radio_button_t

Data Structure

```
#include <ng_radio_button.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct uint32_t secondary_color

Secondary (pressed) color

img_obj_t * background_image

Pointer to background (not selected) image

asset img_obj_t * foreground_image

Pointer to foreground (selected) image asset

uint16_t radius

Inner circle (if applicable) radius

Detailed Description

Radio button widget data struct

14.3.15 gitem_rect_t

Data Structure

```
#include <ng_rect.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t**

data struct uint16_t pen_width Pen width

Rectangle widget data struct

Detailed Description

Rectangle widget data struct

14.3.16 gitem_rounded_rect_t

Data Structure

```
#include <ng_rounded_rect.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t**

data struct uint16_t radius

Radius

Detailed Description

Rounded rectangle widget data struct

14.3.17 gitem_screen_t

Data Structure

```
#include <ng_screen.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct img_obj_t * image

Pointer to image asset

Detailed Description

Screen widget data struct. Each project must contain at least one screen

14.3.18 gitem_slider_t

Data Structure

```
#include <ng_slider.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct float value

Current value [0.f, 1.f]

gitem_base_t * progress

Pointer to its "progress" child item

gitem_base_t * indicator

Pointer to its "indicator" child item

Slider widget data struct

Detailed Description

Slider widget data struct

14.3.19 gitem_swipe_window_t

Data Structure

```
#include <ng_swipe_window.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct tree_node_t * indicators_parent

Pointer to the tree node placeholder of the indicators (if applicable)

uint16_t cur_page_index

Current page index

uint16_t page_count

Total page count

uint16_t spacing

Spacing between pages

uint8_t layout

Layout (horizontal or vertical)

ng_transition_t * animation

Pointer to a transition event (used for animating the swipe window)

ng_git_int_int_ez_t animation_data

Animation data (start point, end point and easing function)

Swipe widnow widget data struct

Detailed Description

Swipe window widget data struct

14.3.20 gitem_toggle_button_t

Data Structure

```
#include <ng_toggle_button.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct uint16_t cur_state

Current state's index

uint16_t max_state

State count **img_obj_t** ** images

Array of pointers to image assets, each image corresponds to each state

Toggle button widget data struct

Detailed Description

Toggle button widget data struct

14.3.21 gitem_watchface_t

Data Structure

```
#include <ng_watchface.h>
```

Data Fields

BASE_STRUCT

Inherited attributes from **gitem_base_t** data

struct img_obj_t * image

Pointer to image asset

gitem_base_t * hour

Pointer to its hours hand child item

(gitem_needle_t) gitem_base_t * minute

Pointer to its minutes hand child item

(gitem_needle_t) gitem_base_t * sec

Pointer to its seconds hand child item

(gitem_needle_t) uint16_t pen_width

Watchface widget data struct

Detailed Description

Watchface widget data struct

14.3.22 gitem_window_t

Data Structure

```
#include <ng_window.h>
```

Data Fields

BASE_STRUCT

Window widget data struct

Detailed Description

Window widget data struct

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 512. 879.2850

A-SOCAPG-UMGA04EN v1.0
December 2025