
NanoGUI Documentation

Release 0.1.0

Wenzel Jakob

Feb 26, 2017

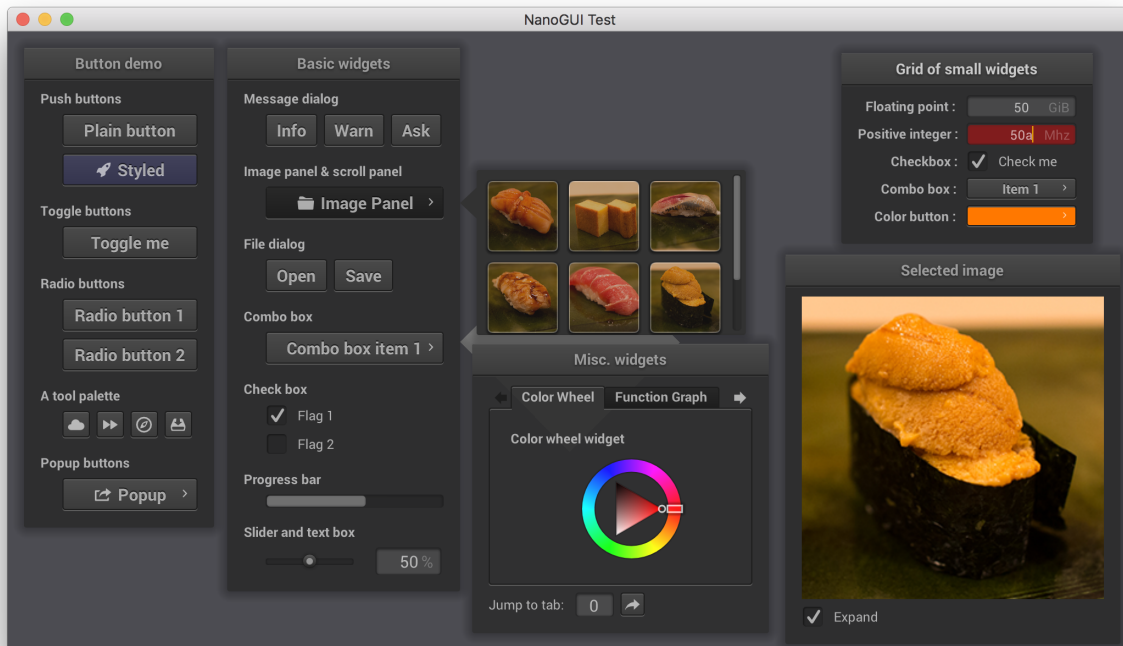
Contents

1	Example Screenshot	3
2	Description	5
3	“Simple mode”	7
4	License	9
5	Contents	11
5.1	Usage	11
5.2	Compilation	12
5.3	Examples	14
5.4	Library API	16
5.5	Contributing	192
6	Indices and tables	199

NanoGUI is a minimalistic cross-platform widget library for OpenGL 3.x or higher. It supports automatic layout generation, stateful C++11 lambdas callbacks, a variety of useful widget types and Retina-capable rendering on Apple devices thanks to [NanoVG](#) by Mikko Mononen. Python bindings of all functionality are provided using [pybind11](#).

CHAPTER 1

Example Screenshot



CHAPTER 2

Description

NanoGUI builds on [GLFW](#) for cross-platform OpenGL context creation and event handling, [GLAD](#) to use OpenGL 3.x or higher Windows, [Eigen](#) for basic vector types, and [NanoVG](#) to draw 2D primitives.

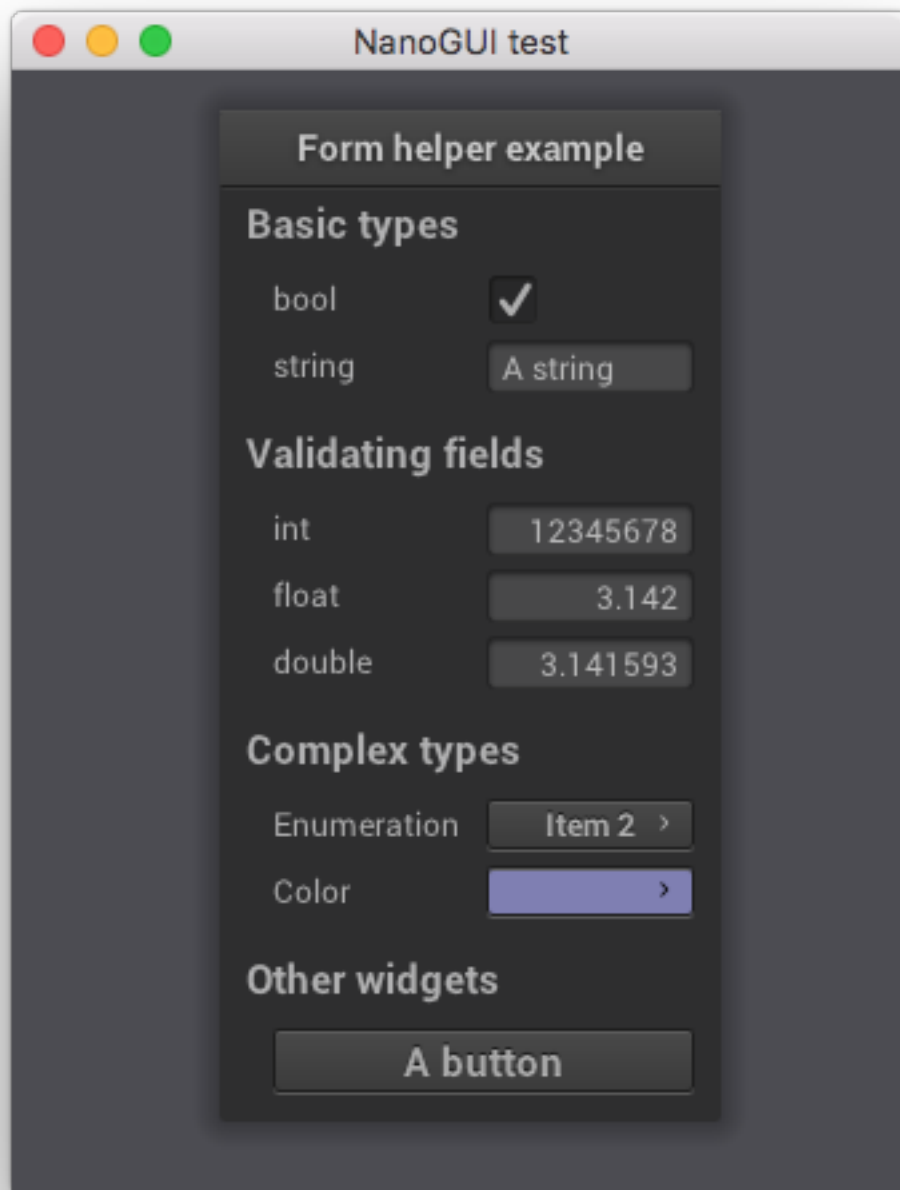
Note that the dependency library NanoVG already includes some basic example code to draw good-looking static widgets; what NanoGUI does is to flesh it out into a complete GUI toolkit with event handling, layout generation, etc.

NanoGUI currently works on Mac OS X (Clang) Linux (GCC or Clang) and Windows (Visual Studio 2015); it requires a recent C++11 capable compiler. All dependencies are jointly built using a CMake-based build system.

CHAPTER 3

“Simple mode”

Christian Schüller contributed a convenience class that makes it possible to create AntTweakBar-style variable manipulators using just a few lines of code. Refer to [Example 2](#) for how to create the image below.



CHAPTER 4

License

NanoGUI is provided under a BSD-style license that can be found in the [LICENSE](#) file. By using, distributing, or contributing to this project, you agree to the terms and conditions of this license.

Usage

C++

There are effectively two ways that you can use NanoGUI in C++: have NanoGUI initialize and manage the OpenGL context (and GLFW), or you do it manually.

1. If you are letting NanoGUI take over, you **must** call *Function `init`* before trying to do anything else. If you are managing OpenGL / GLFW yourself, make sure you **avoid** calling this method.
2. Create an instance of *Class `Screen`* (or a derivative class you have written).
 - NanoGUI managed OpenGL: call the explicit constructor.
 - **Self managed OpenGL: call the empty constructor.**
 - You must call the `nanogui::Screen::initialize()` method.
3. Add any Widgets, Buttons, etc. you want to the screen instance, and call the `nanogui::Screen::setVisible()` and `nanogui::Screen::performLayout()` methods of your instance.
4. Now that everything is ready, call *Function `mainloop`*.
5. When all windows are closed, this function will exit, and you should follow it up with a call to *Function `shutdown`*.

NanoGUI Managed OpenGL / GLFW Refer to *Example 2* for a concise example of what that all looks like.

Self Managed OpenGL / GLFW Refer to *Example 3* for an as concise as possible example of what you will need to do to get the *Class `Screen`* to work.

Python

The Python interface is very similar to the C++ API. When you build NanoGUI with CMake, a `python` folder is created with the library you `import nanogui` from. Though there are implementation details that differ greatly, the documentation and build process for the Python side is roughly the same. Refer to the [Examples](#) and compare the source code for the two.

Example 3 highlights the more notable differences between the APIs. Specifically, that managing GLFW from Python has no meaning, as well as the main loop for Python can easily be detached.

Compilation

NanoGUI uses a CMake build system to ensure portability. All dependencies are cloned and compiled in one batch, which should generally reduce the amount of configuration effort to zero. Assuming that NanoGUI was cloned into the current working directory, the following commands need to be executed:

```
# enter the top-level NanoGUI directory
$ cd nanogui

# make a build directory and enter that
$ mkdir build
$ cd build

# generate your Makefile
$ cmake ..

# now that you have a Makefile, use that to build
$ make -j 4
```

For Windows, the process is nearly the same:

```
# enter the top-level NanoGUI directory
$ cd nanogui

# make a build directory and enter that
$ mkdir build
$ cd build

# Specify VS Version AND 64bit, otherwise it defaults to 32.
# The version number and year may be different for you, Win64
# can be appended to any of them.
# 32 bit Windows builds are /not/ supported
$ cmake -G "Visual Studio 14 2015 Win64" ..

# Either open the .sln with Visual Studio, or run
$ cmake --build . --config Release
```

Default Configurations

By default, NanoGUI will

Impact / effect	CMake Option
Build the example programs.	NANOGUI_BUILD_EXAMPLE
Build as a <i>shared</i> library.	NANOGUI_BUILD_SHARED
Build the Python plugins.	NANOGUI_BUILD_PYTHON
Use GLAD if on Windows.	NANOGUI_USE_GLAD
Generate an install target.	NANOGUI_INSTALL

Users developing projects that reference NanoGUI as a `git` submodule (this is **strongly** encouraged) can set up the parent project's CMake configuration file as follows (this assumes that `nanogui` lives in the directory `ext/nanogui` relative to the parent project):

```
# Disable building extras we won't need (pure C++ project)
set(NANOGUI_BUILD_EXAMPLE OFF CACHE BOOL " " FORCE)
set(NANOGUI_BUILD_PYTHON  OFF CACHE BOOL " " FORCE)
set(NANOGUI_INSTALL       OFF CACHE BOOL " " FORCE)

# Add the configurations from nanogui
add_subdirectory(ext/nanogui)

# For reliability of parallel build, make the NanoGUI targets dependencies
set_property(TARGET glfw glfw_objects nanogui PROPERTY FOLDER "dependencies")
```

Required Variables Exposed

Due to the nature of building an OpenGL application for different platforms, three variables are populated to allow for easy incorporation with your CMake build. After you have executed `add_subdirectory` as shown above, you will need to add the following (assuming the target you are building is called `myTarget`):

```
# Various preprocessor definitions have been generated by NanoGUI
add_definitions(${NANOGUI_EXTRA_DEFS})

# On top of adding the path to nanogui/include, you may need extras
include_directories(${NANOGUI_EXTRA_INCS})

# Compile a target using NanoGUI
add_executable(myTarget myTarget.cpp)

# Lastly, additional libraries may have been built for you. In addition to linking
# against NanoGUI, we need to link against those as well.
target_link_libraries(myTarget nanogui ${NANOGUI_EXTRA_LIBS})
```

Compiling the Documentation

The documentation system relies on ‘Doxygen’, ‘Sphinx’, ‘Breathe’, and ‘Exhale’. It uses the ‘Read the Docs’ theme for the layout of the generated html. So you will need to first

1. Install Doxygen for your operating system. On Unix based systems, this should be available through your package manager (`apt-get`, `brew`, `dnf`, etc).
2. Install Sphinx, Breathe, and the theme:

```
pip install breathe sphinx_rtd_theme
```

Now that you have the relevant tools, you can build the documentation with

```
# Enter the documentation directory
$ cd <path/to/nanogui>/docs

# Build the documentation
$ make html
```

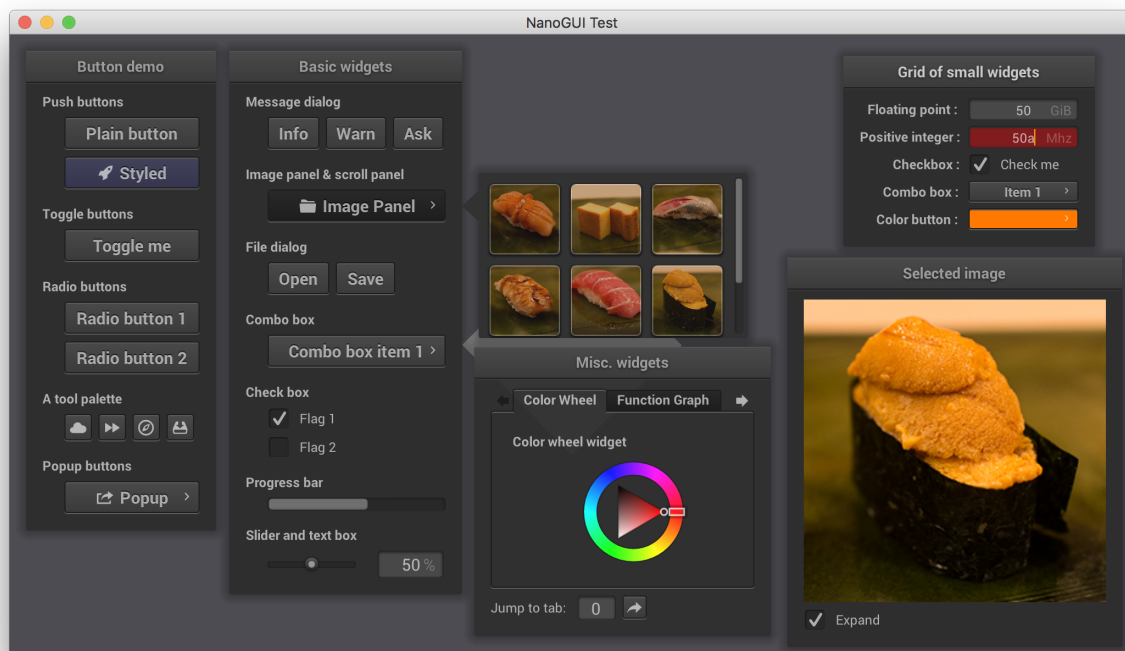
The output will be generated in `_build`, the root html document is located at `_build/html/index.html`.

Note: When building the documentation locally, there can be subtle differences in the rendered pages than what is hosted online. You should largely be able to ignore this.

Examples

There are example programs available for you to play with / understand how the different pieces fit together. The C++ examples are in `nanogui/src/`, and the equivalent Python examples are in `nanogui/python`.

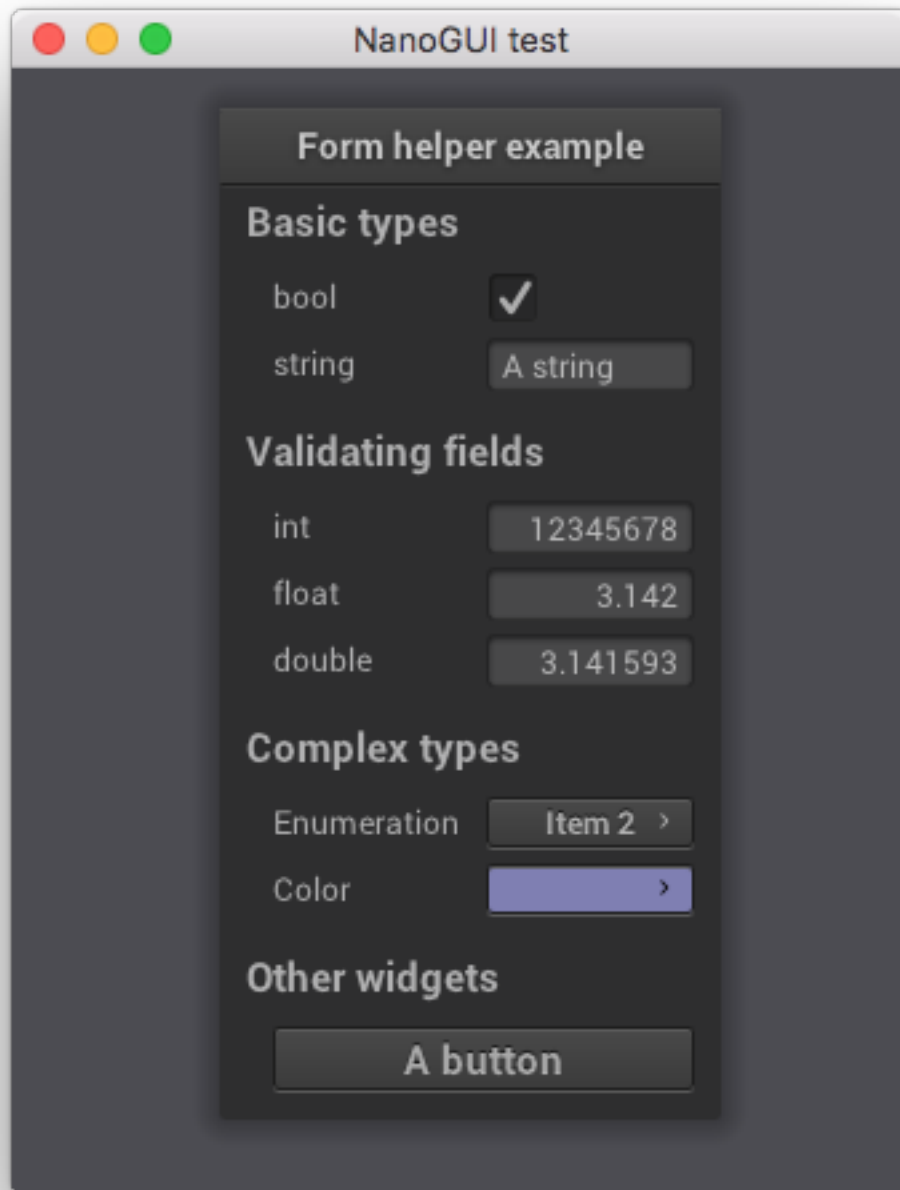
Example 1



The first example program is rather long, but not altogether complicated. It is effectively an enumeration of how one would go about adding various different kinds of Widgets to the window being displayed.

- [Example 1 in C++](#)
- [Example 1 in Python](#)

Example 2



The second program demonstrates how simple label/editor widget-type layouts can be written in a very concise manner.

- [Example 2 in C++](#)
- [Example 2 in Python](#)

Example 3

The third example program demonstrates how to manage OpenGL / GLFW on your own for the C++ side, and how to detach the NanoGUI `mainloop()` on the Python side.

- [Example 3 in C++](#)
- [Example 3 in Python](#)

Example 4

The fourth example program demonstrates the GLCanvas widget, which renders an arbitrary sequence of OpenGL commands into a NanoGUI widget.

- [Example 4 in C++](#)
- [Example 4 in Python](#)

Example Repository

Darren Mothersele has put together a compact and informative example repository that demonstrates how easy it is to include NanoGUI into your project. You download / browse the source on [GitHub](#).

Library API

Welcome to the developer reference to NanoGUI. The documentation is actively being developed / updated. If you would like to help document any part of the project you may be familiar with, please refer to the [Contributing](#) page.

Note: Presented below is only the C++ API. If you are using the Python API, the contents below are still applicable for understanding what methods are available. While the documentation for the C++ API is useful as a reference for understanding what a given class does, the Python API does differ. Please refer to the more concise [Example 2](#) for comparing the differences between the C++ and Python interfaces.

Class Hierarchy

File Hierarchy

Full API

Namespaces

Namespace `nanogui`

Namespaces

- [*Namespace `nanogui::detail`*](#)

Classes

- *Struct AdvancedGridLayout::Anchor*
- *Struct Arcball*
- *Struct GLShader::Buffer*
- *Struct TabButton::StringView*
- *Class AdvancedGridLayout*
- *Class BoxLayout*
- *Class Button*
- *Class CheckBox*
- *Class Color*
- *Class ColorPicker*
- *Class ColorWheel*
- *Class ComboBox*
- *Class FloatBox*
- *Class FormHelper*
- *Class GLCanvas*
- *Class GLFramebuffer*
- *Class GLShader*
- *Class GLUniformBuffer*
- *Class Graph*
- *Class GridLayout*
- *Class GroupLayout*
- *Class ImagePanel*
- *Class ImageView*
- *Class IntBox*
- *Class Label*
- *Class Layout*
- *Class MessageDialog*
- *Class Object*
- *Class Popup*
- *Class PopupButton*
- *Class ProgressBar*
- *Class ref*
- *Class Screen*
- *Class Serializer*
- *Class Slider*

- *Class StackedWidget*
- *Class TabHeader*
- *Class TabHeader::TabButton*
- *Class TabWidget*
- *Class TextBox*
- *Class Theme*
- *Class ToolButton*
- *Class UniformBufferStd140*
- *Class VScrollPanel*
- *Class Widget*
- *Class Window*

Enums

- *Enum TextBox::Alignment*
- *Enum TabHeader::ClickLocation*
- *Enum Button::Flags*
- *Enum Button::IconPosition*
- *Enum Alignment*
- *Enum Cursor*
- *Enum Orientation*
- *Enum ColorWheel::Region*
- *Enum Popup::Side*
- *Enum TextBox::SpinArea*
- *Enum MessageDialog::Type*

Functions

- *Function __nanogui_get_image*
- *Function active*
- *Function chdir_to_bundle_parent*
- *Function file_dialog*
- *Function frustum*
- *Function init*
- *Function leave*
- *Function loadImageDirectory*
- *Function lookAt*

- *Function mainloop*
- *Function nvglIsFontIcon*
- *Function nvglIsImageIcon*
- *Function ortho*
- *Function project*
- *Function scale*
- *Function shutdown*
- *Function translate*
- *Function unproject*
- *Function utf8*

Typedefs

- *Typedef nanogui::MatrixXu*

Namespace nanogui::detail

Classes

- *Struct serialization_helper*
- *Struct serialization_traits*
- *Class FormWidget*

Classes and Structs

Struct AdvancedGridLayout::Anchor

- Defined in *File layout.h*
- Nested type of *Class AdvancedGridLayout*

struct nanogui::AdvancedGridLayout::Anchor
Helper struct to coordinate anchor points for the layout.

Public Functions

Anchor ()

Anchor (int x, int y, *Alignment* horiz = Alignment::Fill, *Alignment* vert = Alignment::Fill)

Anchor (int x, int y, int w, int h, *Alignment* horiz = Alignment::Fill, *Alignment* vert = Alignment::Fill)

operator std::string () const

Public Members

uint8_t **pos**[2]

uint8_t **size**[2]

Alignment **align**[2]

Struct Arcball

- Defined in *File glutil.h*

struct nanogui::**Arcball**

Arcball helper class to interactively rotate objects on-screen.

Public Functions

Arcball (float *speedFactor* = 2.0f)

Arcball (const Quaternionf &*quat*)

Quaternionf &**state** ()

void **setState** (const Quaternionf &*state*)

void **setSize** (Vector2i *size*)

const Vector2i &**size** () const

void **setSpeedFactor** (float *speedFactor*)

float **speedFactor** () const

bool **active** () const

void **button** (Vector2i *pos*, bool *pressed*)

bool **motion** (Vector2i *pos*)

Matrix4f **matrix** () const

Protected Attributes

bool **mActive**

Vector2i **mLastPos**

Vector2i **mSize**

Quaternionf **mQuat**

Quaternionf **mIncr**

float **mSpeedFactor**

Struct `serialization_helper`

- Defined in *File core.h*

template <typename T>

struct nanogui::detail::**serialization_helper**

The primary serialization helper class; preliminary specializations are in *File core.h*, see *Class Serializer*.

Struct `serialization_traits`

- Defined in *File core.h*

template <typename T, typename SFINAE = void>

struct nanogui::detail::**serialization_traits**

SFINAE helper struct for generic traits serialization.

Must be fully specialized for any type that needs to be serialized.

Template Parameters

- T: The type to explicitly be serialized.

Struct `GLShader::Buffer`

- Defined in *File glutil.h*
- Nested type of *Class GLShader*

struct nanogui::GLShader::**Buffer**

A wrapper struct for maintaining various aspects of items being managed by OpenGL.

Public Members

GLuint **id**

GLuint **glType**

GLuint **dim**

GLuint **compSize**

GLuint **size**

int **version**

Struct `TabButton::StringView`

- Defined in *File tabheader.h*
- Nested type of *Class TabHeader::TabButton*

struct nanogui::TabHeader::TabButton::**StringView**

Helper struct to represent the TabButton.

Public Members

`const char *first` = nullptr

`const char *last` = nullptr

Class AdvancedGridLayout

- Defined in *File layout.h*

Nested Types:

- *Struct AdvancedGridLayout::Anchor*

class nanogui::AdvancedGridLayout
Advanced Grid layout.

The is a fancier grid layout with support for items that span multiple rows or columns, and per-widget alignment flags. Each row and column additionally stores a stretch factor that controls how additional space is redistributed. The downside of this flexibility is that a layout anchor data structure must be provided for each widget.

An example:

```
using AdvancedGridLayout::Anchor;
Label *label = new Label(window, "A label");
// Add a centered label at grid position (1, 5), which spans two horizontal cells
layout->setAnchor(label, Anchor(1, 5, 2, 1, Alignment::Middle,
↪Alignment::Middle));
```

The grid is initialized with user-specified column and row size vectors (which can be expanded later on if desired). If a size value of zero is specified for a column or row, the size is set to the maximum preferred size of any widgets contained in the same row or column. Any remaining space is redistributed according to the row and column stretch factors.

The high level usage somewhat resembles the classic HIG layout:

- <https://web.archive.org/web/20070813221705/http://www.autel.cz/dmi/tutorial.html>
- <https://github.com/jaapgeurts/higlayout>

Inherits from *nanogui::Layout*

Public Functions

AdvancedGridLayout (`const` std::vector<int> &*cols* = {}, `const` std::vector<int> &*rows* = {}, int *margin* = 0)

int **margin** () `const`

void **setMargin** (int *margin*)

int **colCount** () `const`
Return the number of cols.

int **rowCount** () `const`
Return the number of rows.

```

void appendRow (int size, float stretch = 0.f)
    Append a row of the given size (and stretch factor)

void appendCol (int size, float stretch = 0.f)
    Append a column of the given size (and stretch factor)

void setRowStretch (int index, float stretch)
    Set the stretch factor of a given row.

void setColStretch (int index, float stretch)
    Set the stretch factor of a given column.

void setAnchor (const Widget *widget, const Anchor &anchor)
    Specify the anchor data structure for a given widget.

Anchor anchor (const Widget *widget) const
    Retrieve the anchor data structure for a given widget.

virtual Vector2i preferredSize (NVGcontext *ctx, const Widget *widget) const

virtual void performLayout (NVGcontext *ctx, Widget *widget) const

```

Protected Functions

```

void computeLayout (NVGcontext *ctx, const Widget *widget, std::vector<int> *grid) const

```

Protected Attributes

```

std::vector<int> mCols
std::vector<int> mRows
std::vector<float> mColStretch
std::vector<float> mRowStretch
std::unordered_map<const Widget *, Anchor> mAnchor
int mMargin

struct Anchor
    Helper struct to coordinate anchor points for the layout.

```

Public Functions

```

Anchor ()

Anchor (int x, int y, Alignment horiz = Alignment::Fill, Alignment vert = Alignment::Fill)

Anchor (int x, int y, int w, int h, Alignment horiz = Alignment::Fill, Alignment vert = Alignment::Fill)

operator std::string() const

```

Public Members

uint8_t **pos**[2]
uint8_t **size**[2]
Alignment **align**[2]

Class BoxLayout

- Defined in *File layout.h*

class nanogui::**BoxLayout**

Simple horizontal/vertical box layout.

This widget stacks up a bunch of widgets horizontally or vertically. It adds margins around the entire container and a custom spacing between adjacent widgets.

Inherits from *nanogui::Layout*

Public Functions

BoxLayout (*Orientation* orientation, *Alignment* alignment = Alignment::Middle, int margin = 0, int spacing = 0)

Construct a box layout which packs widgets in the given Orientation.

Parameters

- alignment: *Widget* alignment perpendicular to the chosen orientation
- margin: Margin around the layout container
- spacing: Extra spacing placed between widgets

Orientation **orientation** () **const**

void **setOrientation** (*Orientation* orientation)

Alignment **alignment** () **const**

void **setAlignment** (*Alignment* alignment)

int **margin** () **const**

void **setMargin** (int margin)

int **spacing** () **const**

void **setSpacing** (int spacing)

virtual Vector2i **preferredSize** (NVGcontext *ctx, **const** *Widget* *widget) **const**

virtual void **performLayout** (NVGcontext *ctx, *Widget* *widget) **const**

Protected Attributes

Orientation **mOrientation**

Alignment **mAlignment**

int mMargin

int mSpacing

Class Button

- Defined in *File button.h*

Nested Types:

- *Enum Button::Flags*
- *Enum Button::IconPosition*

class nanogui::Button

[Normal/Toggle/Radio/Popup] *Button* widget.

Inherits from *nanogui::Widget*

Subclassed by *nanogui::PopupButton*, *nanogui::ToolButton*

Public Types

enum Flags

Flags to specify the button behavior (can be combined with binary OR)

Values:

NormalButton = (1 << 0)

RadioButton = (1 << 1)

ToggleButton = (1 << 2)

PopupButton = (1 << 3)

enum IconPosition

The available icon positions.

Values:

Left

LeftCentered

RightCentered

Right

Public Functions

Button (*Widget* *parent, **const** std::string &caption = “Untitled”, int icon = 0)

const std::string &caption () **const**

```
void setCaption (const std::string &caption)

const Color &backgroundColor () const

void setBackgroundColor (const Color &backgroundColor)

const Color &textColor () const

void setTextColor (const Color &textColor)

int icon () const

void setIcon (int icon)

int flags () const

void setFlags (int buttonFlags)

IconPosition iconPosition () const

void setIconPosition (IconPosition iconPosition)

bool pushed () const

void setPushed (bool pushed)

std::function<void ()> callback
    const Set the push callback (for any type of button)

void setCallback (const std::function<void>
    > &callback)

std::function<void (bool)> changeCallback
    const Set the change callback (for toggle buttons)

void setChangeCallback (const std::function<void> bool
    > &callback)

void setButtonGroup (const std::vector<Button *> &buttonGroup)
    Set the button group (for radio buttons)

const std::vector<Button *> &buttonGroup () const

virtual Vector2i preferredSize (NVGcontext *ctx) const
    Compute the preferred size of the widget.

virtual bool mouseButtonEvent (const Vector2i &p, int button, bool down, int modifiers)
    Handle a mouse button event (default implementation: propagate to children)

virtual void draw (NVGcontext *ctx)
    Draw the widget (and all child widgets)

virtual void save (Serializer &s) const
    Save the state of the widget into the given Serializer instance.

virtual bool load (Serializer &s)
    Restore the state of the widget from the given Serializer instance.
```

Protected Attributes

```
std::string mCaption
int mIcon
IconPosition mIconPosition
bool mPushed
int mFlags
Color mBackgroundColor
Color mTextColor
std::function<void ()> mCallback
std::function<void (bool)> mChangeCallback
std::vector<Button *> mButtonGroup
```

Class CheckBox

- Defined in *File checkbox.h*

class nanogui::**CheckBox**

Two-state check box widget.

Inherits from *nanogui::Widget*

Public Functions

CheckBox (*Widget* *parent, **const** std::string &caption = “Untitled”, **const** std::function<void> bool
> &callback = std::function< void(bool)>())

const std::string &caption () **const**

void **setCaption** (**const** std::string &caption)

const bool &checked () **const**

void **setChecked** (**const** bool &checked)

const bool &pushed () **const**

void **setPushed** (**const** bool &pushed)

std::function<void (bool)> **callback**
const

void **setCallback** (**const** std::function<void> bool
> &callback)

virtual bool **mouseButtonEvent** (**const** Vector2i &p, int button, bool down, int modifiers)
Handle a mouse button event (default implementation: propagate to children)

virtual Vector2i **preferredSize** (NVGcontext *ctx) **const**
Compute the preferred size of the widget.

virtual void draw (NVGcontext *ctx)

Draw the widget (and all child widgets)

virtual void save (*Serializer &s*) **const**

Save the state of the widget into the given *Serializer* instance.

virtual bool load (*Serializer &s*)

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

std::string **mCaption**

bool **mPushed**

bool **mChecked**

std::function<void (bool)> **mCallback**

Class Color

- Defined in *File common.h*

class nanogui::Color

Stores an RGBA floating point color value.

This class simply wraps around an `Eigen::Vector4f`, providing some convenient methods and terminology for thinking of it as a color. The data operates in the same way as `Eigen::Vector4f`, and the following values are identical:

Channel	Array Index	Eigen::Vector4f Value	Color Value
Red	0	x()	r()
Green	1	y()	g()
Blue	2	z()	b()
Alpha	3	w()	w()

Note: The method for the alpha component is **always** `w()`.

You can and should still use the various convenience methods such as `any()`, `all()`, `head<index>()`, etc provided by Eigen.

Inherits from `Vector4f`

Public Functions

Color ()

Default constructor: represents black (`r, g, b, a = 0`)

Color (const Eigen::Vector4f &color)

Makes an exact copy of the data represented by the input parameter.

Parameters

- `color`: The four dimensional float vector being copied.

Color (**const** Eigen::Vector3f &*color*, float *alpha*)

Copies (x, y, z) from the input vector, and uses the value specified by the *alpha* parameter for this *Color* object's alpha component.

Parameters

- *color*: The three dimensional float vector being copied.
- *alpha*: The value to set this object's alpha component to.

Color (**const** Eigen::Vector3i &*color*, int *alpha*)

Copies (x, y, z) from the input vector, casted as floats first and then divided by 255.0, and uses the value specified by the *alpha* parameter, casted to a float and divided by 255.0 as well, for this *Color* object's alpha component.

Parameters

- *color*: The three dimensional integer vector being copied, will be divided by 255.0.
- *alpha*: The value to set this object's alpha component to, will be divided by 255.0.

Color (**const** Eigen::Vector3f &*color*)

Copies (x, y, z) from the input vector, and sets the alpha of this color to be 1.0.

Parameters

- *color*: The three dimensional float vector being copied.

Color (**const** Eigen::Vector3i &*color*)

Copies (x, y, z) from the input vector, casting to floats and dividing by 255.0. The alpha of this color will be set to 1.0.

Parameters

- *color*: The three dimensional integer vector being copied, will be divided by 255.0.

Color (**const** Eigen::Vector4i &*color*)

Copies (x, y, z, w) from the input vector, casting to floats and dividing by 255.0.

Parameters

- *color*: The three dimensional integer vector being copied, will be divided by 255.0.

Color (float *intensity*, float *alpha*)

Creates the *Color* (*intensity*, *intensity*, *intensity*, *alpha*).

Parameters

- *intensity*: The value to be used for red, green, and blue.
- *alpha*: The alpha component of the color.

Color (int *intensity*, int *alpha*)

Creates the *Color* (*intensity*, *intensity*, *intensity*, *alpha*) / 255.0. Values are casted to floats before division.

Parameters

- `intensity`: The value to be used for red, green, and blue, will be divided by 255.0.
- `alpha`: The alpha component of the color, will be divided by 255.0.

Color (float *r*, float *g*, float *b*, float *a*)

Explicit constructor: creates the *Color* (*r*, *g*, *b*, *a*).

Parameters

- *r*: The red component of the color.
- *g*: The green component of the color.
- *b*: The blue component of the color.
- *a*: The alpha component of the color.

Color (int *r*, int *g*, int *b*, int *a*)

Explicit constructor: creates the *Color* (*r*, *g*, *b*, *a*) / 255.0. Values are casted to floats before division.

Parameters

- *r*: The red component of the color, will be divided by 255.0.
- *g*: The green component of the color, will be divided by 255.0.
- *b*: The blue component of the color, will be divided by 255.0.
- *a*: The alpha component of the color, will be divided by 255.0.

template <typename Derived>

Color (const Eigen::MatrixBase<Derived> &*p*)

Construct a color vector from MatrixBase (needed to play nice with Eigen)

template <typename Derived>

Color &**operator=** (const Eigen::MatrixBase<Derived> &*p*)

Assign a color vector from MatrixBase (needed to play nice with Eigen)

float &**r** ()

Return a reference to the red channel.

const float &**r** () **const**

Return a reference to the red channel (const version)

float &**g** ()

Return a reference to the green channel.

const float &**g** () **const**

Return a reference to the green channel (const version)

float &**b** ()

Return a reference to the blue channel.

const float &**b** () **const**

Return a reference to the blue channel (const version)

Color **contrastingColor** () **const**

Computes the luminance as $l = 0.299r + 0.587g + 0.144b + 0.0a$. If the luminance is less than 0.5, white is returned. If the luminance is greater than or equal to 0.5, black is returned. Both returns will have an alpha component of 1.0.

operator const NVGcolor& () const

Allows for conversion between this *Color* and NanoVG's representation.

Allows for conversion between *nanogui::Color* and the NanoVG NVGcolor class.

Class ColorPicker

- Defined in *File colorpicker.h*

class nanogui::ColorPicker

Push button with a popup to tweak a color value.

Inherits from *nanogui::PopupButton*

Public Functions

ColorPicker (*Widget* *parent, const *Color* &color = Color (1.0f, 0.0f, 0.0f, 1.0f))

std::function<void (const *Color*&) > **callback**
const Set the change callback.

void **setCallback** (const std::function<void> const *Color*&
 > &callback

Color **color** () const
 Get the current color.

void **setColor** (const *Color* &color)
 Set the current color.

Protected Attributes

std::function<void (const *Color*&) > **mCallback**

ColorWheel ***mColorWheel**

Button ***mPickButton**

Class ColorWheel

- Defined in *File colorwheel.h*

Nested Types:

- *Enum ColorWheel::Region*

class nanogui::ColorWheel

Fancy analog widget to select a color value.

Inherits from *nanogui::Widget*

Public Functions

ColorWheel (*Widget* *parent, const *Color* &color = Color (1.0f, 0.0f, 0.0f, 1.0f))

std::function<void (const *Color*&) > **callback**
const Set the change callback.

void **setCallback** (const std::function<void> const *Color*&
> &callback

Color **color** () const
Get the current color.

void **setColor** (const *Color* &color)
Set the current color.

virtual Vector2i **preferredSize** (NVGcontext *ctx) const
Compute the preferred size of the widget.

virtual void **draw** (NVGcontext *ctx)
Draw the widget (and all child widgets)

virtual bool **mouseButtonEvent** (const Vector2i &p, int button, bool down, int modifiers)
Handle a mouse button event (default implementation: propagate to children)

virtual bool **mouseDragEvent** (const Vector2i &p, const Vector2i &rel, int button, int modifiers)
Handle a mouse drag event (default implementation: do nothing)

virtual void **save** (*Serializer* &s) const
Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)
Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

float **mHue**

float **mWhite**

float **mBlack**

Region **mDragRegion**

std::function<void (const *Color*&) > **mCallback**

Class ComboBox

- Defined in *File combobox.h*

class nanogui : **ComboBox**
Simple combo box widget based on a popup button.
Inherits from *nanogui::PopupButton*

Public Functions

ComboBox (*Widget* *parent)

Create an empty combo box.

ComboBox (*Widget* *parent, const std::vector<std::string> &items)

Create a new combo box with the given items.

ComboBox (*Widget* *parent, const std::vector<std::string> &items, const std::vector<std::string> &itemsShort)

Create a new combo box with the given items, providing both short and long descriptive labels for each item.

std::function<void (int)> **callback**
const

void **setCallback** (const std::function<void> int
> &callback

int **selectedIndex** () const

void **setSelectedIndex** (int idx)

void **setItems** (const std::vector<std::string> &items, const std::vector<std::string> &itemsShort)

void **setItems** (const std::vector<std::string> &items)

const std::vector<std::string> &**items** () const

const std::vector<std::string> &**itemsShort** () const

virtual bool **scrollEvent** (const Vector2i &p, const Vector2f &rel)

Handle a mouse scroll event (default implementation: propagate to children)

virtual void **save** (*Serializer* &s) const

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

std::vector<std::string> **mItems**

std::vector<std::string> **mItemsShort**

std::function<void (int)> **mCallback**

int **mSelectedIndex**

Class FormWidget

- Defined in *File formhelper.h*

template <typename T, typename sfinae = std::true_type>

class `nanogui::detail::FormWidget`

A template wrapper class for assisting in the creation of various form widgets.

The partial template specializations are:

- Inheritance from *Class ComboBox* for enum types:

```
template <typename T>
class FormWidget<T, typename std::is_enum<T>::type> : public ComboBox
```

- Inheritance from *Class IntBox* for integral types:

```
template <typename T>
class FormWidget<T, typename std::is_integral<T>::type> : public IntBox<T>
```

- Inheritance from *Class FloatBox* for floating point types:

```
template <typename T>
class FormWidget<T, typename std::is_floating_point<T>::type> : public_
↳FloatBox<T>
```

The full template specializations are:

- Inheritance from *Class CheckBox* for booleans:

```
template <>
class FormWidget<bool, std::true_type> : public CheckBox
```

- Inheritance from *Class TextBox* for strings:

```
template <>
class FormWidget<std::string, std::true_type> : public TextBox
```

- Inheritance from *Class ColorPicker* for *Color* types:

```
template <>
class FormWidget<Color, std::true_type> : public ColorPicker
```

Please refer to the bottom of *Program Listing for File formhelper.h* for the implementation details.

Class FloatBox

- Defined in *File textbox.h*

template <typename Scalar>

class `nanogui::FloatBox`

A specialization of *TextBox* representing floating point values.

Template parameters should be float types, e.g. `float`, `double`, `float64_t`, etc.

Inherits from *nanogui::TextBox*

Public Functions

FloatBox (*Widget* *parent, Scalar value = (Scalar) 0.f)

`std::string numberFormat () const`

```

void numberFormat (const std::string &format)

Scalar value () const

void setValue (Scalar value)

void setCallback (const std::function<void> Scalar
    > &cb)

void setValueIncrement (Scalar incr)

void setMinValue (Scalar minValue)

void setMaxValue (Scalar maxValue)

void setMinMaxValues (Scalar minValue, Scalar maxValue)

virtual bool mouseButtonEvent (const Vector2i &p, int button, bool down, int modifiers)
    Handle a mouse button event (default implementation: propagate to children)

virtual bool mouseDragEvent (const Vector2i &p, const Vector2i &rel, int button, int modifiers)
    Handle a mouse drag event (default implementation: do nothing)

virtual bool scrollEvent (const Vector2i &p, const Vector2f &rel)
    Handle a mouse scroll event (default implementation: propagate to children)

```

Class FormHelper

- Defined in *File formhelper.h*

class nanogui::FormHelper

Convenience class to create simple AntTweakBar-style layouts that expose variables of various types using NanoGUI widgets.

Example:

```

// [ ... initialize NanoGUI, construct screen ... ]

FormHelper* h = new FormHelper(screen);

// Add a new windows widget
h->addWindow(Eigen::Vector2i(10,10), "Menu");

// Start a new group
h->addGroup("Group 1");

// Expose an integer variable by reference
h->addVariable("integer variable", aInt);

// Expose a float variable via setter/getter functions
h->addVariable(
    [&](float value) { aFloat = value; },
    [&]() { return *aFloat; },
    "float variable");

// add a new button
h->addButton("Button", [&]() { std::cout << "Button pressed" << std::endl; });

```

Public Functions

FormHelper (*Screen* *screen)

Create a helper class to construct NanoGUI widgets on the given screen.

Window *addWindow (const Vector2i &pos, const std::string &title = “Untitled”)

Add a new top-level window.

Label *addGroup (const std::string &caption)

Add a new group that may contain several sub-widgets.

template <typename Type>

detail::FormWidget<Type> *addVariable (const std::string &label, const std::function<void> const
Type&

> &setter, const std::function<Type> &getter bool editable = true) Add a new data widget controlled using custom getter/setter functions.

template <typename Type>

detail::FormWidget<Type> *addVariable (const std::string &label, Type &value, bool editable =
true)

Add a new data widget that exposes a raw variable in memory.

Button *addButton (const std::string &label, const std::function<void>

> &cb) Add a button with a custom callback.

void addWidget (const std::string &label, *Widget* *widget)

Add an arbitrary (optionally labeled) widget to the layout.

void refresh ()

Cause all widgets to re-synchronize with the underlying variable state.

Window *window ()

Access the currently active *Window* instance.

void setWindow (*Window* *window)

void setFixedSize (const Vector2i &fw)

Specify a fixed size for newly added widgets.

Vector2i fixedSize ()

const std::string &groupFontName () const

void setGroupFontName (const std::string &name)

const std::string &labelFontName () const

void setLabelFontName (const std::string &name)

int groupFontSize () const

void setGroupFontSize (int value)

int labelFontSize () const

void setLabelFontSize (int value)

int widgetFontSize () const

void setWidgetFontSize (int value)

Protected Attributes

```

ref<Screen> mScreen
ref<Window> mWindow
ref<AdvancedGridLayout> mLayout
std::vector<std::function<void () >> mRefreshCallbacks
std::string mGroupFontName = "sans-bold"
std::string mLabelFontName = "sans"
Vector2i mFixedSize = Vector2i(0, 20)
int mGroupFontSize = 20
int mLabelFontSize = 16
int mWidgetFontSize = 16
int mPreGroupSpacing = 15
int mPostGroupSpacing = 5
int mVariableSpacing = 5

```

Class GLCanvas

- Defined in *File glcanvas.h*

class nanogui::GLCanvas

Canvas widget for rendering OpenGL content.

Canvas widget that can be used to display arbitrary OpenGL content. This is useful to display and manipulate 3D objects as part of an interactive application. The implementation uses scissoring to ensure that rendered objects don't spill into neighboring widgets.

Usage: override `drawGL` in subclasses to provide custom drawing code.

Inherits from *nanogui::Widget*

Public Functions

GLCanvas (*Widget* *parent)

const Color &backgroundcolor () const

Return the background color.

void **setBackgroundColor** (const *Color* &backgroundcolor)

Set the background color.

void **setDrawBorder** (const bool bDrawBorder)

Set whether to draw the widget border or not.

const bool &drawBorder () const

Return whether the widget border gets drawn or not.

virtual void draw (NVGcontext *ctx)

Draw the canvas.

virtual void **drawGL** ()

Draw the GL scene. Override this method to draw the actual GL content.

virtual void **save** (*Serializer &s*) **const**

Save and load widget properties.

virtual bool **load** (*Serializer &s*)

Restore the state of the widget from the given *Serializer* instance.

Protected Functions

void **drawWidgetBorder** (NVGcontext *ctx) **const**

Internal helper function for drawing the widget border.

Protected Attributes

Color **mBackgroundColor**

bool **mDrawBorder**

Class GLFramebuffer

- Defined in *File glutil.h*

class nanogui : **GLFramebuffer**

Helper class for creating framebuffer objects.

Public Functions

GLFramebuffer ()

Default constructor: unusable until you call the *init* () method.

void **init** (**const** Vector2i &size, int nSamples)

Create a new framebuffer with the specified size and number of MSAA samples.

void **free** ()

Release all associated resources.

void **bind** ()

Bind the framebuffer object.

void **release** ()

Release/unbind the framebuffer object.

void **blit** ()

Blit the framebuffer object onto the screen.

bool **ready** ()

Return whether or not the framebuffer object has been initialized.

int **samples** () **const**

Return the number of MSAA samples.

void **downloadTGA** (const std::string &filename)

Quick and dirty method to write a TGA (32bpp RGBA) file of the framebuffer contents for debugging.

Protected Attributes

GLuint **mFramebuffer**

GLuint **mDepth**

GLuint **mColor**

Vector2i **mSize**

int **mSamples**

Class GLShader

- Defined in *File glutil.h*

Nested Types:

- *Struct GLShader::Buffer*

class nanogui : **GLShader**

Helper class for compiling and linking OpenGL shaders and uploading associated vertex and index buffers from Eigen matrices.

Public Functions

GLShader ()

Create an uninitialized OpenGL shader.

bool **init** (const std::string &name, const std::string &vertex_str, const std::string &fragment_str, const std::string &geometry_str = "")

Initialize the shader using the specified source strings.

Parameters

- name: The name this shader will be registered as.
- vertex_str: The source of the vertex shader as a string.
- fragment_str: The source of the fragment shader as a string.
- geometry_str: The source of the geometry shader as a string. The default value is the empty string, which indicates no geometry shader will be used.

bool **initFromFiles** (const std::string &name, const std::string &vertex_fname, const std::string &fragment_fname, const std::string &geometry_fname = "")

Initialize the shader using the specified files on disk.

Parameters

- name: The name this shader will be registered as.
- vertex_fname: The path to the file containing the source of the fragment shader.
- fragment_fname: The path to the file containing the source of the vertex shader.

- `geometry_fname`: The path to the file containing the source of the geometry shader. The default value is the empty string, which indicates no geometry shader will be used.

const std::string &**name** () **const**

Return the name of the shader.

void **define** (**const** std::string &*key*, **const** std::string &*value*)

Set a preprocessor definition.

void **bind** ()

Select this shader for subsequent draw calls.

void **free** ()

Release underlying OpenGL objects.

GLint **attrib** (**const** std::string &*name*, bool *warn* = true) **const**

Return the handle of a named shader attribute (-1 if it does not exist)

GLint **uniform** (**const** std::string &*name*, bool *warn* = true) **const**

Return the handle of a uniform attribute (-1 if it does not exist)

template <typename Matrix>

void **uploadAttrib** (**const** std::string &*name*, **const** Matrix &*M*, int *version* = -1)

Upload an Eigen matrix as a vertex buffer object (refreshing it as needed)

template <typename Matrix>

void **downloadAttrib** (**const** std::string &*name*, Matrix &*M*)

Download a vertex buffer object into an Eigen matrix.

template <typename Matrix>

void **uploadIndices** (**const** Matrix &*M*, int *version* = -1)

Upload an index buffer.

void **invalidateAttribs** ()

Invalidate the version numbers associated with attribute data.

void **freeAttrib** (**const** std::string &*name*)

Completely free an existing attribute buffer.

bool **hasAttrib** (**const** std::string &*name*) **const**

Check if an attribute was registered a given name.

void **shareAttrib** (**const** *GLShader* &*otherShader*, **const** std::string &*name*, **const** std::string &*as* =
"")

Create a symbolic link to an attribute of another *GLShader*. This avoids duplicating unnecessary data.

int **attribVersion** (**const** std::string &*name*) **const**

Return the version number of a given attribute.

void **resetAttribVersion** (**const** std::string &*name*)

Reset the version number of a given attribute.

void **drawArray** (int *type*, uint32_t *offset*, uint32_t *count*)

Draw a sequence of primitives.

void **drawIndexed** (int *type*, uint32_t *offset*, uint32_t *count*)

Draw a sequence of primitives using a previously uploaded index buffer.

template <typename T>

void **setUniform** (**const** std::string &*name*, **const** Eigen::Matrix<T, 4, 4> &*mat*, bool *warn* = true)

Initialize a uniform parameter with a 4x4 matrix (float)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==1, int >::type = 0>
 void **setUniform** (const std::string &name, T value, bool warn = true)

Initialize a uniform parameter with an integer value.

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==0, int >::type = 0>
 void **setUniform** (const std::string &name, T value, bool warn = true)

Initialize a uniform parameter with a floating point value.

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==1, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 2, 1> &v, bool warn = true)

Initialize a uniform parameter with a 2D vector (int)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==0, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 2, 1> &v, bool warn = true)

Initialize a uniform parameter with a 2D vector (float)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==1, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 3, 1> &v, bool warn = true)

Initialize a uniform parameter with a 3D vector (int)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==0, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 3, 1> &v, bool warn = true)

Initialize a uniform parameter with a 3D vector (float)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==1, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 4, 1> &v, bool warn = true)

Initialize a uniform parameter with a 4D vector (int)

template <typename T, typename std::enable_if< detail::type_traits< T >::integral==0, int >::type = 0>
 void **setUniform** (const std::string &name, const Eigen::Matrix<T, 4, 1> &v, bool warn = true)

Initialize a uniform parameter with a 4D vector (float)

void **setUniform** (const std::string &name, const *GLUniformBuffer* &buf, bool warn = true)

Initialize a uniform buffer with a uniform buffer object.

size_t **bufferSize** () const

Return the size of all registered buffers in bytes.

void **uploadAttrib** (const std::string &name, size_t size, int dim, uint32_t compSize, GLuint glType,
 bool integral, const void *data, int version = -1)

void **downloadAttrib** (const std::string &name, size_t size, int dim, uint32_t compSize, GLuint
 glType, void *data)

Protected Attributes

std::string **mName**

GLuint **mVertexShader**

GLuint **mFragmentShader**

GLuint **mGeometryShader**

GLuint **mProgramShader**

GLuint **mVertexArrayObject**

std::map<std::string, *Buffer*> **mBufferObjects**

std::map<std::string, std::string> **mDefinitions**

struct Buffer

A wrapper struct for maintaining various aspects of items being managed by OpenGL.

Public Members

GLuint **id**
GLuint **glType**
GLuint **dim**
GLuint **compSize**
GLuint **size**
int **version**

Class GLUniformBuffer

- Defined in *File glutil.h*

class nanogui::**GLUniformBuffer**
Helper class for creating OpenGL Uniform Buffer objects.

Public Functions

GLUniformBuffer ()
Default constructor: unusable until you call the *init()* method.

void **init** ()
Create a new uniform buffer.

void **free** ()
Release underlying OpenGL object.

void **bind** (int *index*)
Bind the uniform buffer to a specific binding point.

void **release** ()
Release/unbind the uniform buffer.

void **update** (const std::vector<uint8_t> &*data*)
Update content on the GPU using data.

int **getBindingPoint** () const
Return the binding point of this uniform buffer.

Class Graph

- Defined in *File graph.h*

class nanogui::**Graph**
Simple graph widget for showing a function plot.
Inherits from *nanogui::Widget*

Public Functions

```

Graph (Widget *parent, const std::string &caption = "Untitled")
const std::string &caption () const
void setCaption (const std::string &caption)
const std::string &header () const
void setHeader (const std::string &header)
const std::string &footer () const
void setFooter (const std::string &footer)
const Color &backgroundColor () const
void setBackgroundColor (const Color &backgroundColor)
const Color &foregroundColor () const
void setForegroundColor (const Color &foregroundColor)
const Color &textColor () const
void setTextColor (const Color &textColor)
const VectorXf &values () const
VectorXf &values ()
void setValues (const VectorXf &values)
virtual Vector2i preferredSize (NVGcontext *ctx) const
    Compute the preferred size of the widget.
virtual void draw (NVGcontext *ctx)
    Draw the widget (and all child widgets)
virtual void save (Serializer &s) const
    Save the state of the widget into the given Serializer instance.
virtual bool load (Serializer &s)
    Restore the state of the widget from the given Serializer instance.

```

Protected Attributes

```

std::string mCaption
std::string mHeader
std::string mFooter
Color mBackgroundColor
Color mForegroundColor
Color mTextColor
VectorXf mValues

```

Class GridLayout

- Defined in *File layout.h*

class nanogui::GridLayout

Grid layout.

Widgets are arranged in a grid that has a fixed grid resolution *resolution* along one of the axes. The layout orientation indicates the fixed dimension; widgets are also appended on this axis. The spacing between items can be specified per axis. The horizontal/vertical alignment can be specified per row and column.

Inherits from *nanogui::Layout*

Public Functions

GridLayout (*Orientation* orientation = Orientation::Horizontal, int *resolution* = 2, *Alignment* alignment = Alignment::Middle, int *margin* = 0, int *spacing* = 0)
Create a 2-column grid layout by default.

Orientation orientation () const

void **setOrientation** (*Orientation* orientation)

int **resolution** () const

void **setResolution** (int *resolution*)

int **spacing** (int *axis*) const

void **setSpacing** (int *axis*, int *spacing*)

void **setSpacing** (int *spacing*)

int **margin** () const

void **setMargin** (int *margin*)

Alignment alignment (int *axis*, int *item*) const

void **setColAlignment** (*Alignment* value)

void **setRowAlignment** (*Alignment* value)

void **setColAlignment** (const std::vector<*Alignment*> &value)

void **setRowAlignment** (const std::vector<*Alignment*> &value)

virtual Vector2i **preferredSize** (NVGcontext *ctx, const *Widget* *widget) const

virtual void **performLayout** (NVGcontext *ctx, *Widget* *widget) const

Protected Functions

void **computeLayout** (NVGcontext *ctx, const *Widget* *widget, std::vector<int> *grid) const

Protected Attributes

Orientation **mOrientation**
Alignment **mDefaultAlignment**[2]
 std::vector<*Alignment*> **mAlignment**[2]
 int **mResolution**
 Vector2i **mSpacing**
 int **mMargin**

Class GroupLayout

- Defined in *File layout.h*

class nanogui::**GroupLayout**

Special layout for widgets grouped by labels.

This widget resembles a box layout in that it arranges a set of widgets vertically. All widgets are indented on the horizontal axis except for *Label* widgets, which are not indented.

This creates a pleasing layout where a number of widgets are grouped under some high-level heading.

Inherits from *nanogui::Layout*

Public Functions

GroupLayout (int *margin* = 15, int *spacing* = 6, int *groupSpacing* = 14, int *groupIndent* = 20)

int **margin** () **const**

void **setMargin** (int *margin*)

int **spacing** () **const**

void **setSpacing** (int *spacing*)

int **groupIndent** () **const**

void **setGroupIndent** (int *groupIndent*)

int **groupSpacing** () **const**

void **setGroupSpacing** (int *groupSpacing*)

virtual Vector2i **preferredSize** (NVGcontext **ctx*, **const** *Widget* **widget*) **const**

virtual void **performLayout** (NVGcontext **ctx*, *Widget* **widget*) **const**

Protected Attributes

```
int mMargin  
int mSpacing  
int mGroupSpacing  
int mGroupIndent
```

Class ImagePanel

- Defined in *File imagepanel.h*

```
class nanogui : : ImagePanel
```

Image panel widget which shows a number of square-shaped icons.

Inherits from *nanogui::Widget*

Public Types

```
typedef std::vector<std::pair<int, std::string>> Images
```

Public Functions

```
ImagePanel (Widget *parent)
```

```
void setImages (const Images &data)
```

```
const Images &images () const
```

```
std::function<void (int) > callback  
const
```

```
void setCallback (const std::function<void>() int  
    > &callback
```

```
virtual bool mouseMotionEvent (const Vector2i &p, const Vector2i &rel, int button, int modifiers)  
    Handle a mouse motion event (default implementation: propagate to children)
```

```
virtual bool mouseButtonEvent (const Vector2i &p, int button, bool down, int modifiers)  
    Handle a mouse button event (default implementation: propagate to children)
```

```
virtual Vector2i preferredSize (NVGcontext *ctx) const  
    Compute the preferred size of the widget.
```

```
virtual void draw (NVGcontext *ctx)  
    Draw the widget (and all child widgets)
```

Protected Functions

```
Vector2i gridSize () const
```

```
int indexForPosition (const Vector2i &p) const
```

Protected Attributes

Images **mImages**
 std::function<void (int) > **mCallback**
 int **mThumbSize**
 int **mSpacing**
 int **mMargin**
 int **mMouseIndex**

Class ImageView

- Defined in *File imageview.h*

class nanogui::ImageView
Widget used to display images.
 Inherits from *nanogui::Widget*

Public Functions

ImageView (*Widget* *parent, GLuint imageID)
 ~ImageView ()
 void **bindImage** (GLuint imageId)
GLShader &imageShader ()
 Vector2f **positionF** () const
 Vector2f **sizeF** () const
 const Vector2i &imageSize () const
 Vector2i **scaledImageSize** () const
 Vector2f **imageSizeF** () const
 Vector2f **scaledImageSizeF** () const
 const Vector2f &offset () const
 void **setOffset** (const Vector2f &offset)
 float **scale** () const
 void **setScale** (float scale)
 bool **fixedOffset** () const
 void **setFixedOffset** (bool fixedOffset)
 bool **fixedScale** () const

void **setFixedScale** (bool *fixedScale*)

float **zoomSensitivity** () **const**

void **setZoomSensitivity** (float *zoomSensitivity*)

float **gridThreshold** () **const**

void **setGridThreshold** (float *gridThreshold*)

float **pixelInfoThreshold** () **const**

void **setPixelInfoThreshold** (float *pixelInfoThreshold*)

void **setFontScaleFactor** (float *fontScaleFactor*)

float **fontScaleFactor** () **const**

Vector2f **imageCoordinateAt** (**const** Vector2f &*position*) **const**

Calculates the image coordinates of the given pixel position on the widget.

Vector2f **clampedImageCoordinateAt** (**const** Vector2f &*position*) **const**

Calculates the image coordinates of the given pixel position on the widget. If the position provided corresponds to a coordinate outside the range of the image, the coordinates are clamped to edges of the image.

Vector2f **positionForCoordinate** (**const** Vector2f &*imageCoordinate*) **const**

Calculates the position inside the widget for the given image coordinate.

void **setImageCoordinateAt** (**const** Vector2f &*position*, **const** Vector2f &*imageCoordinate*)

Modifies the internal state of the image viewer widget so that the pixel at the provided position on the widget has the specified image coordinate. Also clamps the values of offset to the sides of the widget.

void **center** ()

Centers the image without affecting the scaling factor.

void **fit** ()

Centers and scales the image so that it fits inside the widgets.

void **setScaleCentered** (float *scale*)

Set the scale while keeping the image centered.

void **moveOffset** (**const** Vector2f &*delta*)

Moves the offset by the specified amount. Does bound checking.

void **zoom** (int *amount*, **const** Vector2f &*focusPosition*)

Changes the scale factor by the provided amount modified by the zoom sensitivity member variable. The scaling occurs such that the image coordinate under the focused position remains in the same position before and after the scaling.

bool **keyboardEvent** (int *key*, int *scancode*, int *action*, int *modifiers*)

Handle a keyboard event (default implementation: do nothing)

bool **keyboardCharacterEvent** (unsigned int *codepoint*)

Handle text input (UTF-32 format) (default implementation: do nothing)

bool **mouseDragEvent** (**const** Vector2i &*p*, **const** Vector2i &*rel*, int *button*, int *modifiers*)

Handle a mouse drag event (default implementation: do nothing)

bool **scrollEvent** (**const** Vector2i &*p*, **const** Vector2f &*rel*)

Handle a mouse scroll event (default implementation: propagate to children)

bool **gridVisible** () const
Function indicating whether the grid is currently visible.

bool **pixelInfoVisible** () const
Function indicating whether the pixel information is currently visible.

bool **helpersVisible** () const
Function indicating whether any of the overlays are visible.

Vector2i **preferredSize** (NVGcontext *ctx) const
Compute the preferred size of the widget.

void **performLayout** (NVGcontext *ctx)
Invoke the associated layout generator to properly place child widgets, if any.

void **draw** (NVGcontext *ctx)
Draw the widget (and all child widgets)

Class IntBox

- Defined in *File textbox.h*

template <typename Scalar>

class nanogui::IntBox

A specialization of *TextBox* for representing integral values.

Template parameters should be integral types, e.g. int, long, uint32_t, etc.

Inherits from *nanogui::TextBox*

Public Functions

IntBox (*Widget* *parent, Scalar value = (Scalar) 0)

Scalar **value** () const

void **setValue** (Scalar value)

void **setCallback** (const std::function<void> Scalar
> &cb)

void **setValueIncrement** (Scalar incr)

void **setMinValue** (Scalar minValue)

void **setMaxValue** (Scalar maxValue)

void **setMinMaxValues** (Scalar minValue, Scalar maxValue)

virtual bool **mouseButtonEvent** (const Vector2i &p, int button, bool down, int modifiers)
Handle a mouse button event (default implementation: propagate to children)

virtual bool **mouseDragEvent** (const Vector2i &p, const Vector2i &rel, int button, int modifiers)
Handle a mouse drag event (default implementation: do nothing)

virtual bool **scrollEvent** (const Vector2i &p, const Vector2f &rel)
Handle a mouse scroll event (default implementation: propagate to children)

Class Label

- Defined in *File label.h*

class nanogui::Label

Text label widget.

The font and color can be customized. When *Widget::setFixedWidth()* is used, the text is wrapped when it surpasses the specified width.

Inherits from *nanogui::Widget*

Public Functions

Label (*Widget* *parent, const std::string &caption, const std::string &font = “sans”, int fontSize = -1)

const std::string &caption () const

Get the label’s text caption.

void setCaption (const std::string &caption)

Set the label’s text caption.

void setFont (const std::string &font)

Set the currently active font (2 are available by default: ‘sans’ and ‘sans-bold’)

const std::string &font () const

Get the currently active font.

Color color () const

Get the label color.

void setColor (const *Color* &color)

Set the label color.

virtual void setTheme (*Theme* *theme)

Set the *Theme* used to draw this widget.

virtual Vector2i preferredSize (NVGcontext *ctx) const

Compute the size needed to fully display the label.

virtual void draw (NVGcontext *ctx)

Draw the label.

virtual void save (*Serializer* &s) const

Save the state of the widget into the given *Serializer* instance.

virtual bool load (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

std::string mCaption

std::string mFont

Color mColor

Class Layout

- Defined in *File layout.h*

class `nanogui::Layout`

Basic interface of a layout engine.

Inherits from *nanogui::Object*

Subclassed by *nanogui::AdvancedGridLayout*, *nanogui::BoxLayout*, *nanogui::GridLayout*, *nanogui::GroupLayout*

Public Functions

virtual void **performLayout** (NVGcontext *ctx, *Widget* *widget) **const** = 0

virtual Vector2i **preferredSize** (NVGcontext *ctx, **const** *Widget* *widget) **const** = 0

Protected Functions

virtual ~**Layout** ()

Class MessageDialog

- Defined in *File messagedialog.h*

Nested Types:

- *Enum MessageDialog::Type*

class `nanogui::MessageDialog`

Simple “OK” or “Yes/No”-style modal dialogs.

Inherits from *nanogui::Window*

Public Types

enum **Type**

Classification of the type of message this *MessageDialog* represents.

Values:

Information

Question

Warning

Public Functions

MessageDialog (*Widget* *parent, *Type* type, **const** std::string &title = “Untitled”, **const** std::string &message = “Message”, **const** std::string &buttonText = “OK”, **const** std::string &altButtonText = “Cancel”, bool altButton = false)

```
Label *messageLabel ()

const Label *messageLabel () const

std::function<void (int) > callback
    const

void setCallback (const std::function<void> int
    > &callback
```

Protected Attributes

```
std::function<void (int) > mCallback
Label *mMessageLabel
```

Class Object

- Defined in *File object.h*

class nanogui::Object

Reference counted object base class.

Subclassed by *nanogui::Layout*, *nanogui::Theme*, *nanogui::Widget*

Public Functions

Object ()

Default constructor.

Object (const *Object*&)

Copy constructor.

int **getRefCount** () const

Return the current reference count.

void **incRef** () const

Increase the object's reference count by one.

void **decRef** (bool *dealloc* = true) const

Decrease the reference count of the object and possibly deallocate it.

The object will automatically be deallocated once the reference count reaches zero.

Protected Functions

virtual ~Object ()

Virtual protected destructor. (Will only be called by *ref*)

Class Popup

- Defined in *File popup.h*

Nested Types:

- *Enum Popup::Side*

class nanogui : **Popup**

Popup window for combo boxes, popup buttons, nested dialogs etc.

Usually the *Popup* instance is constructed by another widget (e.g. *PopupButton*) and does not need to be created by hand.

Inherits from *nanogui::Window*

Public Types

enum **Side**

Values:

Left = 0

Right

Public Functions

Popup (*Widget* *parent, *Window* *parentWindow)

Create a new popup parented to a screen (first argument) and a parent window.

void **setAnchorPos** (const Vector2i &anchorPos)

Return the anchor position in the parent window; the placement of the popup is relative to it.

const Vector2i &**anchorPos** () const

Set the anchor position in the parent window; the placement of the popup is relative to it.

void **setAnchorHeight** (int anchorHeight)

Set the anchor height; this determines the vertical shift relative to the anchor position.

int **anchorHeight** () const

Return the anchor height; this determines the vertical shift relative to the anchor position.

void **setSide** (*Side* popupSide)

Set the side of the parent window at which popup will appear.

Side **side** () const

Return the side of the parent window at which popup will appear.

Window ***parentWindow** ()

Return the parent window of the popup.

const *Window* ***parentWindow** () const

Return the parent window of the popup.

virtual void **performLayout** (NVGcontext *ctx)

Invoke the associated layout generator to properly place child widgets, if any.

virtual void **draw** (NVGcontext *ctx)

Draw the popup window.

virtual void **save** (*Serializer* &s) **const**

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Functions

virtual void **refreshRelativePlacement** ()

Internal helper function to maintain nested window position values.

Protected Attributes

Window *mParentWindow

Vector2i mAnchorPos

int mAnchorHeight

Side mSide

Class PopupButton

- Defined in *File popupbutton.h*

class nanogui::**PopupButton**

Button which launches a popup widget.

Inherits from *nanogui::Button*

Subclassed by *nanogui::ColorPicker*, *nanogui::ComboBox*

Public Functions

PopupButton (*Widget* *parent, **const** std::string &caption = “Untitled”, int buttonIcon = 0)

void **setChevronIcon** (int icon)

int **chevronIcon** () **const**

void **setSide** (*Popup::Side* popupSide)

Popup::Side **side** () **const**

Popup ***popup** ()

const *Popup* ***popup** () **const**

virtual void **draw** (NVGcontext *ctx)

Draw the widget (and all child widgets)

virtual Vector2i **preferredSize** (NVGcontext *ctx) **const**

Compute the preferred size of the widget.

virtual void **performLayout** (NVGcontext *ctx)

Invoke the associated layout generator to properly place child widgets, if any.

virtual void **save** (*Serializer* &s) **const**

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

Popup *mPopup

int mChevronIcon

Class ProgressBar

- Defined in *File progressbar.h*

class nanogui : : **ProgressBar**

Standard widget for visualizing progress.

Inherits from *nanogui::Widget*

Public Functions

ProgressBar (*Widget* *parent)

float **value** ()

void **setValue** (float value)

virtual Vector2i **preferredSize** (NVGcontext *ctx) **const**

Compute the preferred size of the widget.

virtual void **draw** (NVGcontext *ctx)

Draw the widget (and all child widgets)

virtual void **save** (*Serializer* &s) **const**

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

float mValue

Class `ref`

- Defined in *File `object.h`*

template <typename *T*>

class `nanogui::ref`

Reference counting helper.

The *ref* template is a simple wrapper to store a pointer to an object. It takes care of increasing and decreasing the object's reference count as needed. When the last reference goes out of scope, the associated object will be deallocated.

The advantage over C++ solutions such as `std::shared_ptr` is that the reference count is very compactly integrated into the base object itself.

Public Functions

ref ()

Create a `nullptr`-valued reference.

ref (*T *ptr*)

Construct a reference from a pointer.

ref (**const** *ref* &*r*)

Copy constructor.

ref (*ref* &&*r*)

Move constructor.

~ref ()

Destroy this reference.

ref &**operator=** (*ref* &&*r*)

Move another reference into the current one.

ref &**operator=** (**const** *ref* &*r*)

Overwrite this reference with another reference.

ref &**operator=** (*T *ptr*)

Overwrite this reference with a pointer to another object.

bool operator== (**const** *ref* &*r*) **const**

Compare this reference with another reference.

bool operator!= (**const** *ref* &*r*) **const**

Compare this reference with another reference.

bool operator== (**const** *T *ptr*) **const**

Compare this reference with a pointer.

bool operator!= (**const** *T *ptr*) **const**

Compare this reference with a pointer.

*T ****operator->** ()

Access the object referenced by this reference.

const *T ****operator->** () **const**

Access the object referenced by this reference.

T &operator* ()
Return a C++ reference to the referenced object.

const T &operator* () const
Return a const C++ reference to the referenced object.

operator T * ()
Return a pointer to the referenced object.

T *get ()
Return a const pointer to the referenced object.

const T *get () const
Return a pointer to the referenced object.

operator bool () const
Check if the object is defined.

Class Screen

- Defined in *File screen.h*

class nanogui::Screen

Represents a display surface (i.e. a full-screen or windowed GLFW window) and forms the root element of a hierarchy of nanogui widgets.

Inherits from *nanogui::Widget*

Public Functions

Screen (const Vector2i &size, const std::string &caption, bool resizable = true, bool fullscreen = false, int colorBits = 8, int alphaBits = 8, int depthBits = 24, int stencilBits = 8, int nSamples = 0, unsigned int glMajor = 3, unsigned int glMinor = 3)
Create a new *Screen* instance

Parameters

- **size**: Size in pixels at 96 dpi (on high-DPI screens, the actual resolution in terms of hardware pixels may be larger by an integer factor)
- **caption**: *Window* title (in UTF-8 encoding)
- **resizable**: If creating a window, should it be resizable?
- **fullscreen**: Specifies whether to create a windowed or full-screen view
- **colorBits**: Number of bits per pixel dedicated to the R/G/B color components
- **alphaBits**: Number of bits per pixel dedicated to the alpha channel
- **depthBits**: Number of bits per pixel dedicated to the Z-buffer
- **stencilBits**: Number of bits per pixel dedicated to the stencil buffer (recommended to set this to 8. NanoVG can draw higher-quality strokes using a stencil buffer)
- **nSamples**: Number of MSAA samples (set to 0 to disable)

- `glMajor`: The requested OpenGL Major version number. Default is 3, if changed the value must correspond to a forward compatible core profile (for portability reasons). For example, set this to 4 and `glMinor` to 1 for a forward compatible core OpenGL 4.1 profile. Requesting an invalid profile will result in no context (and therefore no GUI) being created.
- `glMinor`: The requested OpenGL Minor version number. Default is 3, if changed the value must correspond to a forward compatible core profile (for portability reasons). For example, set this to 1 and `glMajor` to 4 for a forward compatible core OpenGL 4.1 profile. Requesting an invalid profile will result in no context (and therefore no GUI) being created.

virtual ~Screen ()

Release all resources.

const std::string &caption () const

Get the window title bar caption.

void setCaption (const std::string &caption)

Set the window title bar caption.

const Color &background () const

Return the screen's background color.

void setBackground (const Color &background)

Set the screen's background color.

void setVisible (bool visible)

Set the top-level window visibility (no effect on full-screen windows)

void setSize (const Vector2i &size)

Set window size.

virtual void drawAll ()

Draw the *Screen* contents.

virtual void drawContents ()

Draw the window contents put your OpenGL draw calls here.

float pixelRatio () const

Return the ratio between pixel and device coordinates (e.g. ≥ 2 on Mac Retina displays)

virtual bool dropEvent (const std::vector<std::string>&)

Handle a file drop event.

virtual bool keyboardEvent (int key, int scancode, int action, int modifiers)

Default keyboard event handler.

virtual bool keyboardCharacterEvent (unsigned int codepoint)

Text input event handler: codepoint is native endian UTF-32 format.

virtual bool resizeEvent (const Vector2i &size)

Window resize event handler.

std::function<void (Vector2i)> &resizeCallback

const Set the resize callback.

**void setResizeCallback (const std::function<void (Vector2i
> &callback**

Vector2i **mousePos** () **const**

Return the last observed mouse position value.

GLFWwindow ***glfwWindow** ()

Return a pointer to the underlying GLFW window data structure.

NVGcontext ***nvgContext** ()

Return a pointer to the underlying nanoVG draw context.

void **setShutdownGLFWOnDestruct** (bool v)

bool **shutdownGLFWOnDestruct** ()

void **performLayout** ()

Compute the layout of all widgets.

Screen ()

Default constructor.

Performs no initialization at all. Use this if the application is responsible for setting up GLFW, OpenGL, etc.

In this case, override *Screen* and call initialize() with a pointer to an existing GLFWwindow instance

You will also be responsible in this case to deliver GLFW callbacks to the appropriate callback event handlers below

void **initialize** (GLFWwindow *window, bool shutdownGLFWOnDestruct)

Initialize the *Screen*.

bool **cursorPosCallbackEvent** (double x, double y)

bool **mouseButtonCallbackEvent** (int button, int action, int modifiers)

bool **keyCallbackEvent** (int key, int scancode, int action, int mods)

bool **charCallbackEvent** (unsigned int codepoint)

bool **dropCallbackEvent** (int count, const char **filenames)

bool **scrollCallbackEvent** (double x, double y)

bool **resizeCallbackEvent** (int width, int height)

void **updateFocus** (*Widget* *widget)

void **disposeWindow** (*Window* *window)

void **centerWindow** (*Window* *window)

void **moveWindowToFront** (*Window* *window)

void **drawWidgets** ()

Protected Attributes

GLFWwindow ***mGLFWWindow**

NVGcontext ***mNVGContext**

GLFWcursor ***mCursors**[(int) Cursor::CursorCount]

```
Cursor mCursor
std::vector<Widget *> mFocusPath
Vector2i mFBSize
float mPixelRatio
int mMouseState
int mModifiers
Vector2i mMousePos
bool mDragActive
Widget *mDragWidget = nullptr
double mLastInteraction
bool mProcessEvent
Color mBackground
std::string mCaption
bool mShutdownGLFWOnDestruct
bool mFullscreen
std::function<void (Vector2i) > mResizeCallback
```

Class Serializer

- Defined in *File core.h*

class nanogui::**Serializer**

Serialization helper class.

This class can be used to store and retrieve a great variety of C++ data types using a compact binary file format. The intended purpose is to quickly save and restore the complete state of an application, e.g. to facilitate debugging sessions. This class supports all core C++ types, NanoGUI widgets, sparse and dense Eigen matrices, as well as OpenGL shaders and buffer objects.

Note that this header file just provides the basics; the files `nanogui/serializer/opengl.h`, and `nanogui/serializer/sparse.h` must be included to serialize the respective data types.

Public Functions

Serializer (**const** std::string &*filename*, bool *write*)

Create a new serialized file for reading or writing.

~Serializer ()

Release all resources.

size_t **size** ()

Return the current size of the output file.

void **push** (**const** std::string &*name*)

Push a name prefix onto the stack (use this to isolate identically-named data fields)


```
void pop ()
    Pop a name prefix from the stack.

std::vector<std::string> keys () const
    Return all field names under the current name prefix.

void setCompatibility (bool compatibility)
    Enable/disable compatibility mode.

    When enabled, missing attributes cause a warning to be printed, but get() does not throw an exception.

bool compatibility ()
    Return whether compatibility mode is enabled.

template <typename T>
void set (const std::string &name, const T &value)
    Store a field in the serialized file (when opened with write=true)

template <typename T>
bool get (const std::string &name, T &value)
    Retrieve a field from the serialized file (when opened with write=false)
```

Public Static Functions

```
static bool isSerializedFile (const std::string &filename)
    Check whether a file contains serialized data.
```

Protected Functions

```
void set_base (const std::string &name, const std::string &type_id)

bool get_base (const std::string &name, const std::string &type_id)

void writeTOC ()

void readTOC ()

void read (void *p, size_t size)

void write (const void *p, size_t size)

void seek (size_t pos)
```

Class Slider

- Defined in *File slider.h*

```
class nanogui::Slider
    Fractional slider widget with mouse control.

    Inherits from nanogui::Widget
```

Public Functions

Slider (*Widget *parent*)

float **value** () **const**

void **setValue** (float *value*)

const Color &highlightColor () **const**

void **setHighlightColor** (**const Color &highlightColor**)

std::pair<float, float> **range** () **const**

void **setRange** (std::pair<float, float> *range*)

std::pair<float, float> **highlightedRange** () **const**

void **setHighlightedRange** (std::pair<float, float> *highlightedRange*)

std::function<void (float)> **callback**
const

void **setCallback** (**const** std::function<void> float
 > *&callback*)

std::function<void (float)> **finalCallback**
const

void **setFinalCallback** (**const** std::function<void> float
 > *&callback*)

virtual Vector2i **preferredSize** (NVGcontext **ctx*) **const**
 Compute the preferred size of the widget.

virtual bool **mouseDragEvent** (**const** Vector2i &*p*, **const** Vector2i &*rel*, int *button*, int *modifiers*)
 Handle a mouse drag event (default implementation: do nothing)

virtual bool **mouseButtonEvent** (**const** Vector2i &*p*, int *button*, bool *down*, int *modifiers*)
 Handle a mouse button event (default implementation: propagate to children)

virtual void **draw** (NVGcontext **ctx*)
 Draw the widget (and all child widgets)

virtual void **save** (*Serializer &s*) **const**
 Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer &s*)
 Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

float **mValue**

std::function<void (float)> **mCallback**

std::function<void (float)> **mFinalCallback**

std::pair<float, float> **mRange**

```
std::pair<float, float> mHighlightedRange
```

```
Color mHighlightColor
```

Class StackedWidget

- Defined in *File stackedwidget.h*

```
class nanogui::StackedWidget
```

A stack widget.

Inherits from *nanogui::Widget*

Public Functions

```
StackedWidget (Widget *parent)
```

```
void setSelectedIndex (int index)
```

```
int selectedIndex () const
```

```
virtual void performLayout (NVGcontext *ctx)
```

Invoke the associated layout generator to properly place child widgets, if any.

```
virtual Vector2i preferredSize (NVGcontext *ctx) const
```

Compute the preferred size of the widget.

```
virtual void addChild (int index, Widget *widget)
```

Add a child widget to the current widget at the specified index.

This function almost never needs to be called by hand, since the constructor of *Widget* automatically adds the current widget to its parent

Class TabHeader

- Defined in *File tabheader.h*

Nested Types:

- *Enum TabHeader::ClickLocation*
- *Class TabHeader::TabButton*
- *Struct TabButton::StringView*

```
class nanogui::TabHeader
```

A Tab navigable widget.

Inherits from *nanogui::Widget*

Public Functions

```
TabHeader (Widget *parent, const std::string &font = "sans-bold")
```

```
void setFont (const std::string &font)
```

const std::string &**font** () **const**

bool **overflowing** () **const**

void **setCallback** (**const** std::function<void> int
> &*callback*) Sets the callable objects which is invoked when a tab button is pressed. The argument provided to the callback is the index of the tab.

const std::function<void (int)> &**callback**
const

void **setActiveTab** (int *tabIndex*)

int **activeTab** () **const**

bool **isTabVisible** (int *index*) **const**

int **tabCount** () **const**

void **addTab** (**const** std::string &*label*)
Inserts a tab at the end of the tabs collection.

void **addTab** (int *index*, **const** std::string &*label*)
Inserts a tab into the tabs collection at the specified index.

int **removeTab** (**const** std::string &*label*)
Removes the tab with the specified label and returns the index of the label. Returns -1 if there was no such tab

void **removeTab** (int *index*)
Removes the tab with the specified index.

const std::string &**tabLabelAt** (int *index*) **const**
Retrieves the label of the tab at a specific index.

int **tabIndex** (**const** std::string &*label*)
Retrieves the index of a specific tab label. Returns the number of tabs (tabCount) if there is no such tab.

void **ensureTabVisible** (int *index*)
Recalculate the visible range of tabs so that the tab with the specified index is visible. The tab with the specified index will either be the first or last visible one depending on the position relative to the old visible range.

std::pair<Vector2i, Vector2i> **visibleButtonArea** () **const**
Returns a pair of Vectors describing the top left (pair.first) and the bottom right (pair.second) positions of the rectangle containing the visible tab buttons.

std::pair<Vector2i, Vector2i> **activeButtonArea** () **const**
Returns a pair of Vectors describing the top left (pair.first) and the bottom right (pair.second) positions of the rectangle containing the active tab button. Returns two zero vectors if the active button is not visible.

virtual void **performLayout** (NVGcontext **ctx*)
Invoke the associated layout generator to properly place child widgets, if any.

virtual Vector2i **preferredSize** (NVGcontext **ctx*) **const**
Compute the preferred size of the widget.

virtual bool **mouseButtonEvent** (**const** Vector2i &*p*, int *button*, bool *down*, int *modifiers*)
Handle a mouse button event (default implementation: propagate to children)

virtual void draw (NVGcontext *ctx)
 Draw the widget (and all child widgets)

Class TabHeader::TabButton

- Defined in *File tabheader.h*
- Nested type of *Class TabHeader*

Nested Types:

- *Struct TabButton::StringView*

class nanogui::TabHeader::TabButton
 Implementation class of the actual tab buttons.

Public Functions

TabButton (*TabHeader* &header, **const** std::string &label)

void **setLabel** (**const** std::string &label)

const std::string &label () **const**

void **setSize** (**const** Vector2i &size)

const Vector2i &size () **const**

Vector2i **preferredSize** (NVGcontext *ctx) **const**

void **calculateVisibleString** (NVGcontext *ctx)

void **drawAtPosition** (NVGcontext *ctx, **const** Vector2i &position, bool active)

void **drawActiveBorderAt** (NVGcontext *ctx, **const** Vector2i &position, float offset, **const** Color &color)

void **drawInactiveBorderAt** (NVGcontext *ctx, **const** Vector2i &position, float offset, **const** Color &color)

Public Static Attributes

constexpr const char *dots = "..."

Class TabWidget

- Defined in *File tabwidget.h*

class nanogui::TabWidget
 A wrapper around the widgets *TabHeader* and *StackedWidget* which hooks the two classes together.
 Inherits from *nanogui::Widget*

Public Functions

TabWidget (*Widget* *parent)

void **setActiveTab** (int tabIndex)

int **activeTab** () const

int **tabCount** () const

void **setCallback** (const std::function<void> int
> &callback) Sets the callable objects which is invoked when a tab is changed. The argument provided to the callback is the index of the new active tab.

const std::function<void (int)> &**callback**
const

Widget ***createTab** (const std::string &label)

Creates a new tab with the specified name and returns a pointer to the layer.

Widget ***createTab** (int index, const std::string &label)

void **addTab** (const std::string &label, *Widget* *tab)

Inserts a tab at the end of the tabs collection and associates it with the provided widget.

void **addTab** (int index, const std::string &label, *Widget* *tab)

Inserts a tab into the tabs collection at the specified index and associates it with the provided widget.

bool **removeTab** (const std::string &label)

Removes the tab with the specified label and returns the index of the label. Returns whether the removal was successful.

void **removeTab** (int index)

Removes the tab with the specified index.

const std::string &**tabLabelAt** (int index) const
Retrieves the label of the tab at a specific index.

int **tabLabelIndex** (const std::string &label)
Retrieves the index of a specific tab using its tab label. Returns -1 if there is no such tab.

int **tabIndex** (*Widget* *tab)
Retrieves the index of a specific tab using a widget pointer. Returns -1 if there is no such tab.

void **ensureTabVisible** (int index)
This function can be invoked to ensure that the tab with the provided index is visible, i.e. to track the given tab. Forwards to the tab header widget. This function should be used whenever the client wishes to make the tab header follow a newly added tab, as the content of the new tab is made visible but the tab header does not track it by default.

const *Widget* ***tab** (const std::string &label) const

Widget ***tab** (const std::string &label)

virtual void **performLayout** (NVGcontext *ctx)
Invoke the associated layout generator to properly place child widgets, if any.

virtual Vector2i **preferredSize** (NVGcontext *ctx) const
Compute the preferred size of the widget.

virtual void draw (NVGcontext *ctx)
 Draw the widget (and all child widgets)

Class TextBox

- Defined in *File textbox.h*

Nested Types:

- *Enum TextBox::Alignment*
- *Enum TextBox::SpinArea*

class nanogui::TextBox

Fancy text box with builtin regular expression-based validation.

Inherits from *nanogui::Widget*

Subclassed by *nanogui::FloatBox< Scalar >*, *nanogui::IntBox< Scalar >*

Public Types

enum Alignment

How to align the text in the text box.

Values:

Left

Center

Right

Public Functions

TextBox (*Widget* *parent, const std::string &value = “Untitled”)

bool **editable** () const

void **setEditable** (bool *editable*)

bool **spinnable** () const

void **setSpinnable** (bool *spinnable*)

const std::string &**value** () const

void **setValue** (const std::string &value)

const std::string &**defaultValue** () const

void **setDefaultValue** (const std::string &defaultValue)

Alignment **alignment** () const

void **setAlignment** (*Alignment* align)

const std::string &**units** () const

void **setUnits** (const std::string &*units*)

int **unitsImage** () const

void **setUnitsImage** (int *image*)

const std::string &**format** () const

Return the underlying regular expression specifying valid formats.

void **setFormat** (const std::string &*format*)

Specify a regular expression specifying valid formats.

virtual void **setTheme** (*Theme* **theme*)

Set the *Theme* used to draw this widget.

std::function<bool (const std::string &str) > **callback**

const Set the change callback.

void **setCallback** (const std::function<bool> const std::string &str
> &*callback*)

virtual bool **mouseButtonEvent** (const Vector2i &*p*, int *button*, bool *down*, int *modifiers*)

Handle a mouse button event (default implementation: propagate to children)

virtual bool **mouseMotionEvent** (const Vector2i &*p*, const Vector2i &*rel*, int *button*, int *modifiers*)

Handle a mouse motion event (default implementation: propagate to children)

virtual bool **mouseDragEvent** (const Vector2i &*p*, const Vector2i &*rel*, int *button*, int *modifiers*)

Handle a mouse drag event (default implementation: do nothing)

virtual bool **focusEvent** (bool *focused*)

Handle a focus change event (default implementation: record the focus status, but do nothing)

virtual bool **keyboardEvent** (int *key*, int *scancode*, int *action*, int *modifiers*)

Handle a keyboard event (default implementation: do nothing)

virtual bool **keyboardCharacterEvent** (unsigned int *codepoint*)

Handle text input (UTF-32 format) (default implementation: do nothing)

virtual Vector2i **preferredSize** (NVGcontext **ctx*) const

Compute the preferred size of the widget.

virtual void **draw** (NVGcontext **ctx*)

Draw the widget (and all child widgets)

virtual void **save** (*Serializer* &*s*) const

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &*s*)

Restore the state of the widget from the given *Serializer* instance.

Protected Types

enum **SpinArea**

The location (if any) for the spin area.

Values:

None
Top
Bottom

Protected Functions

bool **checkFormat** (const std::string &*input*, const std::string &*format*)
bool **copySelection** ()
void **pasteFromClipboard** ()
bool **deleteSelection** ()
void **updateCursor** (NVGcontext **ctx*, float *lastx*, const NVGglyphPosition **glyphs*, int *size*)
float **cursorIndex2Position** (int *index*, float *lastx*, const NVGglyphPosition **glyphs*, int *size*)
int **position2CursorIndex** (float *posx*, float *lastx*, const NVGglyphPosition **glyphs*, int *size*)
SpinArea **spinArea** (const Vector2i &*pos*)

Protected Attributes

bool **mEditable**
bool **mSpinnable**
bool **mCommitted**
std::string **mValue**
std::string **mDefaultValue**
Alignment **mAlignment**
std::string **mUnits**
std::string **mFormat**
int **mUnitsImage**
std::function<bool (const std::string &*str*)> **mCallback**
bool **mValidFormat**
std::string **mValueTemp**
int **mCursorPos**
int **mSelectionPos**
Vector2i **mMousePos**
Vector2i **mMouseDownPos**
Vector2i **mMouseDownDragPos**
int **mMouseDownModifier**
float **mTextOffset**
double **mLastClick**

Class Theme

- Defined in *File theme.h*

class nanogui : : **Theme**

Storage class for basic theme-related properties.

Inherits from *nanogui::Object*

Public Functions

Theme (NVGcontext *ctx)

Public Members

int **mFontNormal**

int **mFontBold**

int **mFontIcons**

int **mStandardFontSize**

int **mButtonFontSize**

int **mTextBoxFontSize**

int **mWindowCornerRadius**

int **mWindowHeaderHeight**

int **mWindowDropShadowSize**

int **mButtonCornerRadius**

float **mTabBorderWidth**

int **mTabInnerMargin**

int **mTabMinButtonWidth**

int **mTabMaxButtonWidth**

int **mTabControlWidth**

int **mTabButtonHorizontalPadding**

int **mTabButtonVerticalPadding**

Color **mDropShadow**

Color **mTransparent**

Color **mBorderDark**

Color **mBorderLight**

Color **mBorderMedium**

Color **mTextColor**

Color **mDisabledTextColor**

Color **mTextColorShadow**

Color **mIconColor**
Color **mButtonGradientTopFocused**
Color **mButtonGradientBotFocused**
Color **mButtonGradientTopUnfocused**
Color **mButtonGradientBotUnfocused**
Color **mButtonGradientTopPushed**
Color **mButtonGradientBotPushed**
Color **mWindowFillUnfocused**
Color **mWindowFillFocused**
Color **mWindowTitleUnfocused**
Color **mWindowTitleFocused**
Color **mWindowHeaderGradientTop**
Color **mWindowHeaderGradientBot**
Color **mWindowHeaderSepTop**
Color **mWindowHeaderSepBot**
Color **mWindowPopup**
Color **mWindowPopupTransparent**

Protected Functions

virtual ~Theme ()

Class ToolButton

- Defined in *File toolbutton.h*

class nanogui::ToolButton

Simple radio+toggle button with an icon.

Inherits from *nanogui::Button*

Public Functions

ToolButton (*Widget* *parent, int icon, **const** std::string &caption = "")

Class UniformBufferStd140

- Defined in *File glutil.h*

class nanogui::UniformBufferStd140

Helper class for accumulating uniform buffer data following the 'std140' packing format.

Inherits from `std::vector< uint8_t >`

Public Types

```
using Parent = std::vector<uint8_t>
```

Public Functions

```
template <typename T, typename std::enable_if< std::is_pod< T >::value, int >::type = 0>
void push_back (T value)
template <typename Derived, typename std::enable_if< Derived::IsVectorAtCompileTime, int >::type = 0>
void push_back (const Eigen::MatrixBase<Derived> &value)
template <typename Derived, typename std::enable_if<!Derived::IsVectorAtCompileTime, int >::type = 0>
void push_back (const Eigen::MatrixBase<Derived> &value, bool colMajor = true)
```

Class VScrollPanel

- Defined in *File vscrollpanel.h*

```
class nanogui::VScrollPanel
```

Adds a vertical scrollbar around a widget that is too big to fit into a certain area.

Inherits from *nanogui::Widget*

Public Functions

```
VScrollPanel (Widget *parent)
```

```
virtual void performLayout (NVGcontext *ctx)
```

Invoke the associated layout generator to properly place child widgets, if any.

```
virtual Vector2i preferredSize (NVGcontext *ctx) const
```

Compute the preferred size of the widget.

```
virtual bool mouseDragEvent (const Vector2i &p, const Vector2i &rel, int button, int modifiers)
```

Handle a mouse drag event (default implementation: do nothing)

```
virtual bool scrollEvent (const Vector2i &p, const Vector2f &rel)
```

Handle a mouse scroll event (default implementation: propagate to children)

```
virtual void draw (NVGcontext *ctx)
```

Draw the widget (and all child widgets)

```
virtual void save (Serializer &s) const
```

Save the state of the widget into the given *Serializer* instance.

```
virtual bool load (Serializer &s)
```

Restore the state of the widget from the given *Serializer* instance.

Protected Attributes

```
int mChildPreferredHeight
```

```
float mScroll
```

```
bool mUpdateLayout
```

Class Widget

- Defined in *File widget.h*

class `nanogui::Widget`

Base class of all widgets.

Widget is the base class of all widgets in `nanogui`. It can also be used as an panel to arrange an arbitrary number of child widgets using a layout generator (see *Layout*).

Inherits from *nanogui::Object*

Subclassed by *nanogui::Button*, *nanogui::CheckBox*, *nanogui::ColorWheel*, *nanogui::GLCanvas*, *nanogui::Graph*, *nanogui::ImagePanel*, *nanogui::ImageView*, *nanogui::Label*, *nanogui::ProgressBar*, *nanogui::Screen*, *nanogui::Slider*, *nanogui::StackedWidget*, *nanogui::TabHeader*, *nanogui::TabWidget*, *nanogui::TextBox*, *nanogui::VScrollPanel*, *nanogui::Window*

Public Functions

Widget (*Widget* *parent)

Construct a new widget with the given parent widget.

Widget *parent ()

Return the parent widget.

const *Widget* *parent () **const**

Return the parent widget.

void **setParent** (*Widget* *parent)

Set the parent widget.

Layout *layout ()

Return the used *Layout* generator.

const *Layout* *layout () **const**

Return the used *Layout* generator.

void **setLayout** (*Layout* *layout)

Set the used *Layout* generator.

Theme *theme ()

Return the *Theme* used to draw this widget.

const *Theme* *theme () **const**

Return the *Theme* used to draw this widget.

virtual void **setTheme** (*Theme* *theme)

Set the *Theme* used to draw this widget.

const Vector2i &position () **const**

Return the position relative to the parent widget.

void **setPosition** (**const** Vector2i &pos)

Set the position relative to the parent widget.

Vector2i **absolutePosition** () **const**

Return the absolute position on screen.

const Vector2i &**size** () **const**

Return the size of the widget.

void **setSize** (const Vector2i &*size*)

set the size of the widget

int **width** () **const**

Return the width of the widget.

void **setWidth** (int *width*)

Set the width of the widget.

int **height** () **const**

Return the height of the widget.

void **setHeight** (int *height*)

Set the height of the widget.

void **setFixedSize** (const Vector2i &*fixedSize*)

Set the fixed size of this widget.

If nonzero, components of the fixed size attribute override any values computed by a layout generator associated with this widget. Note that just setting the fixed size alone is not enough to actually change its size; this is done with a call to [setSize](#) or a call to [performLayout\(\)](#) in the parent widget.

const Vector2i &**fixedSize** () **const**

Return the fixed size (see [setFixedSize\(\)](#))

int **fixedWidth** () **const**

int **fixedHeight** () **const**

void **setFixedWidth** (int *width*)

Set the fixed width (see [setFixedSize\(\)](#))

void **setFixedHeight** (int *height*)

Set the fixed height (see [setFixedSize\(\)](#))

bool **visible** () **const**

Return whether or not the widget is currently visible (assuming all parents are visible)

void **setVisible** (bool *visible*)

Set whether or not the widget is currently visible (assuming all parents are visible)

bool **visibleRecursive** () **const**

Check if this widget is currently visible, taking parent widgets into account.

int **childCount** () **const**

Return the number of child widgets.

const std::vector<[Widget](#) *> &**children** () **const**

Return the list of child widgets of the current widget.

virtual void **addChild** (int *index*, [Widget](#) **widget*)

Add a child widget to the current widget at the specified index.

This function almost never needs to be called by hand, since the constructor of [Widget](#) automatically adds the current widget to its parent

void **addChild** (*Widget* **widget*)
 Convenience function which appends a widget at the end.

void **removeChild** (int *index*)
 Remove a child widget by index.

void **removeChild** (const *Widget* **widget*)
 Remove a child widget by value.

const *Widget* ***childAt** (int *index*) const
 Retrieves the child at the specific position.

Widget ***childAt** (int *index*)
 Retrieves the child at the specific position.

int **childIndex** (*Widget* **widget*) const
 Returns the index of a specific child or -1 if not found.

template <typename WidgetClass, typename... Args>
 WidgetClass ***add** (const Args&... *args*)
 Variadic shorthand notation to construct and add a child widget.

Window ***window** ()
 Walk up the hierarchy and return the parent window.

void **setId** (const std::string &*id*)
 Associate this widget with an ID value (optional)

const std::string &**id** () const
 Return the ID value associated with this widget, if any.

bool **enabled** () const
 Return whether or not this widget is currently enabled.

void **setEnabled** (bool *enabled*)
 Set whether or not this widget is currently enabled.

bool **focused** () const
 Return whether or not this widget is currently focused.

void **setFocused** (bool *focused*)
 Set whether or not this widget is currently focused.

void **requestFocus** ()
 Request the focus to be moved to this widget.

const std::string &**tooltip** () const

void **setTooltip** (const std::string &*tooltip*)

int **fontSize** () const
 Return current font size. If not set the default of the current theme will be returned.

void **setFontSize** (int *fontSize*)
 Set the font size of this widget.

bool **hasFontSize** () const
 Return whether the font size is explicitly specified for this widget.

Cursor **cursor** () **const**

Return a pointer to the cursor of the widget.

void **setCursor** (*Cursor* cursor)

Set the cursor of the widget.

bool **contains** (const Vector2i &p) **const**

Check if the widget contains a certain position.

Widget ***findWidget** (const Vector2i &p)

Determine the widget located at the given position value (recursive)

virtual bool **mouseButtonEvent** (const Vector2i &p, int button, bool down, int modifiers)

Handle a mouse button event (default implementation: propagate to children)

virtual bool **mouseMotionEvent** (const Vector2i &p, const Vector2i &rel, int button, int modifiers)

Handle a mouse motion event (default implementation: propagate to children)

virtual bool **mouseDragEvent** (const Vector2i &p, const Vector2i &rel, int button, int modifiers)

Handle a mouse drag event (default implementation: do nothing)

virtual bool **mouseEnterEvent** (const Vector2i &p, bool enter)

Handle a mouse enter/leave event (default implementation: record this fact, but do nothing)

virtual bool **scrollEvent** (const Vector2i &p, const Vector2f &rel)

Handle a mouse scroll event (default implementation: propagate to children)

virtual bool **focusEvent** (bool focused)

Handle a focus change event (default implementation: record the focus status, but do nothing)

virtual bool **keyboardEvent** (int key, int scancode, int action, int modifiers)

Handle a keyboard event (default implementation: do nothing)

virtual bool **keyboardCharacterEvent** (unsigned int codepoint)

Handle text input (UTF-32 format) (default implementation: do nothing)

virtual Vector2i **preferredSize** (NVGcontext *ctx) **const**

Compute the preferred size of the widget.

virtual void **performLayout** (NVGcontext *ctx)

Invoke the associated layout generator to properly place child widgets, if any.

virtual void **draw** (NVGcontext *ctx)

Draw the widget (and all child widgets)

virtual void **save** (*Serializer* &s) **const**

Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)

Restore the state of the widget from the given *Serializer* instance.

Protected Functions

virtual ~**Widget** ()

Free all resources used by the widget and any children.

Protected Attributes

```

Widget *mParent
ref<Theme> mTheme
ref<Layout> mLayout
std::string mId
Vector2i mPos
Vector2i mSize
Vector2i mFixedSize
std::vector<Widget *> mChildren
bool mVisible
bool mEnabled
bool mFocused
bool mMouseFocus
std::string mTooltip
int mFontSize
Cursor mCursor

```

Class Window

- Defined in *File window.h*

```

class nanogui::Window
    Top-level window widget.

    Inherits from nanogui::Widget

    Subclassed by nanogui::MessageDialog, nanogui::Popup

```

Public Functions

```

Window (Widget *parent, const std::string &title = "Untitled")

```

```

const std::string &title () const
    Return the window title.

```

```

void setTitle (const std::string &title)
    Set the window title.

```

```

bool modal () const
    Is this a modal dialog?

```

```

void setModal (bool modal)
    Set whether or not this is a modal dialog.

```

```

Widget *buttonPanel ()
    Return the panel used to house window buttons.

```

void **dispose** ()
 Dispose the window.

void **center** ()
 Center the window in the current *Screen*.

virtual void **draw** (NVGcontext *ctx)
 Draw the window.

virtual bool **mouseDragEvent** (const Vector2i &p, const Vector2i &rel, int button, int modifiers)
 Handle window drag events.

virtual bool **mouseButtonEvent** (const Vector2i &p, int button, bool down, int modifiers)
 Handle mouse events recursively and bring the current window to the top.

virtual bool **scrollEvent** (const Vector2i &p, const Vector2f &rel)
 Accept scroll events and propagate them to the widget under the mouse cursor.

virtual Vector2i **preferredSize** (NVGcontext *ctx) **const**
 Compute the preferred size of the widget.

virtual void **performLayout** (NVGcontext *ctx)
 Invoke the associated layout generator to properly place child widgets, if any.

virtual void **save** (*Serializer* &s) **const**
 Save the state of the widget into the given *Serializer* instance.

virtual bool **load** (*Serializer* &s)
 Restore the state of the widget from the given *Serializer* instance.

Protected Functions

virtual void **refreshRelativePlacement** ()
 Internal helper function to maintain nested window position values; overridden in *Popup*.

Protected Attributes

std::string **mTitle**
Widget ***mButtonPanel**
 bool **mModal**
 bool **mDrag**

Enums

Enum *TextBox::Alignment*

- Defined in *File textbox.h*
- Nested type of *Class TextBox*

enum nanogui::TextBox::Alignment

How to align the text in the text box.

Values:

Left

Center

Right

Enum TabHeader::ClickLocation

- Defined in *File tabheader.h*
- Nested type of *Class TabHeader*

enum nanogui::TabHeader::ClickLocation

The location in which the *Widget* will be facing.

Values:

LeftControls

RightControls

TabButtons

Enum Button::Flags

- Defined in *File button.h*
- Nested type of *Class Button*

enum nanogui::Button::Flags

Flags to specify the button behavior (can be combined with binary OR)

Values:

NormalButton = (1 << 0)

RadioButton = (1 << 1)

ToggleButton = (1 << 2)

PopupButton = (1 << 3)

Enum Button::IconPosition

- Defined in *File button.h*
- Nested type of *Class Button*

enum nanogui::Button::IconPosition

The available icon positions.

Values:

Left

LeftCentered

RightCentered

Right

Enum Alignment

- Defined in *File layout.h*

enum nanogui::Alignment

The different kinds of alignments a layout can perform.

Values:

Minimum = 0

Middle

Maximum

Fill

Enum Cursor

- Defined in *File common.h*

enum nanogui::Cursor

Cursor shapes available to use in GLFW.

Values:

Arrow = 0

IBeam

Crosshair

Hand

HResize

VResize

CursorCount

Not a cursor should always be last: enables a loop over the cursor types.

Enum Orientation

- Defined in *File layout.h*

enum nanogui::Orientation

The direction of data flow for a layout.

Values:

Horizontal = 0

Vertical

Enum `ColorWheel::Region`

- Defined in *File colorwheel.h*
- Nested type of *Class ColorWheel*

enum `nanogui::ColorWheel::Region`

Values:

None = 0

InnerTriangle = 1

OuterCircle = 2

Both = 3

Enum `Popup::Side`

- Defined in *File popup.h*
- Nested type of *Class Popup*

enum `nanogui::Popup::Side`

Values:

Left = 0

Right

Enum `TextBox::SpinArea`

- Defined in *File textbox.h*
- Nested type of *Class TextBox*

enum `nanogui::TextBox::SpinArea`

The location (if any) for the spin area.

Values:

None

Top

Bottom

Enum `MessageDialog::Type`

- Defined in *File messagedialog.h*
- Nested type of *Class MessageDialog*

enum `nanogui::MessageDialog::Type`

Classification of the type of message this *MessageDialog* represents.

Values:

Information

Question

Warning

Functions

Function `__nanogui_get_image`

- Defined in *File common.h*

```
int nanogui::__nanogui_get_image(NVGcontext *ctx, const std::string &name, uint8_t *data,
                                uint32_t size)
    Helper function used by nvglImageIcon.
```

Function `active`

- Defined in *File common.h*

```
bool nanogui::active()
    Return whether or not a main loop is currently active.
```

Function `chdir_to_bundle_parent`

- Defined in *File common.h*

```
void nanogui::chdir_to_bundle_parent()
    Move to the application bundle's parent directory.

    This is function is convenient when deploying .app bundles on OSX. It adjusts the file path to the parent directory
    containing the bundle.
```

Function `file_dialog`

- Defined in *File common.h*

```
std::string nanogui::file_dialog(const std::vector<std::pair<std::string, std::string>> &filetypes, bool
                                save)
    Open a native file open/save dialog.
```

Parameters

- `filetypes`: Pairs of permissible formats with descriptions like ("png", "Portable Network Graphics").
- `save`: Set to `true` if you would like subsequent file dialogs to open at whatever folder they were in when they close this one.

Function `frustum`

- Defined in *File glutil.h*

```
Matrix4f nanogui::frustum(float left, float right, float bottom, float top, float nearVal, float farVal)
    Creates a perspective projection matrix.
```

Parameters

- `left`: The left border of the viewport.
- `right`: The right border of the viewport.
- `bottom`: The bottom border of the viewport.
- `top`: The top border of the viewport.
- `nearVal`: The near plane.
- `farVal`: The far plane.

Function `init`

- Defined in *File common.h*

`void nanogui::init()`

Static initialization; should be called once before invoking **any** NanoGUI functions **if** you are having NanoGUI manage OpenGL / GLFW. This method is effectively a wrapper call to `glfwInit()`, so if you are managing OpenGL / GLFW on your own *do not call this method*.

Refer to [Example 3](#) for how you might go about managing OpenGL and GLFW on your own, while still using NanoGUI's classes.

Function `leave`

- Defined in *File common.h*

`void nanogui::leave()`

Request the application main loop to terminate (e.g. if you detached mainloop).

Function `loadImageDirectory`

- Defined in *File common.h*

`std::vector<std::pair<int, std::string>> nanogui::loadImageDirectory(NVGcontext *ctx, const std::string &path)`

Load a directory of PNG images and upload them to the GPU (suitable for use with *ImagePanel*)

Function `lookAt`

- Defined in *File glutil.h*

`Matrix4f nanogui::lookAt(const Vector3f &origin, const Vector3f &target, const Vector3f &up)`

Creates a “look at” matrix that describes the position and orientation of e.g. a camera.

Warning: These are used to form an orthonormal basis. The first basis vector is defined as `f = (target - origin).normalized()`.

Parameters

- `origin`: The position of the camera.
- `target`: The gaze target of the camera.

- `up`: The up vector of the camera.

Function `mainloop`

- Defined in *File `common.h`*

`void nanogui : : mainloop (int refresh = 50)`

Enter the application main loop.

Remark Unfortunately, Mac OS X strictly requires all event processing to take place on the application's main thread, which is fundamentally incompatible with this type of approach. Thus, NanoGUI relies on a rather crazy workaround on Mac OS (kudos to Dmitriy Morozov): `mainloop()` launches a new thread as before but then uses `libcoro` to swap the thread execution environment (stack, registers, ..) with the main thread. This means that the main application thread is hijacked and processes events in the main loop to satisfy the requirements on Mac OS, while the thread that actually returns from this function is the newly created one (paradoxical, as that may seem). Deleting or `join()`ing the returned handle causes application to wait for the termination of the main loop and then swap the two thread environments back into their initial configuration.

Parameters

- `refresh`: NanoGUI issues a redraw call whenever an keyboard/mouse/.. event is received. In the absence of any external events, it enforces a redraw once every `refresh` milliseconds. To disable the refresh timer, specify a negative value here.
- `detach`: This parameter only exists in the Python bindings. When the active *Screen* instance is provided via the `detach` parameter, the `mainloop()` function becomes non-blocking and returns immediately (in this case, the main loop runs in parallel on a newly created thread). This feature is convenient for prototyping user interfaces on an interactive Python command prompt. When `detach != None`, the function returns an opaque handle that will release any resources allocated by the created thread when the handle's `join()` method is invoked (or when it is garbage collected).

Function `nvglIsFontIcon`

- Defined in *File `opengl.h`*

`bool nanogui : : nvglIsFontIcon (int value)`

Determine whether an icon ID is a font-based icon (e.g. from the `entypo.ttf` font)

Function `nvglIsImageIcon`

- Defined in *File `opengl.h`*

`bool nanogui : : nvglIsImageIcon (int value)`

Determine whether an icon ID is a texture loaded via `nvglImageIcon`.

Function `ortho`

- Defined in *File `glutil.h`*

`Matrix4f nanogui : : ortho (float left, float right, float bottom, float top, float nearVal, float farVal)`

Creates an orthographic projection matrix.

Parameters

- `left`: The left border of the viewport.
- `right`: The right border of the viewport.
- `bottom`: The bottom border of the viewport.
- `top`: The top border of the viewport.
- `nearVal`: The near plane.
- `farVal`: The far plane.

Function `project`

- Defined in *File `glutil.h`*

`Vector3f nanogui::project (const Vector3f &obj, const Matrix4f &model, const Matrix4f &proj, const Vector2i &viewportSize)`

Projects the vector `obj` into the specified viewport.

Performs a homogeneous transformation of a vector into “screen space”, as defined by the provided model and projection matrices, and the dimensions of the viewport.

Parameters

- `obj`: The vector being transformed.
- `model`: The model matrix.
- `proj`: The projection matrix.
- `viewportSize`: The dimensions of the viewport to project into.

Function `scale`

- Defined in *File `glutil.h`*

`Matrix4f nanogui::scale (const Vector3f &v)`

Construct homogeneous coordinate scaling matrix.

Returns a 3D homogeneous coordinate matrix that scales the X, Y, and Z components with the corresponding entries of the 3D vector `v`. The w component is left unchanged

Parameters

- `v`: The vector representing the scaling for each axis.

Function `shutdown`

- Defined in *File `common.h`*

`void nanogui::shutdown ()`

Static shutdown; should be called before the application terminates.

Function translate

- Defined in *File glutil.h*

`Matrix4f nanogui::translate (const Vector3f &v)`
Construct homogeneous coordinate translation matrix.

Returns a 3D homogeneous coordinate matrix that translates the X, Y, and Z components by the corresponding entries of the 3D vector `v`. The `w` component is left unchanged

Parameters

- `v`: The vector representing the translation for each axis.

Function unproject

- Defined in *File glutil.h*

`Vector3f nanogui::unproject (const Vector3f &win, const Matrix4f &model, const Matrix4f &proj, const Vector2i &viewportSize)`
Unprojects the vector `win` out of the specified viewport.

The reverse transformation of project use the same matrices and viewport dimensions to easily transition between the two spaces.

Parameters

- `win`: The vector being transformed out of “screen space”.
- `model`: The model matrix.
- `proj`: The projection matrix.
- `viewportSize`: The dimensions of the viewport to project out of.

Function utf8

- Defined in *File common.h*

`std::array<char, 8> nanogui::utf8 (int c)`
Convert a single UTF32 character code to UTF8.

NanoGUI uses this to convert the icon character codes defined in *File entypo.h*.

Parameters

- `c`: The UTF32 character to be converted.

Defines

Define GL_HALF_FLOAT

- Defined in *File glutil.h*

`GL_HALF_FLOAT 0x140B`

Define `NAMESPACE_BEGIN`

- Defined in *File common.h*

NAMESPACE_BEGIN (name) namespace name {
Convenience macro for namespace declarations.

The macro `NAMESPACE_BEGIN(nanogui)` will expand to `namespace nanogui {`. This is done to hide the namespace scope from editors and C++ code formatting tools that may otherwise indent the entire file. The corresponding `NAMESPACE_END` macro also lists the namespace name for improved readability.

Parameters

- name: The name of the namespace scope to open

Define `NAMESPACE_END`

- Defined in *File common.h*

NAMESPACE_END (name) }
Convenience macro for namespace declarations.

Closes a namespace (counterpart to `NAMESPACE_BEGIN`) `NAMESPACE_END(nanogui)` will expand to only `}`.

Parameters

- name: The name of the namespace scope to close

Define `NANOGUI_EXPORT`

- Defined in *File common.h*

NANOGUI_EXPORT

If the build flag `NANOGUI_SHARED` is defined, this directive will expand to be the platform specific shared library import / export command depending on the compilation stage. If undefined, it expands to nothing. **Do not** define this directive on your own.

Define `NANOGUI_FORCE_DISCRETE_GPU`

- Defined in *File common.h*

NANOGUI_FORCE_DISCRETE_GPU

On Windows, exports `AmdPowerXpressRequestHighPerformance` and `NvOptimusEnablement` as 1.

Define `NANOGUI_LAYOUT_OVERLOADS`

- Defined in *File python.h*

```
NANOGUI_LAYOUT_OVERLOADS (Parent)          ::nanogui::Vector2i      preferredSize(NVGcontext
    *ctx,  const  ::nanogui::Widget *widget) const { \ PY-
    BIND11_OVERLOAD(::nanogui::Vector2i, Parent, preferredSize, ctx,
    widget); \ } \ void performLayout(NVGcontext *ctx, ::nanogui::Widget
    *widget) const { \ PYBIND11_OVERLOAD(void, Parent, performLay-
    out, ctx, widget); \ }
```

Provides a PYBIND11_OVERLOAD for any relevant Layout items that need to be bound.

Define NANOGUI_SCREEN_OVERLOADS

- Defined in *File python.h*

```
NANOGUI_SCREEN_OVERLOADS (Parent) virtual void drawAll() { \ PYBIND11_OVERLOAD(void,
    Parent, drawAll); \ } \ virtual void drawContents() { \ PY-
    BIND11_OVERLOAD(void, Parent, drawContents); \ } \ virtual
    bool dropEvent(const std::vector<std::string> &filenames) { \ PY-
    BIND11_OVERLOAD(bool, Parent, dropEvent, filenames); \ } \
    virtual bool resizeEvent(const ::nanogui::Vector2i &size) { \ PY-
    BIND11_OVERLOAD(bool, Parent, resizeEvent, size); \ }
```

Provides a PYBIND11_OVERLOAD for any relevant Screen items that need to be bound.

Define NANOGUI_SNPRINTF

- Defined in *File compat.h*

NANOGUI_SNPRINTF snprintf

Platform dependent snprintf (`_snprintf` for MSVC, `snprintf` otherwise).

Define NANOGUI_WIDGET_OVERLOADS

- Defined in *File python.h*

NANOGUI_WIDGET_OVERLOADS (Parent)

Provides a PYBIND11_OVERLOAD for any relevant Widget items that need to be bound.

Define nvglImageIcon

- Defined in *File common.h*

nvglImageIcon (ctx, name) nanogui::__nanogui_get_image(ctx, #name, name##_png, name##_png_size)

Convenience function for instanting a PNG icon from the application's data segment (via `bin2c`)

Define SYSTEM_COMMAND_MOD

- Defined in *File common.h*

SYSTEM_COMMAND_MOD GLFW_MOD_CONTROL

If on OSX, maps to `GLFW_MOD_SUPER`. Otherwise, maps to `GLFW_MOD_CONTROL`.

Typedefs

Typedef nanogui::MatrixXu

- Defined in *File common.h*

typedef Eigen::Matrix<uint32_t, Eigen::Dynamic, Eigen::Dynamic> nanogui::MatrixXu

Convenience typedef for things like index buffers. You would use it the same as Eigen::MatrixXf, only it is storing uint32_t instead of float.

Directories

Directory include

Subdirectories

- *Directory nanogui*

Directory nanogui

Subdirectories

- *Directory serializer*

Files

- *File button.h*
- *File checkbox.h*
- *File colorpicker.h*
- *File colorwheel.h*
- *File combobox.h*
- *File common.h*
- *File compat.h*
- *File entypo.h*
- *File formhelper.h*
- *File glcanvas.h*
- *File glutil.h*
- *File graph.h*
- *File imagepanel.h*
- *File imageview.h*
- *File label.h*
- *File layout.h*

- *File messagedialog.h*
- *File nanogui.h*
- *File object.h*
- *File opengl.h*
- *File popup.h*
- *File popupbutton.h*
- *File progressbar.h*
- *File python.h*
- *File screen.h*
- *File slider.h*
- *File stackedwidget.h*
- *File tabheader.h*
- *File tabwidget.h*
- *File textbox.h*
- *File theme.h*
- *File toolbutton.h*
- *File vscrollpanel.h*
- *File widget.h*
- *File window.h*

Directory serializer

Files

- *File core.h*
- *File opengl.h*
- *File sparse.h*

Files

File button.h

Definition (`include/nanogui/button.h`)

Program Listing for File button.h

- Return to documentation for *File button.h*

```

/*
    nanogui/button.h -- [Normal/Toggle/Radio/Popup] Button widget

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)
class NANOGUI_EXPORT Button : public Widget {
public:
    enum Flags {
        NormalButton = (1 << 0), // 1
        RadioButton = (1 << 1), // 2
        ToggleButton = (1 << 2), // 4
        PopupButton = (1 << 3) // 8
    };

    enum class IconPosition {
        Left,
        LeftCentered,
        RightCentered,
        Right
    };

    Button(Widget *parent, const std::string &caption = "Untitled", int icon = 0);

    const std::string &caption() const { return mCaption; }
    void setCaption(const std::string &caption) { mCaption = caption; }

    const Color &backgroundcolor() const { return mBackgroundColor; }
    void setBackgroundColor(const Color &backgroundcolor) { mBackgroundColor = _
↪backgroundcolor; }

    const Color &textcolor() const { return mTextColor; }
    void setTextColor(const Color &textcolor) { mTextColor = textColor; }

    int icon() const { return mIcon; }
    void setIcon(int icon) { mIcon = icon; }

    int flags() const { return mFlags; }
    void setFlags(int buttonFlags) { mFlags = buttonFlags; }

    IconPosition iconPosition() const { return mIconPosition; }
    void setIconPosition(IconPosition iconPosition) { mIconPosition = iconPosition; }

    bool pushed() const { return mPushed; }
    void setPushed(bool pushed) { mPushed = pushed; }

    std::function<void()> callback() const { return mCallback; }
    void setCallback(const std::function<void()> &callback) { mCallback = callback; }

```

```
    std::function<void(bool)> changeCallback() const { return mChangeCallback; }
    void setChangeCallback(const std::function<void(bool)> &callback) {
↪mChangeCallback = callback; }

    void setButtonGroup(const std::vector<Button *> &buttonGroup) { mButtonGroup =
↪buttonGroup; }
    const std::vector<Button *> &buttonGroup() const { return mButtonGroup; }

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↪modifiers) override;
    virtual void draw(NVGcontext *ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    std::string mCaption;
    int mIcon;
    IconPosition mIconPosition;
    bool mPushed;
    int mFlags;
    Color mBackgroundColor;
    Color mTextColor;
    std::function<void()> mCallback;
    std::function<void(bool)> mChangeCallback;
    std::vector<Button *> mButtonGroup;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*
- *File popupbutton.h*
- *File toolbutton.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Button*

Enums

- *Enum Button::Flags*
- *Enum Button::IconPosition*

File checkbox.h

Definition (`include/nanogui/checkbox.h`)

Program Listing for File checkbox.h

- Return to documentation for *File checkbox.h*

```

/*
    nanogui/checkbox.h -- Two-state check box widget

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT CheckBox : public Widget {
public:
    CheckBox(Widget *parent, const std::string &caption = "Untitled",
             const std::function<void(bool)> &callback = std::function<void(bool)>());

    const std::string &caption() const { return mCaption; }
    void setCaption(const std::string &caption) { mCaption = caption; }

    const bool &checked() const { return mChecked; }
    void setChecked(const bool &checked) { mChecked = checked; }

    const bool &pushed() const { return mPushed; }
    void setPushed(const bool &pushed) { mPushed = pushed; }

    std::function<void(bool)> callback() const { return mCallback; }
    void setCallback(const std::function<void(bool)> &callback) { mCallback =
↳callback; }

    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↳modifiers) override;
    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void draw(NVGcontext *ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;

```

```
protected:
    std::string mCaption;
    bool mPushed, mChecked;
    std::function<void(bool)> mCallback;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class CheckBox*

File colorpicker.h

Definition (`include/nanogui/colorpicker.h`)

Program Listing for File colorpicker.h

- [Return to documentation for *File colorpicker.h*](#)

```
/*
    nanogui/colorpicker.h -- push button with a popup to tweak a color value

    This widget was contributed by Christian Schueller.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once
```

```

#include <nanogui/popupbutton.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT ColorPicker : public PopupButton {
public:
    ColorPicker(Widget *parent, const Color& color = Color(1.0f, 0.0f, 0.0f, 1.0f));

    std::function<void(const Color &)> callback() const { return
↪mCallback; }
    void setCallback(const std::function<void(const Color &)> &callback) { mCallback
↪= callback; }

    Color color() const;
    void setColor(const Color& color);
protected:
    std::function<void(const Color &)> mCallback;
    ColorWheel *mColorWheel;
    Button *mPickButton;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- `nanogui/popupbutton.h` (*File popupbutton.h*)

Included By

- *File formhelper.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ColorPicker*

File colorwheel.h

Definition (`include/nanogui/colorwheel.h`)

Program Listing for File colorwheel.h

- Return to documentation for *File colorwheel.h*

```
/*
    nanogui/colorwheel.h -- fancy analog widget to select a color value

    This widget was contributed by Dmitriy Morozov.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT ColorWheel : public Widget {
public:
    ColorWheel(Widget *parent, const Color& color = Color(1.0f, 0.0f, 0.0f, 1.0f));

    std::function<void(const Color &)> callback() const { return mCallback; }
    void setCallback(const std::function<void(const Color &)> &callback) { mCallback = callback; }

    Color color() const;
    void setColor(const Color& color);

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void draw(NVGcontext *ctx) override;
    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int modifiers) override;
    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button, int modifiers) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
private:
    enum Region {
        None = 0,
        InnerTriangle = 1,
        OuterCircle = 2,
        Both = 3
    };

    Color hue2rgb(float h) const;
    Region adjustPosition(const Vector2i &p, Region consideredRegions = Both);
protected:
    float mHue;
    float mWhite;
    float mBlack;
    Region mDragRegion;
    std::function<void(const Color &)> mCallback;
public:
```

```

    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)

```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ColorWheel*

Enums

- *Enum ColorWheel::Region*

File combobox.h

Definition (`include/nanogui/combobox.h`)

Program Listing for File combobox.h

- [Return to documentation for *File combobox.h*](#)

```

/*
    nanogui/combobox.h -- simple combo box widget based on a popup button

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/popupbutton.h>

NAMESPACE_BEGIN (nanogui)

```

```
class NANOGUI_EXPORT ComboBox : public PopupButton {
public:
    ComboBox(Widget *parent);

    ComboBox(Widget *parent, const std::vector<std::string> &items);

    ComboBox(Widget *parent, const std::vector<std::string> &items,
              const std::vector<std::string> &itemsShort);

    std::function<void(int)> callback() const { return mCallback; }
    void setCallback(const std::function<void(int)> &callback) { mCallback = callback; }
    ↪ }

    int selectedIndex() const { return mSelectedIndex; }
    void setSelectedIndex(int idx);

    void setItems(const std::vector<std::string> &items, const std::vector
    ↪<std::string> &itemsShort);
    void setItems(const std::vector<std::string> &items) { setItems(items, items); }
    const std::vector<std::string> &items() const { return mItems; }
    const std::vector<std::string> &itemsShort() const { return mItemsShort; }

    virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    std::vector<std::string> mItems, mItemsShort;
    std::function<void(int)> mCallback;
    int mSelectedIndex;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/popupbutton.h` (*File popupbutton.h*)

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ComboBox*

File common.h

Definition (include/nanogui/common.h)

Program Listing for File common.h

- [Return to documentation for *File common.h*](#)

```

/*
    nanogui/common.h -- common definitions used by NanoGUI

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <Eigen/Core>
#include <stdint.h>
#include <array>
#include <vector>

/* Set to 1 to draw boxes around widgets */
// #define NANOGUI_SHOW_WIDGET_BOUNDS 1

#if !defined(NAMESPACE_BEGIN) || defined(DOXYGEN_DOCUMENTATION_BUILD)

    #define NAMESPACE_BEGIN(name) namespace name {
#elseif
    #define NAMESPACE_BEGIN(name) namespace name {
#endif

    #define NAMESPACE_END(name) }
#elseif
    #define NAMESPACE_END(name) }
#endif

#if defined(NANOGUI_SHARED)
#   if defined(_WIN32)
#       if defined(NANOGUI_BUILD)
#           define NANOGUI_EXPORT __declspec(dllexport)
#       else
#           define NANOGUI_EXPORT __declspec(dllimport)
#       endif
#   elif defined(NANOGUI_BUILD)
#       define NANOGUI_EXPORT __attribute__((visibility("default")))
#   else
#       define NANOGUI_EXPORT
#   endif
#else
#       define NANOGUI_EXPORT

```

```
#endif

/* Force usage of discrete GPU on laptops (macro must be invoked in main application)
↳ */
#if defined(_WIN32)
#define NANOGUI_FORCE_DISCRETE_GPU() \
    extern "C" { \
        __declspec(dllexport) int AmdPowerXpressRequestHighPerformance = 1; \
        __declspec(dllexport) int NvOptimusEnablement = 1; \
    }
#else

#define NANOGUI_FORCE_DISCRETE_GPU()
#endif

#if defined(_WIN32)
#if defined(NANOGUI_BUILD)
/* Quench a few warnings on when compiling NanoGUI on Windows */
#pragma warning(disable : 4127) // warning C4127: conditional expression is constant
#pragma warning(disable : 4244) // warning C4244: conversion from X to Y, possible
↳ loss of data
#endif
#pragma warning(disable : 4251) // warning C4251: class X needs to have dll-interface
↳ to be used by clients of class Y
#pragma warning(disable : 4714) // warning C4714: function X marked as __forceinline
↳ not inlined
#endif

// These will produce broken links in the docs build
#ifndef DOXYGEN_SHOULD_SKIP_THIS

struct NVGcontext { /* Opaque handle type, never de-referenced within NanoGUI */ };
struct GLFWwindow { /* Opaque handle type, never de-referenced within NanoGUI */ };

struct NVGcolor;
struct NVGglyphPosition;
struct GLFWcursor;

#endif // DOXYGEN_SHOULD_SKIP_THIS

// Define command key for windows/mac/linux
#ifdef __APPLE__
#define SYSTEM_COMMAND_MOD GLFW_MOD_SUPER
#else
#define SYSTEM_COMMAND_MOD GLFW_MOD_CONTROL
#endif

NAMESPACE_BEGIN(nanogui)

enum class Cursor {
    Arrow = 0,
    IBeam,
    Crosshair,
    Hand,
    HResize,
    VResize,
    CursorCount
};
```



```

/* Import some common Eigen types */
using Eigen::Vector2f;
using Eigen::Vector3f;
using Eigen::Vector4f;
using Eigen::Vector2i;
using Eigen::Vector3i;
using Eigen::Vector4i;
using Eigen::Matrix3f;
using Eigen::Matrix4f;
using Eigen::VectorXf;
using Eigen::MatrixXf;

typedef Eigen::Matrix<uint32_t, Eigen::Dynamic, Eigen::Dynamic> MatrixXu;

class Color : public Eigen::Vector4f {
    typedef Eigen::Vector4f Base;
public:
    Color() : Color(0, 0, 0, 0) {}

    Color(const Eigen::Vector4f &color) : Eigen::Vector4f(color) { }

    Color(const Eigen::Vector3f &color, float alpha)
        : Color(color(0), color(1), color(2), alpha) { }

    Color(const Eigen::Vector3i &color, int alpha)
        : Color(color.cast<float>() / 255.f, alpha / 255.f) { }

    Color(const Eigen::Vector3f &color) : Color(color, 1.0f) {}

    Color(const Eigen::Vector3i &color)
        : Color((Vector3f)(color.cast<float>() / 255.f)) { }

    Color(const Eigen::Vector4i &color)
        : Color((Vector4f)(color.cast<float>() / 255.f)) { }

    Color(float intensity, float alpha)
        : Color(Vector3f::Constant(intensity), alpha) { }

    Color(int intensity, int alpha)
        : Color(Vector3i::Constant(intensity), alpha) { }

    Color(float r, float g, float b, float a) : Color(Vector4f(r, g, b, a)) { }

    Color(int r, int g, int b, int a) : Color(Vector4i(r, g, b, a)) { }

    template <typename Derived> Color(const Eigen::MatrixBase<Derived>& p)
        : Base(p) { }

    template <typename Derived> Color &operator=(const Eigen::MatrixBase<Derived>& p)
    ↪{
        this->Base::operator=(p);
        return *this;
    }

    float &r() { return x(); }
    const float &r() const { return x(); }
    float &g() { return y(); }

```

```
const float &g() const { return y(); }
float &b() { return z(); }
const float &b() const { return z(); }

Color contrastingColor() const {
    float luminance = cwiseProduct(Color(0.299f, 0.587f, 0.144f, 0.f)).sum();
    return Color(luminance < 0.5f ? 1.f : 0.f, 1.f);
}

inline operator const NVGcolor &() const;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

// skip the forward declarations for the docs
#ifdef DOXYGEN_SHOULD_SKIP_THIS

/* Forward declarations */
template <typename T> class ref;
class AdvancedGridLayout;
class BoxLayout;
class Button;
class CheckBox;
class ColorWheel;
class ColorPicker;
class ComboBox;
class GLFramebuffer;
class GLShader;
class GridLayout;
class GroupLayout;
class ImagePanel;
class ImageView;
class Label;
class Layout;
class MessageDialog;
class Object;
class Popup;
class PopupButton;
class ProgressBar;
class Screen;
class Serializer;
class Slider;
class StackedWidget;
class TabHeader;
class TabWidget;
class TextBox;
class GLCanvas;
class Theme;
class ToolButton;
class VScrollPanel;
class Widget;
class Window;

#endif // DOXYGEN_SHOULD_SKIP_THIS

extern NANOGUI_EXPORT void init();

extern NANOGUI_EXPORT void shutdown();
```

```

extern NANOGUI_EXPORT void mainloop(int refresh = 50);

extern NANOGUI_EXPORT void leave();

extern NANOGUI_EXPORT bool active();

extern NANOGUI_EXPORT std::string
file_dialog(const std::vector<std::pair<std::string, std::string>> &filetypes,
            bool save);

#ifdef __APPLE__ || defined(DOXYGEN_DOCUMENTATION_BUILD)

extern NANOGUI_EXPORT void chdir_to_bundle_parent();
#endif

extern NANOGUI_EXPORT std::array<char, 8> utf8(int c);

extern NANOGUI_EXPORT std::vector<std::pair<int, std::string>>
loadImageDirectory(NVGcontext *ctx, const std::string &path);

#define nvgImageIcon(ctx, name) nanogui::__nanogui_get_image(ctx, #name, name##_png, \
↳ name##_png_size)
extern NANOGUI_EXPORT int __nanogui_get_image(NVGcontext *ctx, const std::string &
↳ name, uint8_t *data, uint32_t size);

NAMESPACE_END(nanogui)

```

Includes

- Eigen/Core
- array
- stdint.h
- vector

Included By

- *File nanogui.h*
- *File object.h*
- *File opengl.h*
- *File python.h*
- *File theme.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Color*

Enums

- *Enum Cursor*

Functions

- *Function __nanogui_get_image*
- *Function active*
- *Function chdir_to_bundle_parent*
- *Function file_dialog*
- *Function init*
- *Function leave*
- *Function loadImageDirectory*
- *Function mainloop*
- *Function shutdown*
- *Function utf8*

Defines

- *Define NAMESPACE_BEGIN*
- *Define NAMESPACE_END*
- *Define NANOGUI_EXPORT*
- *Define NANOGUI_FORCE_DISCRETE_GPU*
- *Define nvglImageIcon*
- *Define SYSTEM_COMMAND_MOD*

Typedefs

- *Typedef nanogui::MatrixXu*

File compat.h

Definition (`include/nanogui/compat.h`)

Program Listing for File compat.h

- [Return to documentation for *File compat.h*](#)

```

/*
    nanogui/compat.h -- Compatibility layer

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <stdio.h>

#ifdef _MSC_VER
#define NANOGUI_SNPRINTF _snprintf
#else
#define NANOGUI_SNPRINTF snprintf
#endif

```

Includes

- `stdio.h`

Included By

- *File layout.h*
- *File textbox.h*

Defines

- *Define NANOGUI_SNPRINTF*

File core.h

Definition (`include/nanogui/serializer/core.h`)

Program Listing for File core.h

- [Return to documentation for *File core.h*](#)

```

/*
    nanogui/serializer/core.h -- helper class to serialize
    the full state of an application to a convenient binary format

    NanoGUI was developed by Wenzel Jakob <wenzel@inf.ethz.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a

```

```
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>
#include <unordered_map>
#include <fstream>
#include <memory>
#include <set>

#ifndef DOXYGEN_SHOULD_SKIP_THIS
namespace half_float { class half; }
#endif

NAMESPACE_BEGIN(nanogui)

NAMESPACE_BEGIN(detail)
template <typename T> struct serialization_helper;
NAMESPACE_END(detail)

class Serializer {
protected:
    // this friendship breaks the documentation
    #ifndef DOXYGEN_SHOULD_SKIP_THIS
        template <typename T> friend struct detail::serialization_helper;
    #endif

public:
    Serializer(const std::string &filename, bool write);

    ~Serializer();

    static bool isSerializedFile(const std::string &filename);

    size_t size();

    void push(const std::string &name);

    void pop();

    std::vector<std::string> keys() const;

    void setCompatibility(bool compatibility) { mCompatibility = compatibility; }

    bool compatibility() { return mCompatibility; }

    template <typename T> void set(const std::string &name, const T &value) {
        typedef detail::serialization_helper<T> helper;
        set_base(name, helper::type_id());
        if (!name.empty())
            push(name);
        helper::write(*this, &value, 1);
        if (!name.empty())
            pop();
    }

    template <typename T> bool get(const std::string &name, T &value) {
        typedef detail::serialization_helper<T> helper;
```

```

        if (!get_base(name, helper::type_id()))
            return false;
        if (!name.empty())
            push(name);
        helper::read(*this, &value, 1);
        if (!name.empty())
            pop();
        return true;
    }
protected:
    void set_base(const std::string &name, const std::string &type_id);
    bool get_base(const std::string &name, const std::string &type_id);

    void writeTOC();
    void readTOC();

    void read(void *p, size_t size);
    void write(const void *p, size_t size);
    void seek(size_t pos);
private:
    std::string mFilename;
    bool mWrite, mCompatibility;
    std::fstream mFile;
    std::unordered_map<std::string, std::pair<std::string, uint64_t>> mTOC;
    std::vector<std::string> mPrefixStack;
};

NAMESPACE_BEGIN(detail)

template <typename T, typename SFINAE = void> struct serialization_traits { };

// bypass template specializations for now
#ifdef DOXYGEN_SHOULD_SKIP_THIS
template <> struct serialization_traits<int8_t> { const char *type_id = "u8"; };
template <> struct serialization_traits<uint8_t> { const char *type_id = "s8"; };
template <> struct serialization_traits<int16_t> { const char *type_id = "u16"; };
template <> struct serialization_traits<uint16_t> { const char *type_id = "s16"; };
template <> struct serialization_traits<int32_t> { const char *type_id = "u32"; };
template <> struct serialization_traits<uint32_t> { const char *type_id = "s32"; };
template <> struct serialization_traits<int64_t> { const char *type_id = "u64"; };
template <> struct serialization_traits<uint64_t> { const char *type_id = "s64"; };
template <> struct serialization_traits<half_float::half> { const char *type_id = "f16"; };
template <> struct serialization_traits<float> { const char *type_id = "f32"; };
template <> struct serialization_traits<double> { const char *type_id = "f64"; };
template <> struct serialization_traits<bool> { const char *type_id = "b8"; };

```

```
template <> struct serialization_traits<char> { const char *type_id = "c8"; };

template <typename T> struct serialization_traits<T> :
    serialization_traits<typename std::underlying_type<T>::type,
        typename std::enable_if<std::is_enum<T>::value>::type> { };

template <typename T> struct serialization_helper {
    static std::string type_id() { return serialization_traits<T>().type_id; }

    static void write(Serializer &s, const T *value, size_t count) {
        s.write(value, sizeof(T) * count);
    }

    static void read(Serializer &s, T *value, size_t count) {
        s.read(value, sizeof(T) * count);
    }
};

template <> struct serialization_helper<std::string> {
    static std::string type_id() { return "Vc8"; }

    static void write(Serializer &s, const std::string *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            uint32_t length = (uint32_t) value->length();
            s.write(&length, sizeof(uint32_t));
            s.write((char *) value->data(), sizeof(char) * value->length());
            value++;
        }
    }

    static void read(Serializer &s, std::string *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            uint32_t length;
            s.read(&length, sizeof(uint32_t));
            value->resize(length);
            s.read((char *) value->data(), sizeof(char) * length);
            value++;
        }
    }
};

template <typename T1, typename T2> struct serialization_helper<std::pair<T1, T2>> {
    static std::string type_id() {
        return "P" +
            serialization_helper<T1>::type_id() +
            serialization_helper<T2>::type_id();
    }

    static void write(Serializer &s, const std::pair<T1, T1> *value, size_t count) {
        std::unique_ptr<T1> first (new T1[count]);
        std::unique_ptr<T2> second(new T2[count]);

        for (size_t i = 0; i < count; ++i) {
            first.get()[i] = value[i].first;
            second.get()[i] = value[i].second;
        }
    }
};
```



```

        serialization_helper<T1>::write(s, first.get(), count);
        serialization_helper<T2>::write(s, second.get(), count);
    }

    static void read(Serializer &s, std::pair<T1, T1> *value, size_t count) {
        std::unique_ptr<T1> first (new T1[count]);
        std::unique_ptr<T2> second(new T2[count]);

        serialization_helper<T1>::read(s, first.get(), count);
        serialization_helper<T2>::read(s, second.get(), count);

        for (size_t i = 0; i<count; ++i) {
            value[i].first = first.get()[i];
            value[i].second = second.get()[i];
        }
    }
};

template <typename T> struct serialization_helper<std::vector<T>> {
    static std::string type_id() {
        return "V" + serialization_helper<T>::type_id();
    }

    static void write(Serializer &s, const std::vector<T> *value, size_t count) {
        for (size_t i = 0; i<count; ++i) {
            uint32_t size = (uint32_t) value->size();
            s.write(&size, sizeof(uint32_t));
            serialization_helper<T>::write(s, value->data(), size);
            value++;
        }
    }

    static void read(Serializer &s, std::vector<T> *value, size_t count) {
        for (size_t i = 0; i<count; ++i) {
            uint32_t size = 0;
            s.read(&size, sizeof(uint32_t));
            value->resize(size);
            serialization_helper<T>::read(s, value->data(), size);
            value++;
        }
    }
};

template <typename T> struct serialization_helper<std::set<T>> {
    static std::string type_id() {
        return "S" + serialization_helper<T>::type_id();
    }

    static void write(Serializer &s, const std::set<T> *value, size_t count) {
        for (size_t i = 0; i<count; ++i) {
            std::vector<T> temp(value->size());
            uint32_t idx = 0;
            for (auto it = value->begin(); it != value->end(); ++it)
                temp[idx++] = *it;
            serialization_helper<std::vector<T>>::write(s, &temp, 1);
            value++;
        }
    }
};

```

```
static void read(Serializer &s, std::set<T> *value, size_t count) {
    for (size_t i = 0; i < count; ++i) {
        std::vector<T> temp;
        serialization_helper<std::vector<T>>::read(s, &temp, 1);
        value->clear();
        for (auto k: temp)
            value->insert(k);
        value++;
    }
}

};

template <typename Scalar, int Rows, int Cols, int Options, int MaxRows, int MaxCols>
struct serialization_helper<Eigen::Matrix<Scalar, Rows, Cols, Options, MaxRows,
↳MaxCols>> {
    typedef Eigen::Matrix<Scalar, Rows, Cols, Options, MaxRows, MaxCols> Matrix;

    static std::string type_id() {
        return "M" + serialization_helper<Scalar>::type_id();
    }

    static void write(Serializer &s, const Matrix *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            uint32_t rows = value->rows(), cols = value->cols();
            s.write(&rows, sizeof(uint32_t));
            s.write(&cols, sizeof(uint32_t));
            serialization_helper<Scalar>::write(s, value->data(), rows*cols);
            value++;
        }
    }

    static void read(Serializer &s, Matrix *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            uint32_t rows = 0, cols = 0;
            s.read(&rows, sizeof(uint32_t));
            s.read(&cols, sizeof(uint32_t));
            value->resize(rows, cols);
            serialization_helper<Scalar>::read(s, value->data(), rows*cols);
            value++;
        }
    }
};

template <> struct serialization_helper<nanogui::Color>
    : public serialization_helper<Eigen::Matrix<float, 4, 1>> { };

template <typename Scalar, int Options>
struct serialization_helper<Eigen::Quaternion<Scalar, Options>>
    : public serialization_helper<Eigen::Matrix<Scalar, 4, 1>> {
    typedef Eigen::Quaternion<Scalar, Options> Quat;

    static std::string type_id() {
        return "Q" + serialization_helper<Scalar>::type_id();
    }

    static void write(Serializer &s, const Quat *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
```

```

        serialization_helper<Scalar>::write(s, value->coeffs().data(), 4);
        value++;
    }
}

static void read(Serializer &s, Quat *value, size_t count) {
    for (size_t i = 0; i < count; ++i) {
        serialization_helper<Scalar>::read(s, value->coeffs().data(), 4);
        value++;
    }
}
};

template <>
struct serialization_helper<Widget> {
    static std::string type_id() {
        return "W";
    }

    static void write(Serializer &s, const Widget *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            if (!value->id().empty()) {
                if (count > 1)
                    s.push(value->id());
                value->save(s);
            }

            for (const Widget *child : value->children()) {
                if (child->id().empty())
                    write(s, child, 1);
                else
                    s.set(child->id(), *child);
            }

            if (!value->id().empty() && count > 1)
                s.pop();

            ++value;
        }
    }

    static void read(Serializer &s, Widget *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            if (!value->id().empty()) {
                if (count > 1)
                    s.push(value->id());
                value->load(s);
            }

            for (Widget *child : value->children()) {
                if (child->id().empty())
                    read(s, child, 1);
                else
                    s.get(child->id(), *child);
            }

            if (!value->id().empty() && count > 1)
                s.pop();
        }
    }
};

```

```
        ++value;
    }
}
};

#ifdef DOXYGEN_SKIP_THIS

NAMESPACE_END(detail)
NAMESPACE_END(nanogui)
```

Includes

- `fstream`
- `memory`
- `nanogui/widget.h` (*File widget.h*)
- `set`
- `unordered_map`

Included By

- *File opengl.h*
- *File sparse.h*

Namespaces

- *Namespace nanogui*
- *Namespace nanogui::detail*

Classes

- *Struct serialization_helper*
- *Struct serialization_traits*
- *Class Serializer*

File entypo.h

Definition (`include/nanogui/entypo.h`)

Program Listing for File entypo.h

- [Return to documentation for *File entypo.h*](#)

```

#pragma once

#ifndef DOXYGEN_SHOULD_SKIP_THIS

#define ENTYPQ_ICON_PHONE 0x1F4DE
#define ENTYPQ_ICON_MOBILE 0x1F4F1
#define ENTYPQ_ICON_MOUSE 0xE789
#define ENTYPQ_ICON_ADDRESS 0xE723
#define ENTYPQ_ICON_MAIL 0x2709
#define ENTYPQ_ICON_PAPER_PLANE 0x1F53F
#define ENTYPQ_ICON_PENCIL 0x270E
#define ENTYPQ_ICON_FEATHER 0x2712
#define ENTYPQ_ICON_ATTACH 0x1F4CE
#define ENTYPQ_ICON_INBOX 0xE777
#define ENTYPQ_ICON_REPLY 0xE712
#define ENTYPQ_ICON_REPLY_ALL 0xE713
#define ENTYPQ_ICON_FORWARD 0x27A6
#define ENTYPQ_ICON_USER 0x1F464
#define ENTYPQ_ICON_USERS 0x1F465
#define ENTYPQ_ICON_ADD_USER 0xE700
#define ENTYPQ_ICON_VCARD 0xE722
#define ENTYPQ_ICON_EXPORT 0xE715
#define ENTYPQ_ICON_LOCATION 0xE724
#define ENTYPQ_ICON_MAP 0xE727
#define ENTYPQ_ICON_COMPASS 0xE728
#define ENTYPQ_ICON_DIRECTION 0x27A2
#define ENTYPQ_ICON_HAIR_CROSS 0x1F3AF
#define ENTYPQ_ICON_SHARE 0xE73C
#define ENTYPQ_ICON_SHAREABLE 0xE73E
#define ENTYPQ_ICON_HEART 0x2665
#define ENTYPQ_ICON_HEART_EMPTY 0x2661
#define ENTYPQ_ICON_STAR 0x2605
#define ENTYPQ_ICON_STAR_EMPTY 0x2606
#define ENTYPQ_ICON_THUMBS_UP 0x1F44D
#define ENTYPQ_ICON_THUMBS_DOWN 0x1F44E
#define ENTYPQ_ICON_CHAT 0xE720
#define ENTYPQ_ICON_COMMENT 0xE718
#define ENTYPQ_ICON_QUOTE 0x275E
#define ENTYPQ_ICON_HOME 0x2302
#define ENTYPQ_ICON_POPUP 0xE74C
#define ENTYPQ_ICON_SEARCH 0x1F50D
#define ENTYPQ_ICON_FLASHLIGHT 0x1F526
#define ENTYPQ_ICON_PRINT 0xE716
#define ENTYPQ_ICON_BELL 0x1F514
#define ENTYPQ_ICON_LINK 0x1F517
#define ENTYPQ_ICON_FLAG 0x2691
#define ENTYPQ_ICON_COG 0x2699
#define ENTYPQ_ICON_TOOLS 0x2692
#define ENTYPQ_ICON_TROPHY 0x1F3C6
#define ENTYPQ_ICON_TAG 0xE70C
#define ENTYPQ_ICON_CAMERA 0x1F4F7
#define ENTYPQ_ICON_MEGAPHONE 0x1F4E3
#define ENTYPQ_ICON_MOON 0x263D
#define ENTYPQ_ICON_PALETTE 0x1F3A8
#define ENTYPQ_ICON_LEAF 0x1F342
#define ENTYPQ_ICON_NOTE 0x266A
#define ENTYPQ_ICON_BEAMED_NOTE 0x266B
#define ENTYPQ_ICON_NEW 0x1F4A5


```

```
#define ENTYPQ_ICON_GRADUATION_CAP      0x1F393
#define ENTYPQ_ICON_BOOK                 0x1F4D5
#define ENTYPQ_ICON_NEWSPAPER            0x1F4F0
#define ENTYPQ_ICON_BAG                  0x1F45C
#define ENTYPQ_ICON_AIRPLANE             0x2708
#define ENTYPQ_ICON_LIFEBUOY             0xE788
#define ENTYPQ_ICON_EYE                  0xE70A
#define ENTYPQ_ICON_CLOCK                 0x1F554
#define ENTYPQ_ICON_MIC                   0x1F3A4
#define ENTYPQ_ICON_CALENDAR              0x1F4C5
#define ENTYPQ_ICON_FLASH                 0x26A1
#define ENTYPQ_ICON_THUNDER_CLOUD         0x26C8
#define ENTYPQ_ICON_DROPLET               0x1F4A7
#define ENTYPQ_ICON_CD                    0x1F4BF
#define ENTYPQ_ICON_BRIEFCASE             0x1F4BC
#define ENTYPQ_ICON_AIR                   0x1F4A8
#define ENTYPQ_ICON_HOURLASS              0x23F3
#define ENTYPQ_ICON_GAUGE                 0x1F6C7
#define ENTYPQ_ICON_LANGUAGE              0x1F394
#define ENTYPQ_ICON_NETWORK               0xE776
#define ENTYPQ_ICON_KEY                   0x1F511
#define ENTYPQ_ICON_BATTERY                0x1F50B
#define ENTYPQ_ICON_BUCKET                0x1F4FE
#define ENTYPQ_ICON_MAGNET                0xE7A1
#define ENTYPQ_ICON_DRIVE                 0x1F4FD
#define ENTYPQ_ICON_CUP                   0x2615
#define ENTYPQ_ICON_ROCKET                0x1F680
#define ENTYPQ_ICON_BRUSH                 0xE79A
#define ENTYPQ_ICON_SUITCASE              0x1F6C6
#define ENTYPQ_ICON_TRAFFIC_CONE          0x1F6C8
#define ENTYPQ_ICON_GLOBE                 0x1F30E
#define ENTYPQ_ICON_KEYBOARD              0x2328
#define ENTYPQ_ICON_BROWSER               0xE74E
#define ENTYPQ_ICON_PUBLISH               0xE74D
#define ENTYPQ_ICON_PROGRESS_3            0xE76B
#define ENTYPQ_ICON_PROGRESS_2            0xE76A
#define ENTYPQ_ICON_PROGRESS_1            0xE769
#define ENTYPQ_ICON_PROGRESS_0            0xE768
#define ENTYPQ_ICON_LIGHT_DOWN            0x1F505
#define ENTYPQ_ICON_LIGHT_UP              0x1F506
#define ENTYPQ_ICON_ADJUST                 0x25D1
#define ENTYPQ_ICON_CODE                   0xE714
#define ENTYPQ_ICON_MONITOR                0x1F4BB
#define ENTYPQ_ICON_INFINITY              0x221E
#define ENTYPQ_ICON_LIGHT_BULB            0x1F4A1
#define ENTYPQ_ICON_CREDIT_CARD           0x1F4B3
#define ENTYPQ_ICON_DATABASE              0x1F4F8
#define ENTYPQ_ICON_VOICEMAIL             0x2707
#define ENTYPQ_ICON_CLIPBOARD             0x1F4CB
#define ENTYPQ_ICON_CART                   0xE73D
#define ENTYPQ_ICON_BOX                    0x1F4E6
#define ENTYPQ_ICON_TICKET                 0x1F3AB
#define ENTYPQ_ICON_RSS                    0xE73A
#define ENTYPQ_ICON_SIGNAL                 0x1F4F6
#define ENTYPQ_ICON_THERMOMETER           0x1F4FF
#define ENTYPQ_ICON_WATER                  0x1F4A6
#define ENTYPQ_ICON_SWEDEN                 0xF601
#define ENTYPQ_ICON_LINE_GRAPH            0x1F4C8
```

```

#define ENTYP0_ICON_PIE_CHART          0x25F4
#define ENTYP0_ICON_BAR_GRAPH          0x1F4CA
#define ENTYP0_ICON_AREA_GRAPH         0x1F53E
#define ENTYP0_ICON_LOCK               0x1F512
#define ENTYP0_ICON_LOCK_OPEN          0x1F513
#define ENTYP0_ICON_LOGOUT              0xE741
#define ENTYP0_ICON_LOGIN               0xE740
#define ENTYP0_ICON_CHECK               0x2713
#define ENTYP0_ICON_CROSS               0x274C
#define ENTYP0_ICON_SQUARED_MINUS      0x229F
#define ENTYP0_ICON_SQUARED_PLUS       0x229E
#define ENTYP0_ICON_SQUARED_CROSS      0x274E
#define ENTYP0_ICON_CIRCLED_MINUS      0x2296
#define ENTYP0_ICON_CIRCLED_PLUS       0x2295
#define ENTYP0_ICON_CIRCLED_CROSS      0x2716
#define ENTYP0_ICON_MINUS               0x2796
#define ENTYP0_ICON_PLUS                0x2795
#define ENTYP0_ICON_ERASE               0x232B
#define ENTYP0_ICON_BLOCK               0x1F6AB
#define ENTYP0_ICON_INFO                0x2139
#define ENTYP0_ICON_CIRCLED_INFO       0xE705
#define ENTYP0_ICON_HELP                0x2753
#define ENTYP0_ICON_CIRCLED_HELP       0xE704
#define ENTYP0_ICON_WARNING             0x26A0
#define ENTYP0_ICON_CYCLE               0x1F504
#define ENTYP0_ICON_CW                  0x27F3
#define ENTYP0_ICON_CCW                 0x27F2
#define ENTYP0_ICON_SHUFFLE             0x1F500
#define ENTYP0_ICON_BACK                0x1F519
#define ENTYP0_ICON_LEVEL_DOWN          0x21B3
#define ENTYP0_ICON_RETWEET             0xE717
#define ENTYP0_ICON_LOOP                0x1F501
#define ENTYP0_ICON_BACK_IN_TIME        0xE771
#define ENTYP0_ICON_LEVEL_UP            0x21B0
#define ENTYP0_ICON_SWITCH              0x21C6
#define ENTYP0_ICON_NUMBERED_LIST       0xE005
#define ENTYP0_ICON_ADD_TO_LIST         0xE003
#define ENTYP0_ICON_LAYOUT              0x268F
#define ENTYP0_ICON_LIST                0x2630
#define ENTYP0_ICON_TEXT_DOC            0x1F4C4
#define ENTYP0_ICON_TEXT_DOC_INVERTED  0xE731
#define ENTYP0_ICON_DOC                 0xE730
#define ENTYP0_ICON_DOCS                0xE736
#define ENTYP0_ICON_LANDSCAPE_DOC       0xE737
#define ENTYP0_ICON_PICTURE             0x1F304
#define ENTYP0_ICON_VIDEO               0x1F3AC
#define ENTYP0_ICON_MUSIC               0x1F3B5
#define ENTYP0_ICON_FOLDER              0x1F4C1
#define ENTYP0_ICON_ARCHIVE             0xE800
#define ENTYP0_ICON_TRASH               0xE729
#define ENTYP0_ICON_UPLOAD              0x1F4E4
#define ENTYP0_ICON_DOWNLOAD            0x1F4E5
#define ENTYP0_ICON_SAVE                0x1F4BE
#define ENTYP0_ICON_INSTALL              0xE778
#define ENTYP0_ICON_CLOUD               0x2601
#define ENTYP0_ICON_UPLOAD_CLOUD        0xE711
#define ENTYP0_ICON_BOOKMARK            0x1F516
#define ENTYP0_ICON_BOOKMARKS           0x1F4D1

```

```
#define ENTYPQ_ICON_OPEN_BOOK      0x1F4D6
#define ENTYPQ_ICON_PLAY           0x25B6
#define ENTYPQ_ICON_PAUS           0x2016
#define ENTYPQ_ICON_RECORD         0x25CF
#define ENTYPQ_ICON_STOP           0x25A0
#define ENTYPQ_ICON_FF              0x23E9
#define ENTYPQ_ICON_FB              0x23EA
#define ENTYPQ_ICON_TO_START        0x23EE
#define ENTYPQ_ICON_TO_END          0x23ED
#define ENTYPQ_ICON_RESIZE_FULL     0xE744
#define ENTYPQ_ICON_RESIZE_SMALL    0xE746
#define ENTYPQ_ICON_VOLUME          0x23F7
#define ENTYPQ_ICON_SOUND           0x1F50A
#define ENTYPQ_ICON_MUTE            0x1F507
#define ENTYPQ_ICON_FLOW_CASCADE   0x1F568
#define ENTYPQ_ICON_FLOW_BRANCH    0x1F569
#define ENTYPQ_ICON_FLOW_TREE      0x1F56A
#define ENTYPQ_ICON_FLOW_LINE      0x1F56B
#define ENTYPQ_ICON_FLOW_PARALLEL  0x1F56C
#define ENTYPQ_ICON_LEFT_BOLD      0xE4AD
#define ENTYPQ_ICON_DOWN_BOLD      0xE4B0
#define ENTYPQ_ICON_UP_BOLD        0xE4AF
#define ENTYPQ_ICON_RIGHT_BOLD     0xE4AE
#define ENTYPQ_ICON_LEFT           0x2B05
#define ENTYPQ_ICON_DOWN           0x2B07
#define ENTYPQ_ICON_UP             0x2B06
#define ENTYPQ_ICON_RIGHT          0x27A1
#define ENTYPQ_ICON_CIRCLED_LEFT   0xE759
#define ENTYPQ_ICON_CIRCLED_DOWN   0xE758
#define ENTYPQ_ICON_CIRCLED_UP     0xE75B
#define ENTYPQ_ICON_CIRCLED_RIGHT  0xE75A
#define ENTYPQ_ICON_TRIANGLE_LEFT  0x25C2
#define ENTYPQ_ICON_TRIANGLE_DOWN  0x25BE
#define ENTYPQ_ICON_TRIANGLE_UP    0x25B4
#define ENTYPQ_ICON_TRIANGLE_RIGHT 0x25B8
#define ENTYPQ_ICON_CHEVRON_LEFT   0xE75D
#define ENTYPQ_ICON_CHEVRON_DOWN   0xE75C
#define ENTYPQ_ICON_CHEVRON_UP     0xE75F
#define ENTYPQ_ICON_CHEVRON_RIGHT  0xE75E
#define ENTYPQ_ICON_CHEVRON_SMALL_LEFT 0xE761
#define ENTYPQ_ICON_CHEVRON_SMALL_DOWN 0xE760
#define ENTYPQ_ICON_CHEVRON_SMALL_UP 0xE763
#define ENTYPQ_ICON_CHEVRON_SMALL_RIGHT 0xE762
#define ENTYPQ_ICON_CHEVRON_THIN_LEFT 0xE765
#define ENTYPQ_ICON_CHEVRON_THIN_DOWN 0xE764
#define ENTYPQ_ICON_CHEVRON_THIN_UP 0xE767
#define ENTYPQ_ICON_CHEVRON_THIN_RIGHT 0xE766
#define ENTYPQ_ICON_LEFT_THIN      0x2190
#define ENTYPQ_ICON_DOWN_THIN      0x2193
#define ENTYPQ_ICON_UP_THIN        0x2191
#define ENTYPQ_ICON_RIGHT_THIN     0x2192
#define ENTYPQ_ICON_ARROW_COMBO    0xE74F
#define ENTYPQ_ICON_THREE_DOTS     0x23F6
#define ENTYPQ_ICON_TWO_DOTS       0x23F5
#define ENTYPQ_ICON_DOT             0x23F4
#define ENTYPQ_ICON_CC              0x1F545
#define ENTYPQ_ICON_CC_BY           0x1F546
#define ENTYPQ_ICON_CC_NC           0x1F547
```



```

#define ENTYPQ_ICON_CC_NC_EU          0x1F548
#define ENTYPQ_ICON_CC_NC_JP          0x1F549
#define ENTYPQ_ICON_CC_SA              0x1F54A
#define ENTYPQ_ICON_CC_ND              0x1F54B
#define ENTYPQ_ICON_CC_PD              0x1F54C
#define ENTYPQ_ICON_CC_ZERO            0x1F54D
#define ENTYPQ_ICON_CC_SHARE            0x1F54E
#define ENTYPQ_ICON_CC_REMIX            0x1F54F
#define ENTYPQ_ICON_DB_LOGO            0x1F5F9
#define ENTYPQ_ICON_DB_SHAPE            0x1F5FA
#define ENTYPQ_ICON_GITHUB              0xF300
#define ENTYPQ_ICON_C_GITHUB            0xF301
#define ENTYPQ_ICON_FLICKR              0xF303
#define ENTYPQ_ICON_C_FLICKR            0xF304
#define ENTYPQ_ICON_VIMEO              0xF306
#define ENTYPQ_ICON_C_VIMEO            0xF307
#define ENTYPQ_ICON_TWITTER              0xF309
#define ENTYPQ_ICON_C_TWITTER            0xF30A
#define ENTYPQ_ICON_FACEBOOK            0xF30C
#define ENTYPQ_ICON_C_FACEBOOK          0xF30D
#define ENTYPQ_ICON_S_FACEBOOK          0xF30E
#define ENTYPQ_ICON_GOOGLEPLUS          0xF30F
#define ENTYPQ_ICON_C_GOOGLEPLUS        0xF310
#define ENTYPQ_ICON_PINTEREST           0xF312
#define ENTYPQ_ICON_C_PINTEREST         0xF313
#define ENTYPQ_ICON_TUMBLR              0xF315
#define ENTYPQ_ICON_C_TUMBLR            0xF316
#define ENTYPQ_ICON_LINKEDIN            0xF318
#define ENTYPQ_ICON_C_LINKEDIN          0xF319
#define ENTYPQ_ICON_DRIBBBLE            0xF31B
#define ENTYPQ_ICON_C_DRIBBBLE          0xF31C
#define ENTYPQ_ICON_STUMBLEUPON         0xF31E
#define ENTYPQ_ICON_C_STUMBLEUPON       0xF31F
#define ENTYPQ_ICON_LASTFM              0xF321
#define ENTYPQ_ICON_C_LASTFM            0xF322
#define ENTYPQ_ICON_RDIO                0xF324
#define ENTYPQ_ICON_C_RDIO              0xF325
#define ENTYPQ_ICON_SPOTIFY             0xF327
#define ENTYPQ_ICON_C_SPOTIFY           0xF328
#define ENTYPQ_ICON_QQ                  0xF32A
#define ENTYPQ_ICON_INSTAGRAM           0xF32D
#define ENTYPQ_ICON_DROPBOX             0xF330
#define ENTYPQ_ICON_EVERNOTE            0xF333
#define ENTYPQ_ICON_FLATTR              0xF336
#define ENTYPQ_ICON_SKYPE               0xF339
#define ENTYPQ_ICON_C_SKYPE             0xF33A
#define ENTYPQ_ICON_RENREN              0xF33C
#define ENTYPQ_ICON_SINA_WEIBO          0xF33F
#define ENTYPQ_ICON_PAYPAL              0xF342
#define ENTYPQ_ICON_PICASA              0xF345
#define ENTYPQ_ICON_SOUNDCLOUD          0xF348
#define ENTYPQ_ICON_MIXI                0xF34B
#define ENTYPQ_ICON_BEHANCE             0xF34E
#define ENTYPQ_ICON_GOOGLE_CIRCLES      0xF351
#define ENTYPQ_ICON_VK                  0xF354
#define ENTYPQ_ICON_SMASHING            0xF357

#endif // DOXYGEN_SHOULD_SKIP_THIS

```

Included By

- *File nanogui.h*
- *File popupbutton.h*

File formhelper.h

Definition (include/nanogui/formhelper.h)

Program Listing for File formhelper.h

- [Return to documentation for *File formhelper.h*](#)

```
/*
    nanogui/formhelper.h -- helper class to construct forms for editing a set
    of variables of various types

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/screen.h>
#include <nanogui/label.h>
#include <nanogui/checkbox.h>
#include <nanogui/textbox.h>
#include <nanogui/combobox.h>
#include <nanogui/colorpicker.h>
#include <nanogui/layout.h>
#include <cassert>

NAMESPACE_BEGIN(nanogui)

NAMESPACE_BEGIN(detail)
template <typename T, typename sfinae = std::true_type> class FormWidget { };
NAMESPACE_END(detail)

class FormHelper {
public:
    FormHelper(Screen *screen) : mScreen(screen) { }

    Window *addWindow(const Vector2i &pos,
                      const std::string &title = "Untitled") {
        assert(mScreen);
        mWindow = new Window(mScreen, title);
        mLayout = new AdvancedGridLayout({10, 0, 10, 0}, {});
        mLayout->setMargin(10);
        mLayout->setColStretch(2, 1);
        mWindow->setPosition(pos);
        mWindow->setLayout(mLayout);
    }
};
```

```

        mWindow->setVisible(true);
        return mWindow;
    }

    Label *addGroup(const std::string &caption) {
        Label* label = new Label(mWindow, caption, mGroupFontName, mGroupFontSize);
        if (mLayout->rowCount() > 0)
            mLayout->appendRow(mPreGroupSpacing); /* Spacing */
        mLayout->appendRow(0);
        mLayout->setAnchor(label, AdvancedGridLayout::Anchor(0, mLayout->rowCount()-1,
↪ 4, 1));
        mLayout->appendRow(mPostGroupSpacing);
        return label;
    }

    template <typename Type> detail::FormWidget<Type> *
    addVariable(const std::string &label, const std::function<void(const Type &)> &
↪ setter,
                const std::function<Type()> &getter, bool editable = true) {
        Label *labelW = new Label(mWindow, label, mLabelFontName, mLabelFontSize);
        auto widget = new detail::FormWidget<Type>(mWindow);
        auto refresh = [widget, getter] {
            Type value = getter(), current = widget->value();
            if (value != current)
                widget->setValue(value);
        };
        refresh();
        widget->setCallback(setter);
        widget->setEditable(editable);
        widget->setFontSize(mWidgetFontSize);
        Vector2i fs = widget->fixedSize();
        widget->setFixedSize(Vector2i(fs.x() != 0 ? fs.x() : mFixedSize.x(),
                                     fs.y() != 0 ? fs.y() : mFixedSize.y()));
        mRefreshCallbacks.push_back(refresh);
        if (mLayout->rowCount() > 0)
            mLayout->appendRow(mVariableSpacing);
        mLayout->appendRow(0);
        mLayout->setAnchor(labelW, AdvancedGridLayout::Anchor(1, mLayout->rowCount()-
↪ 1));
        mLayout->setAnchor(widget, AdvancedGridLayout::Anchor(3, mLayout->rowCount()-
↪ 1));
        return widget;
    }

    template <typename Type> detail::FormWidget<Type> *
    addVariable(const std::string &label, Type &value, bool editable = true) {
        return addVariable<Type>(label,
            [&](const Type & v) { value = v; },
            [&]() -> Type { return value; },
            editable
        );
    }

    Button *addButton(const std::string &label, const std::function<void()> &cb) {
        Button *button = new Button(mWindow, label);
        button->setCallback(cb);
        button->setFixedHeight(25);
        if (mLayout->rowCount() > 0)

```

```

        mLayout->appendRow(mVariableSpacing);
        mLayout->appendRow(0);
        mLayout->setAnchor(button, AdvancedGridLayout::Anchor(1, mLayout->rowCount()-
↪1, 3, 1));
        return button;
    }

    void addWidget(const std::string &label, Widget *widget) {
        mLayout->appendRow(0);
        if (label == "") {
            mLayout->setAnchor(widget, AdvancedGridLayout::Anchor(1, mLayout->
↪rowCount()-1, 3, 1));
        } else {
            Label *labelW = new Label(mWindow, label, mLabelFontName, mLabelFontSize);
            mLayout->setAnchor(labelW, AdvancedGridLayout::Anchor(1, mLayout->
↪rowCount()-1));
            mLayout->setAnchor(widget, AdvancedGridLayout::Anchor(3, mLayout->
↪rowCount()-1));
        }
    }

    void refresh() {
        for (auto const &callback : mRefreshCallbacks)
            callback();
    }

    Window *window() { return mWindow; }
    void setWindow(Window *window) {
        mWindow = window;
        mLayout = dynamic_cast<AdvancedGridLayout *>(window->layout());
        if (mLayout == nullptr)
            throw std::runtime_error(
                "Internal error: window has an incompatible layout!");
    }

    void setFixedSize(const Vector2i &fw) { mFixedSize = fw; }
    Vector2i fixedSize() { return mFixedSize; }

    /* Set the font size / name of labels, group headers, and data widgets */
    const std::string &groupFontName() const { return mGroupFontName; }
    void setGroupFontName(const std::string &name) { mGroupFontName = name; }
    const std::string &labelFontName() const { return mLabelFontName; }
    void setLabelFontName(const std::string &name) { mLabelFontName = name; }
    int groupFontSize() const { return mGroupFontSize; }
    void setGroupFontSize(int value) { mGroupFontSize = value; }
    int labelFontSize() const { return mLabelFontSize; }
    void setLabelFontSize(int value) { mLabelFontSize = value; }
    int widgetFontSize() const { return mWidgetFontSize; }
    void setWidgetFontSize(int value) { mWidgetFontSize = value; }

protected:
    ref<Screen> mScreen;
    ref<Window> mWindow;
    ref<AdvancedGridLayout> mLayout;
    std::vector<std::function<void()>> mRefreshCallbacks;
    std::string mGroupFontName = "sans-bold";
    std::string mLabelFontName = "sans";
    Vector2i mFixedSize = Vector2i(0, 20);

```

```

    int mGroupFontSize = 20;
    int mLabelFontSize = 16;
    int mWidgetFontSize = 16;
    int mPreGroupSpacing = 15;
    int mPostGroupSpacing = 5;
    int mVariableSpacing = 5;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_BEGIN(detail)

// template specialization unsupported
#ifdef DOXYGEN_SHOULD_SKIP_THIS

/* Various types of form widgets for different input types below */
template <> class FormWidget<bool, std::true_type> : public CheckBox {
public:
    FormWidget(Widget *p) : CheckBox(p, "") { setFixedWidth(20); }
    void setValue(bool v) { setChecked(v); }
    void setEditable(bool e) { setEnabled(e); }
    bool value() const { return checked(); }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <typename T> class FormWidget<T, typename std::is_enum<T>::type> : public_
↳ComboBox {
public:
    FormWidget(Widget *p) : ComboBox(p) { }
    T value() const { return (T) selectedIndex(); }
    void setValue(T value) { setSelectedIndex((int) value); mSelectedIndex = (int)_
↳value; }
    void setCallback(const std::function<void(const T &)> &cb) {
        ComboBox::setCallback([cb](int v) { cb((const T &) v); });
    }
    void setEditable(bool e) { setEnabled(e); }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <typename T> class FormWidget<T, typename std::is_integral<T>::type> :_
↳public IntBox<T> {
public:
    FormWidget(Widget *p) : IntBox<T>(p) { this->
↳setAlignment(TextBox::Alignment::Right); }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <typename T> class FormWidget<T, typename std::is_floating_point<T>::type> :_
↳public FloatBox<T> {
public:
    FormWidget(Widget *p) : FloatBox<T>(p) { this->
↳setAlignment(TextBox::Alignment::Right); }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

```

```
template <> class FormWidget<std::string, std::true_type> : public TextBox {
public:
    FormWidget(Widget *p) : TextBox(p) { setAlignment(TextBox::Alignment::Left); }
    void setCallback(const std::function<void(const std::string&)> &cb) {
        TextBox::setCallback([cb](const std::string &str) { cb(str); return true; });
    }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <> class FormWidget<Color, std::true_type> : public ColorPicker {
public:
    FormWidget(Widget *p) : ColorPicker(p) { }
    void setValue(const Color &c) { setColor(c); }
    void setEditable(bool e) { setEnabled(e); }
    Color value() const { return color(); }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

#endif // DOXYGEN_SHOULD_SKIP_THIS

NAMESPACE_END(detail)
NAMESPACE_END(nanogui)
```

Includes

- `cassert`
- `nanogui/checkbox.h` (*File checkbox.h*)
- `nanogui/colorpicker.h` (*File colorpicker.h*)
- `nanogui/combobox.h` (*File combobox.h*)
- `nanogui/label.h` (*File label.h*)
- `nanogui/layout.h` (*File layout.h*)
- `nanogui/screen.h` (*File screen.h*)
- `nanogui/textbox.h` (*File textbox.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*
- *Namespace nanogui::detail*

Classes

- *Class FormWidget*
- *Class FormHelper*

File glcanvas.h

Definition (`include/nanogui/glcanvas.h`)

Program Listing for File glcanvas.h

- [Return to documentation for *File glcanvas.h*](#)

```

/*
    nanogui/glcanvas.h -- Canvas widget for rendering OpenGL content

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/

#pragma once

#include <iostream>

#include <nanogui/widget.h>
#include <nanogui/opengl.h>
#include <nanogui/glutil.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT GLCanvas : public Widget {
public:
    GLCanvas(Widget *parent);

    const Color &backgroundColor() const { return mBackgroundColor; }
    void setBackgroundColor(const Color &backgroundColor) { mBackgroundColor =
↪backgroundColor; }

    void setDrawBorder(const bool bDrawBorder) { mDrawBorder = bDrawBorder; }
    const bool &drawBorder() const { return mDrawBorder; }

    virtual void draw(NVGcontext *ctx) override;

    virtual void drawGL() {}

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;

protected:
    void drawWidgetBorder(NVGcontext* ctx) const;

```

```
protected:
    Color mBackgroundColor;
    bool mDrawBorder;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)
```

Includes

- `iostream`
- `nanogui/glutil.h` (*File glutil.h*)
- `nanogui/opengl.h` (*File opengl.h*)
- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class GLCanvas*

File glutil.h

Definition (`include/nanogui/glutil.h`)

Program Listing for File glutil.h

- [Return to documentation for *File glutil.h*](#)

```
/*
    nanogui/glutil.h -- Convenience classes for accessing OpenGL >= 3.x

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
```



```

#pragma once

#include <nanogui/opengl.h>
#include <Eigen/Geometry>
#include <map>

#ifdef DOXYGEN_SHOULD_SKIP_THIS
namespace half_float { class half; }
#endif

#ifdef GL_HALF_FLOAT
# define GL_HALF_FLOAT 0x140B
#endif

NAMESPACE_BEGIN(nanogui)

// bypass template specializations
#ifdef DOXYGEN_SHOULD_SKIP_THIS

NAMESPACE_BEGIN(detail)
template <typename T> struct type_traits;
template <> struct type_traits<uint32_t> { enum { type = GL_UNSIGNED_INT, integral = 1 }; };
template <> struct type_traits<int32_t> { enum { type = GL_INT, integral = 1 }; };
template <> struct type_traits<uint16_t> { enum { type = GL_UNSIGNED_SHORT, integral = 1 }; };
template <> struct type_traits<int16_t> { enum { type = GL_SHORT, integral = 1 }; };
template <> struct type_traits<uint8_t> { enum { type = GL_UNSIGNED_BYTE, integral = 1 }; };
template <> struct type_traits<int8_t> { enum { type = GL_BYTE, integral = 1 }; };
template <> struct type_traits<double> { enum { type = GL_DOUBLE, integral = 0 }; };
template <> struct type_traits<float> { enum { type = GL_FLOAT, integral = 0 }; };
template <> struct type_traits<half_float::half> { enum { type = GL_HALF_FLOAT, integral = 0 }; };
template <typename T> struct serialization_helper;
NAMESPACE_END(detail)

#endif // DOXYGEN_SHOULD_SKIP_THIS

using Eigen::Quaternionf;

class GLUniformBuffer;

// -----

class NANOGUI_EXPORT GLShader {
// this friendship breaks the documentation
#ifdef DOXYGEN_SHOULD_SKIP_THIS
    template <typename T> friend struct detail::serialization_helper;
#endif
public:
    GLShader()
        : mVertexShader(0), mFragmentShader(0), mGeometryShader(0),
          mProgramShader(0), mVertexArrayObject(0) { }

    bool init(const std::string &name, const std::string &vertex_str,
              const std::string &fragment_str,
              const std::string &geometry_str = "");

```

```
bool initFromFiles(const std::string &name,
                  const std::string &vertex_fname,
                  const std::string &fragment_fname,
                  const std::string &geometry_fname = "");

const std::string &name() const { return mName; }

void define(const std::string &key, const std::string &value) { mDefinitions[key]_
↳= value; }

void bind();

void free();

GLint attrib(const std::string &name, bool warn = true) const;

GLint uniform(const std::string &name, bool warn = true) const;

template <typename Matrix> void uploadAttrib(const std::string &name, const_
↳Matrix &M, int version = -1) {
    uint32_t compSize = sizeof(typename Matrix::Scalar);
    GLuint glType = (GLuint) detail::type_traits<typename Matrix::Scalar>::type;
    bool integral = (bool) detail::type_traits<typename Matrix::Scalar>::integral;

    uploadAttrib(name, (uint32_t) M.size(), (int) M.rows(), compSize,
                  glType, integral, M.data(), version);
}

template <typename Matrix> void downloadAttrib(const std::string &name, Matrix &
↳M) {
    uint32_t compSize = sizeof(typename Matrix::Scalar);
    GLuint glType = (GLuint) detail::type_traits<typename Matrix::Scalar>::type;

    auto it = mBufferObjects.find(name);
    if (it == mBufferObjects.end())
        throw std::runtime_error("downloadAttrib(" + mName + ", " + name + ") :_
↳buffer not found!");

    const Buffer &buf = it->second;
    M.resize(buf.dim, buf.size / buf.dim);

    downloadAttrib(name, M.size(), M.rows(), compSize, glType, M.data());
}

template <typename Matrix> void uploadIndices(const Matrix &M, int version = -1) {
    uploadAttrib("indices", M, version);
}

void invalidateAttribs();

void freeAttrib(const std::string &name);

bool hasAttrib(const std::string &name) const {
    auto it = mBufferObjects.find(name);
    if (it == mBufferObjects.end())
        return false;
    return true;
}
```

```

    }

    void shareAttrib(const GLShader &otherShader, const std::string &name, const_
↪std::string &as = "");

    int attribVersion(const std::string &name) const {
        auto it = mBufferObjects.find(name);
        if (it == mBufferObjects.end())
            return -1;
        return it->second.version;
    }

    void resetAttribVersion(const std::string &name) {
        auto it = mBufferObjects.find(name);
        if (it != mBufferObjects.end())
            it->second.version = -1;
    }

    void drawArray(int type, uint32_t offset, uint32_t count);

    void drawIndexed(int type, uint32_t offset, uint32_t count);

    template <typename T>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 4, 4> &mat, bool_
↪warn = true) {
        glUniformMatrix4fv(uniform(name, warn), 1, GL_FALSE, mat.template cast<float>
↪().data());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral ==_
↪1, int>::type = 0>
    void setUniform(const std::string &name, T value, bool warn = true) {
        glUniform1i(uniform(name, warn), (int) value);
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral ==_
↪0, int>::type = 0>
    void setUniform(const std::string &name, T value, bool warn = true) {
        glUniform1f(uniform(name, warn), (float) value);
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral ==_
↪1, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 2, 1> &v, bool_
↪warn = true) {
        glUniform2i(uniform(name, warn), (int) v.x(), (int) v.y());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral ==_
↪0, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 2, 1> &v, bool_
↪warn = true) {
        glUniform2f(uniform(name, warn), (float) v.x(), (float) v.y());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral ==_
↪1, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 3, 1> &v, bool_
↪warn = true) {

```

```

        glUniform3i(uniform(name, warn), (int) v.x(), (int) v.y(), (int) v.z());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral == 0, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 3, 1> &v, bool warn = true) {
        glUniform3f(uniform(name, warn), (float) v.x(), (float) v.y(), (float) v.z());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral == 1, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 4, 1> &v, bool warn = true) {
        glUniform4i(uniform(name, warn), (int) v.x(), (int) v.y(), (int) v.z(), (int) v.w());
    }

    template <typename T, typename std::enable_if<detail::type_traits<T>::integral == 0, int>::type = 0>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 4, 1> &v, bool warn = true) {
        glUniform4f(uniform(name, warn), (float) v.x(), (float) v.y(), (float) v.z(), (float) v.w());
    }

    void setUniform(const std::string &name, const GLUniformBuffer &buf, bool warn = true);

    size_t bufferSize() const {
        size_t size = 0;
        for (auto const &buf : mBufferObjects)
            size += buf.second.size;
        return size;
    }

public:
    /* Low-level API */
    void uploadAttrib(const std::string &name, size_t size, int dim,
                    uint32_t compSize, GLuint glType, bool integral,
                    const void *data, int version = -1);
    void downloadAttrib(const std::string &name, size_t size, int dim,
                    uint32_t compSize, GLuint glType, void *data);

protected:
    struct Buffer {
        GLuint id;
        GLuint glType;
        GLuint dim;
        GLuint compSize;
        GLuint size;
        int version;
    };
    std::string mName;
    GLuint mVertexShader;
    GLuint mFragmentShader;
    GLuint mGeometryShader;
    GLuint mProgramShader;

```

```

    GLuint mVertexArrayObject;
    std::map<std::string, Buffer> mBufferObjects;
    std::map<std::string, std::string> mDefinitions;
};

// -----

class NANOGUI_EXPORT GLUniformBuffer {
public:
    GLUniformBuffer() : mID(0), mBindingPoint(0) { }

    void init();

    void free();

    void bind(int index);

    void release();

    void update(const std::vector<uint8_t> &data);

    int getBindingPoint() const { return mBindingPoint; }
private:
    GLuint mID;
    int mBindingPoint;
};

// -----

class UniformBufferStd140 : public std::vector<uint8_t> {
public:
    using Parent = std::vector<uint8_t>;

    using Parent::push_back;

    template <typename T, typename std::enable_if<std::is_pod<T>::value, int>::type = 0>
    ↪ void push_back(T value) {
        uint8_t *tmp = (uint8_t*) &value;
        for (int i = 0; i < sizeof(T); i++)
            Parent::push_back(tmp[i]);
    }

    template <typename Derived, typename std::enable_if
    ↪ <Derived::IsVectorAtCompileTime, int>::type = 0>
    void push_back(const Eigen::MatrixBase<Derived> &value) {
        const int n = (int) value.size();
        int i;
        for (i = 0; i < n; ++i)
            push_back(value[i]);
        const int pad = n == 1 ? 1 : (n == 2 ? 2 : 4);
        while ((i++) % pad != 0)
            push_back((typename Derived::Scalar) 0);
    }

    template <typename Derived, typename std::enable_if<
    ↪ Derived::IsVectorAtCompileTime, int>::type = 0>
    void push_back(const Eigen::MatrixBase<Derived> &value, bool colMajor = true) {

```

```
const int n = (int) (colMajor ? value.rows() : value.cols());
const int m = (int) (colMajor ? value.cols() : value.rows());
const int pad = n == 1 ? 1 : (n == 2 ? 2 : 4);

for (int i = 0; i < m; ++i) {
    int j;
    for (j = 0; j < n; ++j)
        push_back(colMajor ? value(j, i) : value(i, j));
    while ((j++) % pad != 0)
        push_back((typename Derived::Scalar) 0);
}
}
};

// -----

class NANOGUI_EXPORT GLFramebuffer {
public:
    GLFramebuffer() : mFramebuffer(0), mDepth(0), mColor(0), mSamples(0) { }

    void init(const Vector2i &size, int nSamples);

    void free();

    void bind();

    void release();

    void blit();

    bool ready() { return mFramebuffer != 0; }

    int samples() const { return mSamples; }

    void downloadTGA(const std::string &filename);
protected:
    GLuint mFramebuffer, mDepth, mColor;
    Vector2i mSize;
    int mSamples;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

// -----

struct Arcball {
    Arcball(float speedFactor = 2.0f)
        : mActive(false), mLastPos(Vector2i::Zero()), mSize(Vector2i::Zero()),
          mQuat(Quaternionf::Identity()),
          mIncr(Quaternionf::Identity()),
          mSpeedFactor(speedFactor) { }

    Arcball(const Quaternionf &quat)
        : mActive(false), mLastPos(Vector2i::Zero()), mSize(Vector2i::Zero()),
          mQuat(quat),
          mIncr(Quaternionf::Identity()),
          mSpeedFactor(2.0f) { }
```

```

Quaternionf &state() { return mQuat; }

void setState(const Quaternionf &state) {
    mActive = false;
    mLastPos = Vector2i::Zero();
    mQuat = state;
    mIncr = Quaternionf::Identity();
}

void setSize(Vector2i size) { mSize = size; }
const Vector2i &size() const { return mSize; }
void setSpeedFactor(float speedFactor) { mSpeedFactor = speedFactor; }
float speedFactor() const { return mSpeedFactor; }
bool active() const { return mActive; }

void button(Vector2i pos, bool pressed) {
    mActive = pressed;
    mLastPos = pos;
    if (!mActive)
        mQuat = (mIncr * mQuat).normalized();
    mIncr = Quaternionf::Identity();
}

bool motion(Vector2i pos) {
    if (!mActive)
        return false;

    /* Based on the rotation controller form AntTweakBar */
    float invMinDim = 1.0f / mSize.minCoeff();
    float w = (float) mSize.x(), h = (float) mSize.y();

    float ox = (mSpeedFactor * (2*mLastPos.x() - w) + w) - w - 1.0f;
    float tx = (mSpeedFactor * (2*pos.x() - w) + w) - w - 1.0f;
    float oy = (mSpeedFactor * (h - 2*mLastPos.y()) + h) - h - 1.0f;
    float ty = (mSpeedFactor * (h - 2*pos.y()) + h) - h - 1.0f;

    ox *= invMinDim; oy *= invMinDim;
    tx *= invMinDim; ty *= invMinDim;

    Vector3f v0(ox, oy, 1.0f), v1(tx, ty, 1.0f);
    if (v0.squaredNorm() > 1e-4f && v1.squaredNorm() > 1e-4f) {
        v0.normalize(); v1.normalize();
        Vector3f axis = v0.cross(v1);
        float sa = std::sqrt(axis.dot(axis)),
              ca = v0.dot(v1),
              angle = std::atan2(sa, ca);
        if (tx*tx + ty*ty > 1.0f)
            angle *= 1.0f + 0.2f * (std::sqrt(tx*tx + ty*ty) - 1.0f);
        mIncr = Eigen::AngleAxisf(angle, axis.normalized());
        if (!std::isfinite(mIncr.norm()))
            mIncr = Quaternionf::Identity();
    }
    return true;
}

Matrix4f matrix() const {
    Matrix4f result2 = Matrix4f::Identity();
    result2.block<3,3>(0, 0) = (mIncr * mQuat).toRotationMatrix();
}

```

```
        return result2;
    }

protected:
    bool mActive;
    Vector2i mLastPos;
    Vector2i mSize;
    Quaternionf mQuat, mIncr;
    float mSpeedFactor;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

// -----

extern NANOGUI_EXPORT Vector3f project(const Vector3f &obj,
                                      const Matrix4f &model,
                                      const Matrix4f &proj,
                                      const Vector2i &viewportSize);

extern NANOGUI_EXPORT Vector3f unproject(const Vector3f &win,
                                         const Matrix4f &model,
                                         const Matrix4f &proj,
                                         const Vector2i &viewportSize);

extern NANOGUI_EXPORT Matrix4f lookAt(const Vector3f &origin,
                                       const Vector3f &target,
                                       const Vector3f &up);

extern NANOGUI_EXPORT Matrix4f ortho(float left, float right,
                                     float bottom, float top,
                                     float nearVal, float farVal);

extern NANOGUI_EXPORT Matrix4f frustum(float left, float right,
                                       float bottom, float top,
                                       float nearVal, float farVal);
extern NANOGUI_EXPORT Matrix4f scale(const Vector3f &v);

extern NANOGUI_EXPORT Matrix4f translate(const Vector3f &v);

NAMESPACE_END(nanogui)
```

Includes

- Eigen/Geometry
- map
- nanogui/opengl.h (*File opengl.h*)

Included By

- *File glcanvas.h*
- *File imageview.h*

- File *opengl.h*

Namespaces

- Namespace *nanogui*

Classes

- Struct *Arcball*
- Struct *GLShader::Buffer*
- Class *GLFramebuffer*
- Class *GLShader*
- Class *GLUniformBuffer*
- Class *UniformBufferStd140*

Functions

- Function *frustum*
- Function *lookAt*
- Function *ortho*
- Function *project*
- Function *scale*
- Function *translate*
- Function *unproject*

Defines

- Define *GL_HALF_FLOAT*

File graph.h

Definition (`include/nanogui/graph.h`)

Program Listing for File graph.h

- [Return to documentation for File graph.h](#)

```
/*  
    nanogui/graph.h -- Simple graph widget for showing a function plot  
  
    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.  
    The widget drawing code is based on the NanoVG demo application  
    by Mikko Mononen.
```

```
All rights reserved. Use of this source code is governed by a
BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Graph : public Widget {
public:
    Graph(Widget *parent, const std::string &caption = "Untitled");

    const std::string &caption() const { return mCaption; }
    void setCaption(const std::string &caption) { mCaption = caption; }

    const std::string &header() const { return mHeader; }
    void setHeader(const std::string &header) { mHeader = header; }

    const std::string &footer() const { return mFooter; }
    void setFooter(const std::string &footer) { mFooter = footer; }

    const Color &backgroundcolor() const { return mBackgroundColor; }
    void setBackgroundColor(const Color &backgroundcolor) { mBackgroundColor = _
↳backgroundcolor; }

    const Color &foregroundcolor() const { return mForegroundColor; }
    void setForegroundColor(const Color &foregroundcolor) { mForegroundColor = _
↳foregroundcolor; }

    const Color &textcolor() const { return mTextColor; }
    void setTextColor(const Color &textcolor) { mTextColor = textcolor; }

    const VectorXf &values() const { return mValues; }
    VectorXf &values() { return mValues; }
    void setValues(const VectorXf &values) { mValues = values; }

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void draw(NVGcontext *ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    std::string mCaption, mHeader, mFooter;
    Color mBackgroundColor, mForegroundColor, mTextColor;
    VectorXf mValues;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Graph*

File imagepanel.h

Definition (`include/nanogui/imagepanel.h`)

Program Listing for File imagepanel.h

- [Return to documentation for *File imagepanel.h*](#)

```

/*
   nanogui/imagepanel.h -- Image panel widget which shows a number of
   square-shaped icons

   NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
   The widget drawing code is based on the NanoVG demo application
   by Mikko Mononen.

   All rights reserved. Use of this source code is governed by a
   BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT ImagePanel : public Widget {
public:
    typedef std::vector<std::pair<int, std::string>> Images;
public:
    ImagePanel(Widget *parent);

    void setImages(const Images &data) { mImages = data; }
    const Images& images() const { return mImages; }

```

```
std::function<void(int)> callback() const { return mCallback; }
void setCallback(const std::function<void(int)> &callback) { mCallback = callback;
↪ }

virtual bool mouseMotionEvent(const Vector2i &p, const Vector2i &rel, int button, ↪
↪int modifiers) override;
virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int ↪
↪modifiers) override;
virtual Vector2i preferredSize(NVGcontext *ctx) const override;
virtual void draw(NVGcontext* ctx) override;
protected:
    Vector2i gridSize() const;
    int indexOfPosition(const Vector2i &p) const;
protected:
    Images mImages;
    std::function<void(int)> mCallback;
    int mThumbSize;
    int mSpacing;
    int mMargin;
    int mMouseIndex;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ImagePanel*

File imageview.h

Definition (`include/nanogui/imageview.h`)

Program Listing for File imageview.h

- [Return to documentation for *File imageview.h*](#)

```

/*
    nanogui/imageview.h -- Widget used to display images.

    The image view widget was contributed by Stefan Ivanov.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>
#include <nanogui/glutil.h>
#include <functional>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT ImageView : public Widget {
public:
    ImageView(Widget* parent, GLuint imageID);
    ~ImageView();

    void bindImage(GLuint imageId);

    GLShader& imageShader() { return mShader; }

    Vector2f positionF() const { return mPos.cast<float>(); }
    Vector2f sizeF() const { return mSize.cast<float>(); }

    const Vector2i& imageSize() const { return mImageSize; }
    Vector2i scaledImageSize() const { return (mScale * mImageSize.cast<float>()).cast
↪<int>(); }
    Vector2f imageSizeF() const { return mImageSize.cast<float>(); }
    Vector2f scaledImageSizeF() const { return (mScale * mImageSize.cast<float>()); }

    const Vector2f& offset() const { return mOffset; }
    void setOffset(const Vector2f& offset) { mOffset = offset; }
    float scale() const { return mScale; }
    void setScale(float scale) { mScale = scale > 0.01f ? scale : 0.01f; }

    bool fixedOffset() const { return mFixedOffset; }
    void setFixedOffset(bool fixedOffset) { mFixedOffset = fixedOffset; }
    bool fixedScale() const { return mFixedScale; }
    void setFixedScale(bool fixedScale) { mFixedScale = fixedScale; }

    float zoomSensitivity() const { return mZoomSensitivity; }
    void setZoomSensitivity(float zoomSensitivity) { mZoomSensitivity =
↪zoomSensitivity; }

    float gridThreshold() const { return mGridThreshold; }
    void setGridThreshold(float gridThreshold) { mGridThreshold = gridThreshold; }

    float pixelInfoThreshold() const { return mPixelInfoThreshold; }
    void setPixelInfoThreshold(float pixelInfoThreshold) { mPixelInfoThreshold =
↪pixelInfoThreshold; }

```

```
#ifndef DOXYGEN_SHOULD_SKIP_THIS
    void setPixelInfoCallback(const std::function<std::pair<std::string, Color>(const_
↳Vector2i&)>& callback) {
        mPixelInfoCallback = callback;
    }
    const std::function<std::pair<std::string, Color>(const Vector2i&)>&_
↳pixelInfoCallback() const {
        return mPixelInfoCallback;
    }
#endif // DOXYGEN_SHOULD_SKIP_THIS

    void setFontScaleFactor(float fontScaleFactor) { mFontScaleFactor =_
↳fontScaleFactor; }
    float fontScaleFactor() const { return mFontScaleFactor; }

    // Image transformation functions.

    Vector2f imageCoordinateAt(const Vector2f& position) const;

    Vector2f clampedImageCoordinateAt(const Vector2f& position) const;

    Vector2f positionForCoordinate(const Vector2f& imageCoordinate) const;

    void setImageCoordinateAt(const Vector2f& position, const Vector2f&_
↳imageCoordinate);

    void center();

    void fit();

    void setScaleCentered(float scale);

    void moveOffset(const Vector2f& delta);

    void zoom(int amount, const Vector2f& focusPosition);

    bool keyboardEvent(int key, int scancode, int action, int modifiers) override;
    bool keyboardCharacterEvent(unsigned int codepoint) override;
    bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button, int_
↳modifiers) override;
    bool scrollEvent(const Vector2i &p, const Vector2f &rel) override;

    bool gridVisible() const;

    bool pixelInfoVisible() const;

    bool helpersVisible() const;

    Vector2i preferredSize(NVGcontext* ctx) const override;
    void performLayout(NVGcontext* ctx) override;
    void draw(NVGcontext* ctx) override;

private:
    // Helper image methods.
    void updateImageParameters();

    // Helper drawing methods.
```

```

void drawWidgetBorder(NVGcontext* ctx) const;
void drawImageBorder(NVGcontext* ctx) const;
void drawHelpers(NVGcontext* ctx) const;
static void drawPixelGrid(NVGcontext* ctx, const Vector2f& upperLeftCorner,
                           const Vector2f& lowerRightCorner, float stride);
void drawPixelInfo(NVGcontext* ctx, float stride) const;
void writePixelInfo(NVGcontext* ctx, const Vector2f& cellPosition,
                   const Vector2i& pixel, float stride, float fontSize) const;

// Image parameters.
GLShader mShader;
GLuint mImageID;
Vector2i mImageSize;

// Image display parameters.
float mScale;
Vector2f mOffset;
bool mFixedScale;
bool mFixedOffset;

// Fine-tuning parameters.
float mZoomSensitivity = 1.1f;

// Image info parameters.
float mGridThreshold = -1;
float mPixelInfoThreshold = -1;

// Image pixel data display members.
std::function<std::pair<std::string, Color>(const Vector2i&)> mPixelInfoCallback;
float mFontScaleFactor = 0.2f;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- functional
- nanogui/glutil.h (*File glutil.h*)
- nanogui/widget.h (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ImageView*

File label.h

Definition (include/nanogui/label.h)

Program Listing for File label.h

- [Return to documentation for *File label.h*](#)

```
/*
    nanogui/label.h -- Text label with an arbitrary font, color, and size

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Label : public Widget {
public:
    Label(Widget *parent, const std::string &caption,
          const std::string &font = "sans", int fontSize = -1);

    const std::string &caption() const { return mCaption; }
    void setCaption(const std::string &caption) { mCaption = caption; }

    void setFont(const std::string &font) { mFont = font; }
    const std::string &font() const { return mFont; }

    Color color() const { return mColor; }
    void setColor(const Color& color) { mColor = color; }

    virtual void setTheme(Theme *theme) override;

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;

    virtual void draw(NVGcontext *ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    std::string mCaption;
    std::string mFont;
    Color mColor;
public:
```



```

    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)

```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Label*

File layout.h

Definition (`include/nanogui/layout.h`)

Program Listing for File layout.h

- [Return to documentation for *File layout.h*](#)

```

/*
    nanogui/layout.h -- A collection of useful layout managers

    The grid layout was contributed by Christian Schueller.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/compat.h>
#include <nanogui/object.h>
#include <unordered_map>

NAMESPACE_BEGIN (nanogui)

```

```
enum class Alignment : uint8_t {
    Minimum = 0,
    Middle,
    Maximum,
    Fill
};

enum class Orientation {
    Horizontal = 0,
    Vertical
};

class NANOGUI_EXPORT Layout : public Object {
public:
    virtual void performLayout(NVGcontext *ctx, Widget *widget) const = 0;
    virtual Vector2i preferredSize(NVGcontext *ctx, const Widget *widget) const = 0;
protected:
    virtual ~Layout() { }
};

class NANOGUI_EXPORT BoxLayout : public Layout {
public:
    BoxLayout(Orientation orientation, Alignment alignment = Alignment::Middle,
              int margin = 0, int spacing = 0);

    Orientation orientation() const { return mOrientation; }
    void setOrientation(Orientation orientation) { mOrientation = orientation; }

    Alignment alignment() const { return mAlignment; }
    void setAlignment(Alignment alignment) { mAlignment = alignment; }

    int margin() const { return mMargin; }
    void setMargin(int margin) { mMargin = margin; }

    int spacing() const { return mSpacing; }
    void setSpacing(int spacing) { mSpacing = spacing; }

    /* Implementation of the layout interface */
    virtual Vector2i preferredSize(NVGcontext *ctx, const Widget *widget) const_
↳ override;
    virtual void performLayout(NVGcontext *ctx, Widget *widget) const override;

protected:
    Orientation mOrientation;
    Alignment mAlignment;
    int mMargin;
    int mSpacing;
};

class NANOGUI_EXPORT GroupLayout : public Layout {
public:
    GroupLayout(int margin = 15, int spacing = 6, int groupSpacing = 14,
                int groupIndent = 20)
        : mMargin(margin), mSpacing(spacing), mGroupSpacing(groupSpacing),
          mGroupIndent(groupIndent) {}

    int margin() const { return mMargin; }
```

```

    void setMargin(int margin) { mMargin = margin; }

    int spacing() const { return mSpacing; }
    void setSpacing(int spacing) { mSpacing = spacing; }

    int groupIndent() const { return mGroupIndent; }
    void setGroupIndent(int groupIndent) { mGroupIndent = groupIndent; }

    int groupSpacing() const { return mGroupSpacing; }
    void setGroupSpacing(int groupSpacing) { mGroupSpacing = groupSpacing; }

    /* Implementation of the layout interface */
    virtual Vector2i preferredSize(NVGcontext *ctx, const Widget *widget) const override;
    virtual void performLayout(NVGcontext *ctx, Widget *widget) const override;

protected:
    int mMargin;
    int mSpacing;
    int mGroupSpacing;
    int mGroupIndent;
};

class NANOGUI_EXPORT GridLayout : public Layout {
public:
    GridLayout(Orientation orientation = Orientation::Horizontal, int resolution = 2,
               Alignment alignment = Alignment::Middle,
               int margin = 0, int spacing = 0)
        : mOrientation(orientation), mResolution(resolution), mMargin(margin) {
        mDefaultAlignment[0] = mDefaultAlignment[1] = alignment;
        mSpacing = Vector2i::Constant(spacing);
    }

    Orientation orientation() const { return mOrientation; }
    void setOrientation(Orientation orientation) {
        mOrientation = orientation;
    }

    int resolution() const { return mResolution; }
    void setResolution(int resolution) { mResolution = resolution; }

    int spacing(int axis) const { return mSpacing[axis]; }
    void setSpacing(int axis, int spacing) { mSpacing[axis] = spacing; }
    void setSpacing(int spacing) { mSpacing[0] = mSpacing[1] = spacing; }

    int margin() const { return mMargin; }
    void setMargin(int margin) { mMargin = margin; }

    Alignment alignment(int axis, int item) const {
        if (item < (int) mAlignment[axis].size())
            return mAlignment[axis][item];
        else
            return mDefaultAlignment[axis];
    }

    void setColAlignment(Alignment value) { mDefaultAlignment[0] = value; }
    void setRowAlignment(Alignment value) { mDefaultAlignment[1] = value; }
    void setColAlignment(const std::vector<Alignment> &value) { mAlignment[0] = value; }
}

```

```

    void setRowAlignment(const std::vector<Alignment> &value) { mAlignment[1] = value;
↪ }

    /* Implementation of the layout interface */
    virtual Vector2i preferredSize(NVGcontext *ctx, const Widget *widget) const_
↪ override;
    virtual void performLayout(NVGcontext *ctx, Widget *widget) const override;

protected:
    // Compute the maximum row and column sizes
    void computeLayout(NVGcontext *ctx, const Widget *widget,
                      std::vector<int> *grid) const;

protected:
    Orientation mOrientation;
    Alignment mDefaultAlignment[2];
    std::vector<Alignment> mAlignment[2];
    int mResolution;
    Vector2i mSpacing;
    int mMargin;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

class NANOGUI_EXPORT AdvancedGridLayout : public Layout {
public:
    struct Anchor {
        uint8_t pos[2];
        uint8_t size[2];
        Alignment align[2];

        Anchor() { }

        Anchor(int x, int y, Alignment horiz = Alignment::Fill,
              Alignment vert = Alignment::Fill) {
            pos[0] = (uint8_t) x; pos[1] = (uint8_t) y;
            size[0] = size[1] = 1;
            align[0] = horiz; align[1] = vert;
        }

        Anchor(int x, int y, int w, int h,
              Alignment horiz = Alignment::Fill,
              Alignment vert = Alignment::Fill) {
            pos[0] = (uint8_t) x; pos[1] = (uint8_t) y;
            size[0] = (uint8_t) w; size[1] = (uint8_t) h;
            align[0] = horiz; align[1] = vert;
        }

        operator std::string() const {
            char buf[50];
            NANOGUI_SNPRINTF(buf, 50, "Format[pos=(%i, %i), size=(%i, %i), align=(%i,
↪ %i)]",
                            pos[0], pos[1], size[0], size[1], (int) align[0], (int) align[1]);
            return buf;
        }
    };

    AdvancedGridLayout(const std::vector<int> &cols = {}, const std::vector<int> &
↪ rows = {}, int margin = 0);

```

```

    int margin() const { return mMargin; }
    void setMargin(int margin) { mMargin = margin; }

    int colCount() const { return (int) mCols.size(); }

    int rowCount() const { return (int) mRows.size(); }

    void appendRow(int size, float stretch = 0.f) { mRows.push_back(size);
↳mRowStretch.push_back(stretch); };

    void appendCol(int size, float stretch = 0.f) { mCols.push_back(size);
↳mColStretch.push_back(stretch); };

    void setRowStretch(int index, float stretch) { mRowStretch.at(index) = stretch; }

    void setColStretch(int index, float stretch) { mColStretch.at(index) = stretch; }

    void setAnchor(const Widget *widget, const Anchor &anchor) { mAnchor[widget] =
↳anchor; }

    Anchor anchor(const Widget *widget) const {
        auto it = mAnchor.find(widget);
        if (it == mAnchor.end())
            throw std::runtime_error("Widget was not registered with the grid layout!
↳");
        return it->second;
    }

    /* Implementation of the layout interface */
    virtual Vector2i preferredSize(NVGcontext *ctx, const Widget *widget) const
↳override;
    virtual void performLayout(NVGcontext *ctx, Widget *widget) const override;

protected:
    void computeLayout(NVGcontext *ctx, const Widget *widget,
                      std::vector<int> *grid) const;

protected:
    std::vector<int> mCols, mRows;
    std::vector<float> mColStretch, mRowStretch;
    std::unordered_map<const Widget *, Anchor> mAnchor;
    int mMargin;
};

NAMESPACE_END(nanogui)

```

Includes

- nanogui/compat.h (*File compat.h*)
- nanogui/object.h (*File object.h*)
- unordered_map

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Struct AdvancedGridLayout::Anchor*
- *Class AdvancedGridLayout*
- *Class BoxLayout*
- *Class GridLayout*
- *Class GroupLayout*
- *Class Layout*

Enums

- *Enum Alignment*
- *Enum Orientation*

File messagedialog.h

Definition (include/nanogui/messagedialog.h)

Program Listing for File messagedialog.h

- [Return to documentation for *File messagedialog.h*](#)

```
/*  
    nanogui/messagedialog.h -- Simple "OK" or "Yes/No"-style modal dialogs  
  
    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.  
    The widget drawing code is based on the NanoVG demo application  
    by Mikko Mononen.  
  
    All rights reserved. Use of this source code is governed by a  
    BSD-style license that can be found in the LICENSE.txt file.  
*/  
#pragma once  
  
#include <nanogui/window.h>  
  
NAMESPACE_BEGIN(nanogui)
```

```

class NANOGUI_EXPORT MessageDialog : public Window {
public:
    enum class Type {
        Information,
        Question,
        Warning
    };

    MessageDialog(Widget *parent, Type type, const std::string &title = "Untitled",
                  const std::string &message = "Message",
                  const std::string &buttonText = "OK",
                  const std::string &altButtonText = "Cancel", bool altButton =
↪false);

    Label *messageLabel() { return mMessageLabel; }
    const Label *messageLabel() const { return mMessageLabel; }

    std::function<void(int)> callback() const { return mCallback; }
    void setCallback(const std::function<void(int)> &callback) { mCallback = callback;
↪ }
protected:
    std::function<void(int)> mCallback;
    Label *mMessageLabel;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- nanogui/window.h (*File window.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class MessageDialog*

Enums

- *Enum MessageDialog::Type*

File nanogui.h

Definition (`include/nanogui/nanogui.h`)

Program Listing for File nanogui.h

- [Return to documentation for *File nanogui.h*](#)

```
/*
    nanogui/nanogui.h -- Pull in *everything* from NanoGUI

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/

#pragma once

#include <nanogui/common.h>
#include <nanogui/widget.h>
#include <nanogui/screen.h>
#include <nanogui/theme.h>
#include <nanogui/window.h>
#include <nanogui/layout.h>
#include <nanogui/label.h>
#include <nanogui/checkbox.h>
#include <nanogui/button.h>
#include <nanogui/toolbutton.h>
#include <nanogui/popup.h>
#include <nanogui/popupbutton.h>
#include <nanogui/combobox.h>
#include <nanogui/progressbar.h>
#include <nanogui/entypo.h>
#include <nanogui/messagedialog.h>
#include <nanogui/textbox.h>
#include <nanogui/slider.h>
#include <nanogui/imagepanel.h>
#include <nanogui/imageview.h>
#include <nanogui/vscrollpanel.h>
#include <nanogui/colorwheel.h>
#include <nanogui/graph.h>
#include <nanogui/formhelper.h>
#include <nanogui/stackedwidget.h>
#include <nanogui/tabheader.h>
#include <nanogui/tabwidget.h>
#include <nanogui/glcanvas.h>
```

Includes

- `nanogui/button.h` (*File button.h*)
- `nanogui/checkbox.h` (*File checkbox.h*)

- `nanogui/colorwheel.h` (*File colorwheel.h*)
- `nanogui/combobox.h` (*File combobox.h*)
- `nanogui/common.h` (*File common.h*)
- `nanogui/entypo.h` (*File entypo.h*)
- `nanogui/formhelper.h` (*File formhelper.h*)
- `nanogui/glcanvas.h` (*File glcanvas.h*)
- `nanogui/graph.h` (*File graph.h*)
- `nanogui/imagepanel.h` (*File imagepanel.h*)
- `nanogui/imageview.h` (*File imageview.h*)
- `nanogui/label.h` (*File label.h*)
- `nanogui/layout.h` (*File layout.h*)
- `nanogui/messagedialog.h` (*File messagedialog.h*)
- `nanogui/popup.h` (*File popup.h*)
- `nanogui/popupbutton.h` (*File popupbutton.h*)
- `nanogui/progressbar.h` (*File progressbar.h*)
- `nanogui/screen.h` (*File screen.h*)
- `nanogui/slider.h` (*File slider.h*)
- `nanogui/stackedwidget.h` (*File stackedwidget.h*)
- `nanogui/tabheader.h` (*File tabheader.h*)
- `nanogui/tabwidget.h` (*File tabwidget.h*)
- `nanogui/textbox.h` (*File textbox.h*)
- `nanogui/theme.h` (*File theme.h*)
- `nanogui/toolbutton.h` (*File toolbutton.h*)
- `nanogui/vscrollpanel.h` (*File vscrollpanel.h*)
- `nanogui/widget.h` (*File widget.h*)
- `nanogui/window.h` (*File window.h*)

File `object.h`

Definition (`include/nanogui/object.h`)

Program Listing for File `object.h`

- Return to documentation for *File object.h*

```
/*
    nanogui/object.h -- Object base class with support for reference counting

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
```

```
by Mikko Mononen.

All rights reserved. Use of this source code is governed by a
BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/common.h>
#include <atomic>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Object {
public:
    Object() { }

    Object(const Object &) : m_refCount(0) {}

    int getRefCount() const { return m_refCount; };

    void incRef() const { ++m_refCount; }

    void decRef(bool dealloc = true) const noexcept;
protected:
    virtual ~Object();
private:
    mutable std::atomic<int> m_refCount { 0 };
};

template <typename T> class ref {
public:
    ref() { }

    ref(T *ptr) : m_ptr(ptr) {
        if (m_ptr)
            ((Object *) m_ptr)->incRef();
    }

    ref(const ref &r) : m_ptr(r.m_ptr) {
        if (m_ptr)
            ((Object *) m_ptr)->incRef();
    }

    ref(ref &&r) noexcept : m_ptr(r.m_ptr) {
        r.m_ptr = nullptr;
    }

    ~ref() {
        if (m_ptr)
            ((Object *) m_ptr)->decRef();
    }

    ref& operator=(ref&& r) noexcept {
        if (&r != this) {
            if (m_ptr)
                ((Object *) m_ptr)->decRef();
            m_ptr = r.m_ptr;
        }
    }
};
```

```

        r.m_ptr = nullptr;
    }
    return *this;
}

ref& operator=(const ref& r) noexcept {
    if (m_ptr != r.m_ptr) {
        if (r.m_ptr)
            ((Object *) r.m_ptr)->incRef();
        if (m_ptr)
            ((Object *) m_ptr)->decRef();
        m_ptr = r.m_ptr;
    }
    return *this;
}

ref& operator=(T *ptr) noexcept {
    if (m_ptr != ptr) {
        if (ptr)
            ((Object *) ptr)->incRef();
        if (m_ptr)
            ((Object *) m_ptr)->decRef();
        m_ptr = ptr;
    }
    return *this;
}

bool operator==(const ref &r) const { return m_ptr == r.m_ptr; }

bool operator!=(const ref &r) const { return m_ptr != r.m_ptr; }

bool operator==(const T* ptr) const { return m_ptr == ptr; }

bool operator!=(const T* ptr) const { return m_ptr != ptr; }

T* operator->() { return m_ptr; }

const T* operator->() const { return m_ptr; }

T& operator*() { return *m_ptr; }

const T& operator*() const { return *m_ptr; }

operator T* () { return m_ptr; }

T* get() { return m_ptr; }

const T* get() const { return m_ptr; }

operator bool() const { return m_ptr != nullptr; }
private:
    T *m_ptr = nullptr;
};

NAMESPACE_END(nanogui)

```

Includes

- `atomic`
- `nanogui/common.h` (*File common.h*)

Included By

- *File layout.h*
- *File theme.h*
- *File widget.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Object*
- *Class ref*

File `opengl.h`

Definition (`include/nanogui/serializer/opengl.h`)

Program Listing for File `opengl.h`

- [Return to documentation for *File opengl.h*](#)

```
/*
    nanogui/serializer/opengl.h -- serialization support for OpenGL buffers

    NanoGUI was developed by Wenzel Jakob <wenzel@inf.ethz.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/serializer/core.h>
#include <nanogui/glutil.h>
#include <set>

NAMESPACE_BEGIN(nanogui)
NAMESPACE_BEGIN(detail)

// bypass template specializations
#ifndef DOXYGEN_SHOULD_SKIP_THIS
```

```

template<>
struct serialization_helper<GLShader> {
    static std::string type_id() {
        return "G";
    }

    static void write(Serializer &s, const GLShader *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            if (count > 1)
                s.push(value->name());
            for (auto &item : value->mBufferObjects) {
                const GLShader::Buffer &buf = item.second;
                size_t totalSize = (size_t) buf.size * (size_t) buf.compSize;
                s.push(item.first);
                s.set("glType", buf.glType);
                s.set("compSize", buf.compSize);
                s.set("dim", buf.dim);
                s.set("size", buf.size);
                s.set("version", buf.version);
                Eigen::Matrix<uint8_t, Eigen::Dynamic, Eigen::Dynamic> temp(1,
↪totalSize);

                if (item.first == "indices") {
                    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buf.id);
                    glGetBufferSubData(GL_ELEMENT_ARRAY_BUFFER, 0, totalSize,
                                     temp.data());
                } else {
                    glBindBuffer(GL_ARRAY_BUFFER, buf.id);
                    glGetBufferSubData(GL_ARRAY_BUFFER, 0, totalSize, temp.data());
                }
                s.set("data", temp);
                s.pop();
            }
            if (count > 1)
                s.pop();
            ++value;
        }
    }

    static void read(Serializer &s, GLShader *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {
            if (count > 1)
                s.push(value->name());
            auto all_keys = s.keys();

            std::set<std::string> keys;
            for (auto key : all_keys) {
                auto it = key.find(".");
                if (it != std::string::npos)
                    keys.insert(key.substr(0, it));
            }
            value->bind();
            for (auto key : keys) {
                if (value->mBufferObjects.find(key) == value->mBufferObjects.end()) {
                    GLuint bufferID;
                    glGenBuffers(1, &bufferID);
                    value->mBufferObjects[key].id = bufferID;
                }
            }
        }
    }
};

```

```
    }
    GLShader::Buffer &buf = value->mBufferObjects[key];
    Eigen::Matrix<uint8_t, Eigen::Dynamic, Eigen::Dynamic> data;

    s.push(key);
    s.get("glType", buf.glType);
    s.get("compSize", buf.compSize);
    s.get("dim", buf.dim);
    s.get("size", buf.size);
    s.get("version", buf.version);
    s.get("data", data);
    s.pop();

    size_t totalSize = (size_t) buf.size * (size_t) buf.compSize;
    if (key == "indices") {
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buf.id);
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, totalSize,
                     (void *) data.data(), GL_DYNAMIC_DRAW);
    } else {
        int attribID = value->attrib(key);
        glEnableVertexAttribArray(attribID);
        glBindBuffer(GL_ARRAY_BUFFER, buf.id);
        glBufferData(GL_ARRAY_BUFFER, totalSize, (void *) data.data(),
                     GL_DYNAMIC_DRAW);
        glVertexAttribPointer(attribID, buf.dim, buf.glType,
                              buf.compSize == 1 ? GL_TRUE : GL_FALSE, 0,
↪0);
    }
}
if (count > 1)
    s.pop();
++value;
}
}
};

#endif // DOXYGEN_SHOULD_SKIP_THIS

NAMESPACE_END(detail)
NAMESPACE_END(nanogui)
```

Includes

- nanogui/glutil.h (*File glutil.h*)
- nanogui/serializer/core.h (*File core.h*)
- set

Namespaces

- *Namespace nanogui*
- *Namespace nanogui::detail*

File opengl.h

Definition (include/nanogui/opengl.h)

Program Listing for File opengl.h

- [Return to documentation for File opengl.h](#)

```

/*
   nanogui/opengl.h -- Pulls in OpenGL, GLAD (if needed), GLFW, and
   NanoVG header files

   NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
   The widget drawing code is based on the NanoVG demo application
   by Mikko Mononen.

   All rights reserved. Use of this source code is governed by a
   BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/common.h>

#ifndef DOXYGEN_SHOULD_SKIP_THIS
#if defined(NANOGUI_GLAD)
    #if defined(NANOGUI_SHARED) && !defined(GLAD_GLAPI_EXPORT)
        #define GLAD_GLAPI_EXPORT
    #endif
#endif

    #include <glad/glad.h>
#else
    #if defined(__APPLE__)
        #define GLFW_INCLUDE_GLCOREARB
    #else
        #define GL_GLEXT_PROTOTYPES
    #endif
#endif
#endif // DOXYGEN_SHOULD_SKIP_THIS

#include <GLFW/glfw3.h>
#include <nanovg.h>

// Special treatment of linux Nvidia opengl headers
#if !defined(_WIN32) && !defined(__APPLE__)
    #if !defined(GL_UNIFORM_BUFFER)
        #warning NanoGUI suspects you have the NVIDIA OpenGL headers installed. \
            Compilation will likely fail. If it does, you have two choices: \
            (1) Re-install the mesa-libGL header files. \
            (2) Compile with NANOGUI_USE_GLAD.
    #endif
#endif

namespace nanogui

inline Color::operator const NVGcolor &() const {
    return reinterpret_cast<const NVGcolor &>(*this->data());
}

```

```
inline bool nvglIsImageIcon(int value) { return value < 1024; }

inline bool nvglIsFontIcon(int value) { return value >= 1024; }

NAMESPACE_END(nanogui)
```

Includes

- `GLFW/glfw3.h`
- `nanogui/common.h` (*File common.h*)
- `nanovg.h`

Included By

- *File glcanvas.h*
- *File glutil.h*

Namespaces

- *Namespace nanogui*

Functions

- *Function nvglIsFontIcon*
- *Function nvglIsImageIcon*

File popup.h

Definition (`include/nanogui/popup.h`)

Program Listing for File popup.h

- [Return to documentation for *File popup.h*](#)

```
/*
    nanogui/popup.h -- Simple popup widget which is attached to another given
    window (can be nested)

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once
```



```

#include <nanogui/window.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Popup : public Window {
public:
    enum Side { Left = 0, Right };

    Popup(Widget *parent, Window *parentWindow);

    void setAnchorPos(const Vector2i &anchorPos) { mAnchorPos = anchorPos; }
    const Vector2i &anchorPos() const { return mAnchorPos; }

    void setAnchorHeight(int anchorHeight) { mAnchorHeight = anchorHeight; }
    int anchorHeight() const { return mAnchorHeight; }

    void setSide(Side popupSide) { mSide = popupSide; }
    Side side() const { return mSide; }

    Window *parentWindow() { return mParentWindow; }
    const Window *parentWindow() const { return mParentWindow; }

    virtual void performLayout(NVGcontext *ctx) override;

    virtual void draw(NVGcontext* ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    virtual void refreshRelativePlacement() override;

protected:
    Window *mParentWindow;
    Vector2i mAnchorPos;
    int mAnchorHeight;
    Side mSide;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- `nanogui/window.h` (*File window.h*)

Included By

- *File nanogui.h*
- *File popupbutton.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Popup*

Enums

- *Enum Popup::Side*

File popupbutton.h

Definition (include/nanogui/popupbutton.h)

Program Listing for File popupbutton.h

- [Return to documentation for *File popupbutton.h*](#)

```
/*
    nanogui/popupbutton.h -- Button which launches a popup widget

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/button.h>
#include <nanogui/popup.h>
#include <nanogui/entypo.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT PopupButton : public Button {
public:
    PopupButton(Widget *parent, const std::string &caption = "Untitled",
               int buttonIcon = 0);

    void setChevronIcon(int icon) { mChevronIcon = icon; }
    int chevronIcon() const { return mChevronIcon; }

    void setSide(Popup::Side popupSide);
    Popup::Side side() const { return mPopup->side(); }

    Popup *popup() { return mPopup; }
    const Popup *popup() const { return mPopup; }
```

```

    virtual void draw(NVGcontext* ctx) override;
    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void performLayout(NVGcontext *ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    Popup *mPopup;
    int mChevronIcon;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- `nanogui/button.h` (*File button.h*)
- `nanogui/entypo.h` (*File entypo.h*)
- `nanogui/popup.h` (*File popup.h*)

Included By

- *File colorpicker.h*
- *File combobox.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class PopupButton*

File progressbar.h

Definition (`include/nanogui/progressbar.h`)

Program Listing for File progressbar.h

- [Return to documentation for *File progressbar.h*](#)

```

/*
    nanogui/progressbar.h -- Standard widget for visualizing progress

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.

```

```
The widget drawing code is based on the NanoVG demo application
by Mikko Mononen.

All rights reserved. Use of this source code is governed by a
BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT ProgressBar : public Widget {
public:
    ProgressBar(Widget *parent);

    float value() { return mValue; }
    void setValue(float value) { mValue = value; }

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void draw(NVGcontext* ctx) override;

    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    float mValue;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ProgressBar*

File python.h

Definition (include/nanogui/python.h)

Program Listing for File python.h

- [Return to documentation for File python.h](#)

```

/*
    nanogui/python.h -- Macros to facilitate Python bindings of custom widgets

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/common.h>
#include <pybind11/pybind11.h>

#define NANOGUI_WIDGET_OVERLOADS(Parent) \
    bool mouseButtonEvent(const ::nanogui::Vector2i &p, int button, bool down, int_ \
    ↪modifiers) { \
        PYBIND11_OVERLOAD(bool, Parent, mouseButtonEvent, p, button, down, modifiers); \
    } \
    bool mouseMotionEvent(const ::nanogui::Vector2i &p, const ::nanogui::Vector2i & \
    ↪rel, int button, int modifiers) { \
        PYBIND11_OVERLOAD(bool, Parent, mouseMotionEvent, p, rel, button, modifiers); \
    } \
    bool mouseDragEvent(const ::nanogui::Vector2i &p, const ::nanogui::Vector2i &rel, \
    ↪int button, int modifiers) { \
        PYBIND11_OVERLOAD(bool, Parent, mouseDragEvent, p, rel, button, modifiers); \
    } \
    bool mouseEnterEvent(const ::nanogui::Vector2i &p, bool enter) { \
        PYBIND11_OVERLOAD(bool, Parent, mouseEnterEvent, p, enter); \
    } \
    bool scrollEvent(const ::nanogui::Vector2i &p, const ::nanogui::Vector2f &rel) { \
        PYBIND11_OVERLOAD(bool, Parent, scrollEvent, p, rel); \
    } \
    bool focusEvent(bool focused) { \
        PYBIND11_OVERLOAD(bool, Parent, focusEvent, focused); \
    } \
    bool keyboardEvent(int key, int scancode, int action, int modifiers) { \
        PYBIND11_OVERLOAD(bool, Parent, keyboardEvent, key, scancode, action, \
    ↪modifiers); \
    } \
    bool keyboardCharacterEvent(unsigned int codepoint) { \
        PYBIND11_OVERLOAD(bool, Parent, keyboardCharacterEvent, codepoint); \
    } \
    ::nanogui::Vector2i preferredSize(NVGcontext *ctx) const { \
        PYBIND11_OVERLOAD(::nanogui::Vector2i, Parent, preferredSize, ctx); \
    } \

```

```
void performLayout(NVGcontext *ctx) { \
    PYBIND11_OVERLOAD(void, Parent, performLayout, ctx); \
} \
void draw(NVGcontext *ctx) { \
    PYBIND11_OVERLOAD(void, Parent, draw, ctx); \
}

#define NANOGUI_LAYOUT_OVERLOADS(Parent) \
    ::nanogui::Vector2i preferredSize(NVGcontext *ctx, const ::nanogui::Widget_ \
↪ *widget) const { \
        PYBIND11_OVERLOAD(::nanogui::Vector2i, Parent, preferredSize, ctx, widget); \
    } \
    void performLayout(NVGcontext *ctx, ::nanogui::Widget *widget) const { \
        PYBIND11_OVERLOAD(void, Parent, performLayout, ctx, widget); \
    }

#define NANOGUI_SCREEN_OVERLOADS(Parent) \
    virtual void drawAll() { \
        PYBIND11_OVERLOAD(void, Parent, drawAll); \
    } \
    virtual void drawContents() { \
        PYBIND11_OVERLOAD(void, Parent, drawContents); \
    } \
    virtual bool dropEvent(const std::vector<std::string> &filenames) { \
        PYBIND11_OVERLOAD(bool, Parent, dropEvent, filenames); \
    } \
    virtual bool resizeEvent(const ::nanogui::Vector2i &size) { \
        PYBIND11_OVERLOAD(bool, Parent, resizeEvent, size); \
    }
```

Includes

- `nanogui/common.h` (*File common.h*)
- `pybind11/pybind11.h`

Defines

- *Define NANOGUI_LAYOUT_OVERLOADS*
- *Define NANOGUI_SCREEN_OVERLOADS*
- *Define NANOGUI_WIDGET_OVERLOADS*

File screen.h

Definition (`include/nanogui/screen.h`)

Program Listing for File screen.h

- Return to documentation for *File screen.h*

```

/*
    nanogui/screen.h -- Top-level widget and interface between NanoGUI and GLFW

    A significant redesign of this code was contributed by Christian Schueller.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Screen : public Widget {
    friend class Widget;
    friend class Window;
public:
    Screen(const Vector2i &size, const std::string &caption,
           bool resizable = true, bool fullscreen = false, int colorBits = 8,
           int alphaBits = 8, int depthBits = 24, int stencilBits = 8,
           int nSamples = 0,
           unsigned int glMajor = 3, unsigned int glMinor = 3);

    virtual ~Screen();

    const std::string &caption() const { return mCaption; }

    void setCaption(const std::string &caption);

    const Color &background() const { return mBackground; }

    void setBackground(const Color &background) { mBackground = background; }

    void setVisible(bool visible);

    void setSize(const Vector2i& size);

    virtual void drawAll();

    virtual void drawContents() { /* To be overridden */ }

    float pixelRatio() const { return mPixelRatio; }

    virtual bool dropEvent(const std::vector<std::string> & /* filenames */) { return
↪false; /* To be overridden */ }

    virtual bool keyboardEvent(int key, int scancode, int action, int modifiers);

    virtual bool keyboardCharacterEvent(unsigned int codepoint);

    virtual bool resizeEvent(const Vector2i& size);

```

```
std::function<void(Vector2i)> resizeCallback() const { return mResizeCallback; }
void setResizeCallback(const std::function<void(Vector2i)> &callback) {
↳mResizeCallback = callback; }

Vector2i mousePos() const { return mMousePos; }

GLFWwindow *glfwWindow() { return mGLFWWindow; }

NVGcontext *nvgContext() { return mNVGContext; }

void setShutdownGLFWOnDestruct(bool v) { mShutdownGLFWOnDestruct = v; }
bool shutdownGLFWOnDestruct() { return mShutdownGLFWOnDestruct; }

using Widget::performLayout;

void performLayout() {
    Widget::performLayout(mNVGContext);
}

public:
    /****** API for applications which manage GLFW themselves *****/

    Screen();

    void initialize(GLFWwindow *window, bool shutdownGLFWOnDestruct);

    /* Event handlers */
    bool cursorPosCallbackEvent(double x, double y);
    bool mouseButtonCallbackEvent(int button, int action, int modifiers);
    bool keyCallbackEvent(int key, int scancode, int action, int mods);
    bool charCallbackEvent(unsigned int codepoint);
    bool dropCallbackEvent(int count, const char **filenames);
    bool scrollCallbackEvent(double x, double y);
    bool resizeCallbackEvent(int width, int height);

    /* Internal helper functions */
    void updateFocus(Widget *widget);
    void disposeWindow(Window *window);
    void centerWindow(Window *window);
    void moveWindowToFront(Window *window);
    void drawWidgets();

protected:
    GLFWwindow *mGLFWWindow;
    NVGcontext *mNVGContext;
    GLFWcursor *mCursors[(int) Cursor::CursorCount];
    Cursor mCursor;
    std::vector<Widget *> mFocusPath;
    Vector2i mFBSize;
    float mPixelRatio;
    int mMouseState, mModifiers;
    Vector2i mMousePos;
    bool mDragActive;
    Widget *mDragWidget = nullptr;
    double mLastInteraction;
    bool mProcessEvent;
    Color mBackground;
    std::string mCaption;
```



```

    bool mShutdownGLFWOnDestruct;
    bool mFullscreen;
    std::function<void(Vector2i)> mResizeCallback;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)

```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Screen*

File slider.h

Definition (`include/nanogui/slider.h`)

Program Listing for File slider.h

- [Return to documentation for *File slider.h*](#)

```

/*
    nanogui/slider.h -- Fractional slider widget with mouse control

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN (nanogui)

```

```

class NANOGUI_EXPORT Slider : public Widget {
public:
    Slider(Widget *parent);

    float value() const { return mValue; }
    void setValue(float value) { mValue = value; }

    const Color &highlightColor() const { return mHighlightColor; }
    void setHighlightColor(const Color &highlightColor) { mHighlightColor =
↪highlightColor; }

    std::pair<float, float> range() const { return mRange; }
    void setRange(std::pair<float, float> range) { mRange = range; }

    std::pair<float, float> highlightedRange() const { return mHighlightedRange; }
    void setHighlightedRange(std::pair<float, float> highlightedRange) {
↪mHighlightedRange = highlightedRange; }

    std::function<void(float)> callback() const { return mCallback; }
    void setCallback(const std::function<void(float)> &callback) { mCallback =
↪callback; }

    std::function<void(float)> finalCallback() const { return mFinalCallback; }
    void setFinalCallback(const std::function<void(float)> &callback) {
↪mFinalCallback = callback; }

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↪int modifiers) override;
    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int
↪modifiers) override;
    virtual void draw(NVGcontext* ctx) override;
    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;

protected:
    float mValue;
    std::function<void(float)> mCallback;
    std::function<void(float)> mFinalCallback;
    std::pair<float, float> mRange;
    std::pair<float, float> mHighlightedRange;
    Color mHighlightColor;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- nanogui/widget.h (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Slider*

File sparse.h

Definition (include/nanogui/serializer/sparse.h)

Program Listing for File sparse.h

- [Return to documentation for *File sparse.h*](#)

```

/*
    nanogui/serializer/sparse.h -- serialization support for sparse matrices

    NanoGUI was developed by Wenzel Jakob <wenzel@inf.ethz.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/serializer/core.h>
#include <Eigen/SparseCore>

NAMESPACE_BEGIN(nanogui)
NAMESPACE_BEGIN(detail)

// bypass template specializations
#ifndef DOXYGEN_SHOULD_SKIP_THIS

template <typename Scalar, int Options, typename Index>
struct serialization_helper<Eigen::SparseMatrix<Scalar, Options, Index>> {
    typedef Eigen::SparseMatrix<Scalar, Options, Index> Matrix;
    typedef Eigen::Triplet<Scalar> Triplet;

    static std::string type_id() {
        return "S" + serialization_helper<Index>::type_id() + serialization_helper
↪<Scalar>::type_id();
    }

    static void write(Serializer &s, const Matrix *value, size_t count) {
        for (size_t i = 0; i < count; ++i) {

```

```

    size_t index = 0;
    std::vector<std::pair<Index, Index>> positions(value->nonZeros());
    std::vector<Scalar> coeffs(value->nonZeros());

    for (int k = 0; k < value->outerSize(); ++k) {
        for (typename Matrix::InnerIterator it(*value, k); it; ++it) {
            positions[index] = std::make_pair(it.row(), it.col());
            coeffs[index] = it.value();
            index++;
        }
    }

    Index rows = value->rows(), cols = value->cols();
    s.write(&rows, sizeof(Index));
    s.write(&cols, sizeof(Index));
    serialization_helper<std::vector<std::pair<Index, Index>>>::write(s, &
↪positions, 1);
    serialization_helper<std::vector<Scalar>>::write(s, &coeffs, 1);

    ++value;
}

}

static void read(Serializer &s, Matrix *value, size_t count) {
    for (size_t i = 0; i < count; ++i) {
        Index rows, cols;
        s.read(&rows, sizeof(Index));
        s.read(&cols, sizeof(Index));

        std::vector<std::pair<Index, Index>> positions;
        std::vector<Scalar> coeffs;
        serialization_helper<std::vector<std::pair<Index, Index>>>::read(s, &
↪positions, 1);
        serialization_helper<std::vector<Scalar>>::read(s, &coeffs, 1);

        if (coeffs.size() != positions.size())
            throw std::runtime_error("Encountered corrupt data while_
↪unserializing sparse matrix!");

        std::vector<Triplet> triplets(coeffs.size());

        for (uint32_t i=0; i<coeffs.size(); ++i)
            triplets[i] = Triplet(positions[i].first, positions[i].second,
↪coeffs[i]);

        value->resize(rows, cols);
        value->setFromTriplets(triplets.begin(), triplets.end());

        ++value;
    }
}

};

#endif // DOXYGEN_SHOULD_SKIP_THIS

NAMESPACE_END(detail)
NAMESPACE_END(nanogui)

```

Includes

- Eigen/SparseCore
- nanogui/serializer/core.h (*File core.h*)

Namespaces

- *Namespace nanogui*
- *Namespace nanogui::detail*

File stackedwidget.h

Definition (include/nanogui/stackedwidget.h)

Program Listing for File stackedwidget.h

- Return to documentation for *File stackedwidget.h*

```

/*
    nanogui/stackedwidget.h -- Widget used to stack widgets on top
    of each other. Only the active widget is visible.

    The stacked widget was contributed by Stefan Ivanov.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT StackedWidget : public Widget {
public:
    StackedWidget(Widget* parent);

    void setSelectedIndex(int index);
    int selectedIndex() const;

    virtual void performLayout(NVGcontext* ctx) override;
    virtual Vector2i preferredSize(NVGcontext* ctx) const override;
    virtual void addChild(int index, Widget* widget) override;

private:
    int mSelectedIndex = -1;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

```

```
};  
  
NAMESPACE_END (nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class StackedWidget*

File tabheader.h

Definition (`include/nanogui/tabheader.h`)

Program Listing for File tabheader.h

- [Return to documentation for *File tabheader.h*](#)

```
/*  
    nanogui/tabheader.h -- Widget used to control tabs.  
  
    The tab header widget was contributed by Stefan Ivanov.  
  
    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.  
    The widget drawing code is based on the NanoVG demo application  
    by Mikko Mononen.  
  
    All rights reserved. Use of this source code is governed by a  
    BSD-style license that can be found in the LICENSE.txt file.  
*/  
#pragma once  
  
#include <nanogui/widget.h>  
#include <vector>  
#include <string>  
#include <functional>  
#include <utility>  
#include <iterator>
```

```

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT TabHeader : public Widget {
public:
    TabHeader(Widget *parent, const std::string &font = "sans-bold");

    void setFont(const std::string& font) { mFont = font; }
    const std::string& font() const { return mFont; }
    bool overflowing() const { return mOverflowing; }

    void setCallback(const std::function<void(int)>& callback) { mCallback = callback; }
    ↪;
    const std::function<void(int)>& callback() const { return mCallback; }

    void setActiveTab(int tabIndex);
    int activeTab() const;
    bool isTabVisible(int index) const;
    int tabCount() const { return (int) mTabButtons.size(); }

    void addTab(const std::string& label);

    void addTab(int index, const std::string& label);

    int removeTab(const std::string& label);

    void removeTab(int index);

    const std::string& tabLabelAt(int index) const;

    int tabIndex(const std::string& label);

    void ensureTabVisible(int index);

    std::pair<Vector2i, Vector2i> visibleButtonArea() const;

    std::pair<Vector2i, Vector2i> activeButtonArea() const;

    virtual void performLayout(NVGcontext* ctx) override;
    virtual Vector2i preferredSize(NVGcontext* ctx) const override;
    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
    ↪modifiers) override;

    virtual void draw(NVGcontext* ctx) override;

private:
    class TabButton {
    public:
        constexpr static const char* dots = "...";

        TabButton(TabHeader& header, const std::string& label);

        void setLabel(const std::string& label) { mLabel = label; }
        const std::string& label() const { return mLabel; }
        void setSize(const Vector2i& size) { mSize = size; }
        const Vector2i& size() const { return mSize; }

        Vector2i preferredSize(NVGcontext* ctx) const;

```

```

    void calculateVisibleString(NVGcontext* ctx);
    void drawAtPosition(NVGcontext* ctx, const Vector2i& position, bool active);
    void drawActiveBorderAt(NVGcontext * ctx, const Vector2i& position, float_
↳offset, const Color& color);
    void drawInactiveBorderAt(NVGcontext * ctx, const Vector2i& position, float_
↳offset, const Color& color);

private:
    TabHeader* mHeader;
    std::string mLabel;
    Vector2i mSize;

    struct StringView {
        const char* first = nullptr;
        const char* last = nullptr;
    };
    StringView mVisibleText;
    int mVisibleWidth = 0;
};

using TabIterator = std::vector<TabButton>::iterator;
using ConstTabIterator = std::vector<TabButton>::const_iterator;

enum class ClickLocation {
    LeftControls, RightControls, TabButtons
};

TabIterator visibleBegin() { return std::next(mTabButtons.begin(), mVisibleStart);
↳ }
TabIterator visibleEnd() { return std::next(mTabButtons.begin(), mVisibleEnd); }
TabIterator activeIterator() { return std::next(mTabButtons.begin(), mActiveTab);
↳ }
TabIterator tabIterator(int index) { return std::next(mTabButtons.begin(), index);
↳ }

ConstTabIterator visibleBegin() const { return std::next(mTabButtons.begin(),
↳mVisibleStart); }
ConstTabIterator visibleEnd() const { return std::next(mTabButtons.begin(),
↳mVisibleEnd); }
ConstTabIterator activeIterator() const { return std::next(mTabButtons.begin(),
↳mActiveTab); }
ConstTabIterator tabIterator(int index) const { return std::next(mTabButtons.
↳begin(), index); }

void calculateVisibleEnd();

void drawControls(NVGcontext* ctx);
ClickLocation locateClick(const Vector2i& p);
void onArrowLeft();
void onArrowRight();

std::function<void(int)> mCallback;
std::vector<TabButton> mTabButtons;
int mVisibleStart = 0;
int mVisibleEnd = 0;
int mActiveTab = 0;
bool mOverflowing = false;

```



```
    std::string mFont;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)
```

Includes

- `functional`
- `iterator`
- `nanogui/widget.h` (*File widget.h*)
- `string`
- `utility`
- `vector`

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Struct TabButton::StringView*
- *Class TabHeader*
- *Class TabHeader::TabButton*

Enums

- *Enum TabHeader::ClickLocation*

File tabwidget.h

Definition (`include/nanogui/tabwidget.h`)

Program Listing for File tabwidget.h

- [Return to documentation for *File tabwidget.h*](#)

```
/*
    nanogui/tabwidget.h -- A wrapper around the widgets TabHeader and StackedWidget
    which hooks the two classes together.

    The tab widget was contributed by Stefan Ivanov.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>
#include <functional>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT TabWidget : public Widget {
public:
    TabWidget(Widget* parent);

    void setActiveTab(int tabIndex);
    int activeTab() const;
    int tabCount() const;

    void setCallback(const std::function<void(int)> &callback) { mCallback = callback; }
    ↪ const std::function<void(int)> &callback() const { return mCallback; }

    Widget* createTab(const std::string &label);
    Widget* createTab(int index, const std::string &label);

    void addTab(const std::string &label, Widget *tab);

    void addTab(int index, const std::string &label, Widget *tab);

    bool removeTab(const std::string &label);

    void removeTab(int index);

    const std::string &tabLabelAt(int index) const;

    int tabLabelIndex(const std::string &label);

    int tabIndex(Widget* tab);

    void ensureTabVisible(int index);

    const Widget* tab(const std::string &label) const;
    Widget* tab(const std::string &label);

    virtual void performLayout(NVGcontext* ctx) override;
    virtual Vector2i preferredSize(NVGcontext* ctx) const override;
    virtual void draw(NVGcontext* ctx) override;
};
```

```

private:
    TabHeader* mHeader;
    StackedWidget* mContent;
    std::function<void(int)> mCallback;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)

```

Includes

- functional
- nanogui/widget.h (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class TabWidget*

File textbox.h

Definition (include/nanogui/textbox.h)

Program Listing for File textbox.h

- [Return to documentation for *File textbox.h*](#)

```

/*
    nanogui/textbox.h -- Fancy text box with builtin regular
    expression-based validation

    The text box widget was contributed by Christian Schueller.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/

```

```

#pragma once

#include <nanogui/compat.h>
#include <nanogui/widget.h>
#include <sstream>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT TextBox : public Widget {
public:
    enum class Alignment {
        Left,
        Center,
        Right
    };

    TextBox(Widget *parent, const std::string &value = "Untitled");

    bool editable() const { return mEditable; }
    void setEditable(bool editable);

    bool spinnable() const { return mSpinnable; }
    void setSpinnable(bool spinnable) { mSpinnable = spinnable; }

    const std::string &value() const { return mValue; }
    void setValue(const std::string &value) { mValue = value; }

    const std::string &defaultValue() const { return mDefaultValue; }
    void setDefaultValue(const std::string &defaultValue) { mDefaultValue =
↳defaultValue; }

    Alignment alignment() const { return mAlignment; }
    void setAlignment(Alignment align) { mAlignment = align; }

    const std::string &units() const { return mUnits; }
    void setUnits(const std::string &units) { mUnits = units; }

    int unitsImage() const { return mUnitsImage; }
    void setUnitsImage(int image) { mUnitsImage = image; }

    const std::string &format() const { return mFormat; }
    void setFormat(const std::string &format) { mFormat = format; }

    virtual void setTheme(Theme *theme) override;

    std::function<bool(const std::string& str)> callback() const { return mCallback; }
    void setCallback(const std::function<bool(const std::string& str)> &callback) {
↳mCallback = callback; }

    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int
↳modifiers) override;
    virtual bool mouseMotionEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers) override;
    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers) override;
    virtual bool focusEvent(bool focused) override;
    virtual bool keyboardEvent(int key, int scancode, int action, int modifiers)
↳override;

```

```

    virtual bool keyboardCharacterEvent(unsigned int codepoint) override;

    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void draw(NVGcontext* ctx) override;
    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    bool checkFormat(const std::string& input, const std::string& format);
    bool copySelection();
    void pasteFromClipboard();
    bool deleteSelection();

    void updateCursor(NVGcontext *ctx, float lastx,
                     const NVGglyphPosition *glyphs, int size);
    float cursorIndex2Position(int index, float lastx,
                              const NVGglyphPosition *glyphs, int size);
    int position2CursorIndex(float posx, float lastx,
                            const NVGglyphPosition *glyphs, int size);

    enum class SpinArea { None, Top, Bottom };
    SpinArea spinArea(const Vector2i & pos);

protected:
    bool mEditable;
    bool mSpinnable;
    bool mCommitted;
    std::string mValue;
    std::string mDefaultValue;
    Alignment mAlignment;
    std::string mUnits;
    std::string mFormat;
    int mUnitsImage;
    std::function<bool(const std::string& str)> mCallback;
    bool mValidFormat;
    std::string mValueTemp;
    int mCursorPos;
    int mSelectionPos;
    Vector2i mMousePos;
    Vector2i mMouseDownPos;
    Vector2i mMouseDownDragPos;
    int mMouseDownModifier;
    float mTextOffset;
    double mLastClick;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <typename Scalar>
class IntBox : public TextBox {
public:
    IntBox(Widget *parent, Scalar value = (Scalar) 0) : TextBox(parent) {
        setDefaultValue("0");
        setFormat(std::is_signed<Scalar>::value ? "[-]?[0-9]*" : "[0-9]*");
        setValueIncrement(1);
        setMinMaxValues(std::numeric_limits<Scalar>::lowest(), std::numeric_limits
↪<Scalar>::max());
        setValue(value);
        setSpinnable(false);
    }
};

```

```
}

Scalar value() const {
    std::istringstream iss(TextBox::value());
    Scalar value = 0;
    iss >> value;
    return value;
}

void setValue(Scalar value) {
    Scalar clampedValue = std::min(std::max(value, mMinValue), mMaxValue);
    TextBox::setValue(std::to_string(clampedValue));
}

void setCallback(const std::function<void(Scalar)> &cb) {
    TextBox::setCallback(
        [cb, this](const std::string &str) {
            std::istringstream iss(str);
            Scalar value = 0;
            iss >> value;
            setValue(value);
            cb(value);
            return true;
        }
    );
}

void setValueIncrement(Scalar incr) {
    mValueIncrement = incr;
}

void setMinValue(Scalar minValue) {
    mMinValue = minValue;
}

void setMaxValue(Scalar maxValue) {
    mMaxValue = maxValue;
}

void setMinMaxValues(Scalar minValue, Scalar maxValue) {
    setMinValue(minValue);
    setMaxValue(maxValue);
}

virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↳modifiers) override {
    if ((mEditable || mSpinnable) && down)
        mMouseDownValue = value();

    SpinArea area = spinArea(p);
    if (mSpinnable && area != SpinArea::None && down && !focused()) {
        if (area == SpinArea::Top) {
            setValue(value() + mValueIncrement);
            if (mCallback)
                mCallback(mValue);
        } else if (area == SpinArea::Bottom) {
            setValue(value() - mValueIncrement);
            if (mCallback)
                mCallback(mValue);
        }
    }
    return true;
}
```

```

    }

    return TextBox::mouseButtonEvent(p, button, down, modifiers);
}

virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers) override {
    if (TextBox::mouseDragEvent(p, rel, button, modifiers)) {
        return true;
    }
    if (mSpinnable && !focused() && button == 2 /* 1 << GLFW_MOUSE_BUTTON_2 */ &&
↳mMouseDownPos.x() != -1) {
        int valueDelta = static_cast<int>((p.x() - mMouseDownPos.x()) /
↳float(10));
        setValue(mMouseDownValue + valueDelta * mValueIncrement);
        if (mCallback)
            mCallback(mValue);
        return true;
    }
    return false;
}

virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel) override {
    if (Widget::scrollEvent(p, rel)) {
        return true;
    }
    if (mSpinnable && !focused()) {
        int valueDelta = (rel.y() > 0) ? 1 : -1;
        setValue(value() + valueDelta*mValueIncrement);
        if (mCallback)
            mCallback(mValue);
        return true;
    }
    return false;
}

private:
    Scalar mMouseDownValue;
    Scalar mValueIncrement;
    Scalar mMinValue, mMaxValue;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

template <typename Scalar>
class FloatBox : public TextBox {
public:
    FloatBox(Widget *parent, Scalar value = (Scalar) 0.f) : TextBox(parent) {
        mNumberFormat = sizeof(Scalar) == sizeof(float) ? "%.4g" : "%.7g";
        setDefaultValue("0");
        setFormat("[+]?[0-9]*\\.?[0-9]+([eE][+]?[0-9]+)?");
        setValueIncrement((Scalar) 0.1);
        setMinMaxValues(std::numeric_limits<Scalar>::lowest(), std::numeric_limits
↳<Scalar>::max());
        setValue(value);
        setSpinnable(false);
    }

    std::string numberFormat() const { return mNumberFormat; }
    void numberFormat(const std::string &format) { mNumberFormat = format; }

```

```
Scalar value() const {
    return (Scalar) std::stod(TextBox::value());
}

void setValue(Scalar value) {
    Scalar clampedValue = std::min(std::max(value, mMinValue), mMaxValue);
    char buffer[50];
    NANOGUI_SNPRINTF(buffer, 50, mNumberFormat.c_str(), clampedValue);
    TextBox::setValue(buffer);
}

void setCallback(const std::function<void(Scalar)> &cb) {
    TextBox::setCallback([cb, this](const std::string &str) {
        Scalar scalar = (Scalar) std::stod(str);
        setValue(scalar);
        cb(scalar);
        return true;
    });
}

void setValueIncrement(Scalar incr) {
    mValueIncrement = incr;
}

void setMinValue(Scalar minValue) {
    mMinValue = minValue;
}

void setMaxValue(Scalar maxValue) {
    mMaxValue = maxValue;
}

void setMinMaxValues(Scalar minValue, Scalar maxValue) {
    setMinValue(minValue);
    setMaxValue(maxValue);
}

virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↳modifiers) override {
    if ((mEditable || mSpinnable) && down)
        mMouseDownValue = value();

    SpinArea area = spinArea(p);
    if (mSpinnable && area != SpinArea::None && down && !focused()) {
        if (area == SpinArea::Top) {
            setValue(value() + mValueIncrement);
            if (mCallback)
                mCallback(mValue);
        } else if (area == SpinArea::Bottom) {
            setValue(value() - mValueIncrement);
            if (mCallback)
                mCallback(mValue);
        }
        return true;
    }

    return TextBox::mouseButtonEvent(p, button, down, modifiers);
}

virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers) override {
    if (TextBox::mouseDragEvent(p, rel, button, modifiers)) {
```



```

        return true;
    }
    if (mSpinnable && !focused() && button == 2 /* 1 << GLFW_MOUSE_BUTTON_2 */ &&
↪mMouseDownPos.x() != -1) {
        int valueDelta = static_cast<int>((p.x() - mMouseDownPos.x()) /
↪float(10));
        setValue(mMouseDownValue + valueDelta * mValueIncrement);
        if (mCallback)
            mCallback(mValue);
        return true;
    }
    return false;
}
virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel) override {
    if (Widget::scrollEvent(p, rel)) {
        return true;
    }
    if (mSpinnable && !focused()) {
        int valueDelta = (rel.y() > 0) ? 1 : -1;
        setValue(value() + valueDelta*mValueIncrement);
        if (mCallback)
            mCallback(mValue);
        return true;
    }
    return false;
}

private:
    std::string mNumberFormat;
    Scalar mMouseDownValue;
    Scalar mValueIncrement;
    Scalar mMinValue, mMaxValue;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- nanogui/compat.h (*File compat.h*)
- nanogui/widget.h (*File widget.h*)
- sstream

Included By

- *File formhelper.h*
- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class FloatBox*
- *Class IntBox*
- *Class TextBox*

Enums

- *Enum TextBox::Alignment*
- *Enum TextBox::SpinArea*

File theme.h

Definition (include/nanogui/theme.h)

Program Listing for File theme.h

- [Return to documentation for *File theme.h*](#)

```
/*
    nanogui/theme.h -- Storage class for basic theme-related properties

    The text box widget was contributed by Christian Schueller.

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/common.h>
#include <nanogui/object.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Theme : public Object {
public:
    Theme(NVGcontext *ctx);

    /* Fonts */
    int mFontNormal;
    int mFontBold;
    int mFontIcons;

    /* Spacing-related parameters */
    int mStandardFontSize;
    int mButtonFontSize;
    int mTextBoxFontSize;
}
```

```

    int mWindowCornerRadius;
    int mWindowHeaderHeight;
    int mWindowDropShadowSize;
    int mButtonCornerRadius;
    float mTabBorderWidth;
    int mTabInnerMargin;
    int mTabMinButtonWidth;
    int mTabMaxButtonWidth;
    int mTabControlWidth;
    int mTabButtonHorizontalPadding;
    int mTabButtonVerticalPadding;

    /* Generic colors */
    Color mDropShadow;
    Color mTransparent;
    Color mBorderDark;
    Color mBorderLight;
    Color mBorderMedium;
    Color mTextColor;
    Color mDisabledTextColor;
    Color mTextColorShadow;
    Color mIconColor;

    /* Button colors */
    Color mButtonGradientTopFocused;
    Color mButtonGradientBotFocused;
    Color mButtonGradientTopUnfocused;
    Color mButtonGradientBotUnfocused;
    Color mButtonGradientTopPushed;
    Color mButtonGradientBotPushed;

    /* Window colors */
    Color mWindowFillUnfocused;
    Color mWindowFillFocused;
    Color mWindowTitleUnfocused;
    Color mWindowTitleFocused;

    Color mWindowHeaderGradientTop;
    Color mWindowHeaderGradientBot;
    Color mWindowHeaderSepTop;
    Color mWindowHeaderSepBot;

    Color mWindowPopup;
    Color mWindowPopupTransparent;
protected:
    virtual ~Theme() { };
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)

```

Includes

- nanogui/common.h (*File common.h*)
- nanogui/object.h (*File object.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Theme*

File toolbutton.h

Definition (include/nanogui/toolbutton.h)

Program Listing for File toolbutton.h

- [Return to documentation for *File toolbutton.h*](#)

```
/*
    nanogui/toolbutton.h -- Simple radio+toggle button with an icon

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/button.h>

NAMESPACE_BEGIN(nanogui)

class ToolButton : public Button {
public:
    ToolButton(Widget *parent, int icon,
               const std::string &caption = "")
        : Button(parent, caption, icon) {
        setFlags(Flags::RadioButton | Flags::ToggleButton);
        setFixedSize(Vector2i(25, 25));
    }
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/button.h` (*File button.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class ToolButton*

File vscrollpanel.h

Definition (`include/nanogui/vscrollpanel.h`)

Program Listing for File vscrollpanel.h

- [Return to documentation for *File vscrollpanel.h*](#)

```

/*
   nanogui/vscrollpanel.h -- Adds a vertical scrollbar around a widget
   that is too big to fit into a certain area

   NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
   The widget drawing code is based on the NanoVG demo application
   by Mikko Mononen.

   All rights reserved. Use of this source code is governed by a
   BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT VScrollPanel : public Widget {
public:
    VScrollPanel(Widget *parent);

    virtual void performLayout(NVGcontext *ctx) override;
    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↪int modifiers) override;
    virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel) override;

```

```
virtual void draw(NVGcontext *ctx) override;
virtual void save(Serializer &s) const override;
virtual bool load(Serializer &s) override;
protected:
    int mChildPreferredHeight;
    float mScroll;
    bool mUpdateLayout;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END (nanogui)
```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File nanogui.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class VScrollPanel*

File widget.h

Definition (`include/nanogui/widget.h`)

Program Listing for File widget.h

- [Return to documentation for *File widget.h*](#)

```
/*
    nanogui/widget.h -- Base class of all widgets

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once
```

```

#include <nanogui/object.h>
#include <vector>

NAMESPACE_BEGIN(nanogui)

enum class Cursor; // do not put a docstring, this is already documented

class NANOGUI_EXPORT Widget : public Object {
public:
    Widget(Widget *parent);

    Widget *parent() { return mParent; }
    const Widget *parent() const { return mParent; }
    void setParent(Widget *parent) { mParent = parent; }

    Layout *layout() { return mLayout; }
    const Layout *layout() const { return mLayout.get(); }
    void setLayout(Layout *layout) { mLayout = layout; }

    Theme *theme() { return mTheme; }
    const Theme *theme() const { return mTheme.get(); }
    virtual void setTheme(Theme *theme);

    const Vector2i &position() const { return mPos; }
    void setPosition(const Vector2i &pos) { mPos = pos; }

    Vector2i absolutePosition() const {
        return mParent ?
            (parent()->absolutePosition() + mPos) : mPos;
    }

    const Vector2i &size() const { return mSize; }
    void setSize(const Vector2i &size) { mSize = size; }

    int width() const { return mSize.x(); }
    void setWidth(int width) { mSize.x() = width; }

    int height() const { return mSize.y(); }
    void setHeight(int height) { mSize.y() = height; }

    void setFixedSize(const Vector2i &fixedSize) { mFixedSize = fixedSize; }

    const Vector2i &fixedSize() const { return mFixedSize; }

    // Return the fixed width (see \ref setFixedSize())
    int fixedWidth() const { return mFixedSize.x(); }
    // Return the fixed height (see \ref setFixedSize())
    int fixedHeight() const { return mFixedSize.y(); }
    void setFixedWidth(int width) { mFixedSize.x() = width; }
    void setFixedHeight(int height) { mFixedSize.y() = height; }

    bool visible() const { return mVisible; }
    void setVisible(bool visible) { mVisible = visible; }

    bool visibleRecursive() const {
        bool visible = true;
        const Widget *widget = this;
        while (widget) {

```

```
        visible &= widget->visible();
        widget = widget->parent();
    }
    return visible;
}

int childCount() const { return (int) mChildren.size(); }

const std::vector<Widget *> &children() const { return mChildren; }

virtual void addChild(int index, Widget *widget);

void addChild(Widget *widget);

void removeChild(int index);

void removeChild(const Widget *widget);

const Widget* childAt(int index) const { return mChildren[index]; }

Widget* childAt(int index) { return mChildren[index]; }

int childIndex(Widget* widget) const;

template<typename WidgetClass, typename... Args>
WidgetClass* add(const Args&... args) {
    return new WidgetClass(this, args...);
}

Window *window();

void setId(const std::string &id) { mId = id; }
const std::string &id() const { return mId; }

bool enabled() const { return mEnabled; }
void setEnabled(bool enabled) { mEnabled = enabled; }

bool focused() const { return mFocused; }
void setFocused(bool focused) { mFocused = focused; }
void requestFocus();

const std::string &tooltip() const { return mTooltip; }
void setTooltip(const std::string &tooltip) { mTooltip = tooltip; }

int fontSize() const;
void setFontSize(int fontSize) { mFontSize = fontSize; }
bool hasFontSize() const { return mFontSize > 0; }

Cursor cursor() const { return mCursor; }
void setCursor(Cursor cursor) { mCursor = cursor; }

bool contains(const Vector2i &p) const {
    auto d = (p-mPos).array();
    return (d >= 0).all() && (d < mSize.array()).all();
}

Widget *findWidget(const Vector2i &p);
```



```

    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↳modifiers);

    virtual bool mouseMotionEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers);

    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↳int modifiers);

    virtual bool mouseEnterEvent(const Vector2i &p, bool enter);

    virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel);

    virtual bool focusEvent(bool focused);

    virtual bool keyboardEvent(int key, int scancode, int action, int modifiers);

    virtual bool keyboardCharacterEvent(unsigned int codepoint);

    virtual Vector2i preferredSize(NVGcontext *ctx) const;

    virtual void performLayout(NVGcontext *ctx);

    virtual void draw(NVGcontext *ctx);

    virtual void save(Serializer &s) const;

    virtual bool load(Serializer &s);

protected:
    virtual ~Widget();

protected:
    Widget *mParent;
    ref<Theme> mTheme;
    ref<Layout> mLayout;
    std::string mId;
    Vector2i mPos, mSize, mFixedSize;
    std::vector<Widget *> mChildren;
    bool mVisible, mEnabled;
    bool mFocused, mMouseFocus;
    std::string mTooltip;
    int mFontSize;
    Cursor mCursor;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- nanogui/object.h (*File object.h*)
- vector

Included By

- *File button.h*
- *File checkbox.h*
- *File colorwheel.h*
- *File glcanvas.h*
- *File graph.h*
- *File imagepanel.h*
- *File imageview.h*
- *File label.h*
- *File nanogui.h*
- *File progressbar.h*
- *File screen.h*
- *File core.h*
- *File slider.h*
- *File stackedwidget.h*
- *File tabheader.h*
- *File tabwidget.h*
- *File textbox.h*
- *File vscrollpanel.h*
- *File window.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Widget*

File window.h

Definition (`include/nanogui/window.h`)

Program Listing for File window.h

- Return to documentation for *File window.h*

```

/*
    nanogui/window.h -- Top-level window widget

    NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
    The widget drawing code is based on the NanoVG demo application
    by Mikko Mononen.

    All rights reserved. Use of this source code is governed by a
    BSD-style license that can be found in the LICENSE.txt file.
*/
#pragma once

#include <nanogui/widget.h>

NAMESPACE_BEGIN(nanogui)

class NANOGUI_EXPORT Window : public Widget {
    friend class Popup;
public:
    Window(Widget *parent, const std::string &title = "Untitled");

    const std::string &title() const { return mTitle; }
    void setTitle(const std::string &title) { mTitle = title; }

    bool modal() const { return mModal; }
    void setModal(bool modal) { mModal = modal; }

    Widget *buttonPanel();

    void dispose();

    void center();

    virtual void draw(NVGcontext *ctx) override;
    virtual bool mouseDragEvent(const Vector2i &p, const Vector2i &rel, int button,
↪int modifiers) override;
    virtual bool mouseButtonEvent(const Vector2i &p, int button, bool down, int_
↪modifiers) override;
    virtual bool scrollEvent(const Vector2i &p, const Vector2f &rel) override;
    virtual Vector2i preferredSize(NVGcontext *ctx) const override;
    virtual void performLayout(NVGcontext *ctx) override;
    virtual void save(Serializer &s) const override;
    virtual bool load(Serializer &s) override;
protected:
    virtual void refreshRelativePlacement();
protected:
    std::string mTitle;
    Widget *mButtonPanel;
    bool mModal;
    bool mDrag;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)

```

Includes

- `nanogui/widget.h` (*File widget.h*)

Included By

- *File messagedialog.h*
- *File nanogui.h*
- *File popup.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Window*

Contributing

Thank you for your interest in this project! Please refer to the following sections on how to contribute code and bug reports.

Reporting bugs

At the moment, this project is run in the spare time of a single person ([Wenzel Jakob](#)) with very limited resources for issue tracker tickets. Thus, before submitting a question or bug report, please take a moment of your time and ensure that your issue isn't already discussed in the project documentation elsewhere on this site.

Feature requests are generally closed unless they come with a pull request that implements the desired functionality.

Assuming that you have identified a previously unknown problem or an important question, it's essential that you submit a self-contained and minimal piece of code that reproduces the problem. In other words: no external dependencies, isolate the function(s) that cause breakage, submit matched and complete C++ or Python snippets (depending on how you are using NanoGUI) that can be easily compiled and run on my end.

Pull requests

Contributions are submitted, reviewed, and accepted using Github pull requests. Please refer to [this article](#) for details and adhere to the following rules to make the process as smooth as possible:

- Make a new branch for every feature you're working on.
- Make small and clean pull requests that are easy to review but make sure they do add value by themselves.
- Make sure you have tested any new functionality (e.g. if you made a new Widget).
- This project has a strong focus on providing general solutions using a minimal amount of code, thus small pull requests are greatly preferred.

- Read the remainder of this document, adhering to the bindings and documentation requirements.
- If making a purely documentation PR, please prefix the commit with `[docs]`
 - E.g. `[docs] Adding documentation for class X.`

Python Binding Requirements

Since NanoGUI builds for both C++, as well as produces Python bindings, you **must** account for both sides of the API regardless of how you use the project. If you are adding a new method, class, etc (not fixing an existing one), you must write the code to produce the relevant binding in `python/python.cpp`.

Code Style Requirements

- Tabs are 4 spaces – please do not submit PRs with *tab* characters.
- Most code follows an 80 column ruler, wherever reasonable.
- Pointers and references have modifiers next to variable name, not the type:
 - **Yes:** `void *p`, **No:** `void* p`
 - **Yes:** `Color &c`, **No:** `Color& c`
- Template classes / functions: `template <typename T> method()`
 - Space between template and `<`, on same line where reasonable
- Opening curly braces for definitions / loops / ifs are on the same line as the statement
 - **Yes:**

```
for (auto &&c : myVec) {
    // ... computation ...
}
```

- **No:**

```
for(auto &&c : myVec)
{
    // ... computation ...
}
```

Code Documentation Requirements

When adding new classes, methods, etc please provide meaningful and well formatted documentation of the new functionality. We use Doxygen comments to document the code, using the “JavaDoc” style. For consistency, please do not use the QT or other formats.

If you are familiar with how to use Doxygen-style comments:

- You should indent by four spaces for things like `param`, etc.
- `\brief`: a brief description.
- `\tparam`: a template parameter.
- `\param`: a parameter.
- `\return`: what the return value represents (where applicable).

For a quick crash-course on documenting using Doxygen:

1. If you are adding a new file, please include the disclaimer at the top **immediately** followed by `/** \file */`. So if you are creating the file `nanogui/file.h`

```
/*
  nanogui/file.h -- A brief description of what the file contains.

  NanoGUI was developed by Wenzel Jakob <wenzel.jakob@epfl.ch>.
  The widget drawing code is based on the NanoVG demo application
  by Mikko Mononen.

  All rights reserved. Use of this source code is governed by a
  BSD-style license that can be found in the LICENSE.txt file.
*/
/** \file */
```

changing the first line to the right name / description of your file.

2. Documenting a newly added Struct or Class requires special attention. If you are adding a class `Thing` in file `nanogui/thing.h`, the class level documentation needs to explicitly declare the location for Doxygen to parse everything correctly.

```
/**
 * \class Thing thing.h nanogui/thing.h
 *
 * This is the actual documentation for the thing.
 */
class Thing { ... };
```

This simply tells Doxygen how to format the various include directives. If you are writing a Struct, replace `\class` with `\struct`.

3. Please fully document all parameters, template parameters, and return types where applicable. In some cases it is sufficient to include just a brief one-line documentation string, e.g. the for the `nanogui::Screen::caption()` method, it is simple enough that the following is sufficient (note the **three** `///`):

```
/// Get the window title bar caption
const std::string &caption() const { return mCaption; }
```

However, more complicated methods should be thoroughly documented. As an example, this method demonstrates template parameters, parameters, and return value documentation:

```
/**
 * \brief A useless function for getting sizes.
 *
 * This method has specific things that must be pointed out, but they
 * were too long to include in the 'brief' documentation.
 *
 * \tparam T
 *     The type we are evaluating the size of.
 *
 * \param returnFake
 *     If set to true, a random positive number will be returned. This
 *     comment is a bit longer and can span multiple lines, making sure
 *     to indent each new line.
 *
 *     Warning: this had an empty line before it and will NOT appear in
```

```

*   the documentation of this parameter, but instead it will appear
*   in the documentation of the method!
*
* \return
*   The result of ``sizeof(T)``.
*/
template <typename T>
size_t exampleTemplateFunction(bool returnFake = false) { ... }

```

Styling the Code

Since we are using both Doxygen and Sphinx, we have access to a wealth of interesting documentation styling.

From Doxygen You can use things like `\throws`, `\remark`, and even `\ref` to generate html links to other items.

From Sphinx On the Sphinx side, you now have access to full reStructuredText syntax. This includes:

- `**bold**` to make **bold** text
- `*italics*` for *italics*
- ```teletype``` for teletype text.

You can additionally include more complex reStructuredText such as grid tables, as well as Sphinx directives. You will need to use the `\rst` and `\endrst` commands for these:

```

/**
 * \brief Some method you are documenting.
 *
 * \rst
 * I am now in a verbatim reStructuredText environment, and can create a
↪grid table.
 *
 * I could create a python code listing using
 *
 * .. code-block:: py
 *
 *     print("Some python code.")
 *
 * You can also use the note or warning directives to highlight
↪important concepts:
 *
 * .. note::
 *     You may or may not segfault.
 *
 * .. warning::
 *     I guarantee you will segfault.
 * \endrst
 */

```

Warning: In normal reStructuredText, if you simply indent a block of code by four spaces it will render as a code listing. While this will build as expected for the C++ documentation on RTD, it will **fail** to build `py_doc.h` correctly.

For code listings, **always** begin an `\rst` section and use `.. code-block` as shown above.

TODO

Documentation Completion

Already familiar with NanoGUI or a subset of its classes? The documentation for the following files is incomplete, waiting for your PR. Document a whole class, or even just a method of a given class.

If you make progress on / complete an item with your PR, please update / remove it from the table on this page ([docs/contributing.rst](#)).

Note: The NanoGUI documentation hosted online does not include `private` methods or member variables at this time. However, documentation for these is welcome!

Warning: In some of these files, you will see preprocessor blocks like

```
#ifndef DOXYGEN_SHOULD_SKIP_THIS
... code that the breaks the documentation ...
#endif // DOXYGEN_SHOULD_SKIP_THIS
```

Please take care not to remove these!

Filename	Action Item
button.h	<ul style="list-style-type: none"> • Most member methods. • All member variables.
checkbox.h	<ul style="list-style-type: none"> • All member methods and variables.
colorpicker.h	<ul style="list-style-type: none"> • Constructor and callback. • All member variables.
colorwheel.h	<ul style="list-style-type: none"> • Most methods and member variables.
combobox.h	<ul style="list-style-type: none"> • Most member methods and variables.
formhelper.h	<ul style="list-style-type: none"> • More detailed documentation explaining parameters for <code>FormHelper</code> methods. • Most member variables.
graph.h	<ul style="list-style-type: none"> • All member methods and variables.
imagepanel.h	<ul style="list-style-type: none"> • All member methods and variables.
imageview.h	<ul style="list-style-type: none"> • Most member methods.
label.h	<ul style="list-style-type: none"> • Some member methods and variables.
layout.h	<ul style="list-style-type: none"> • Nearly everything.
popup.h	<ul style="list-style-type: none"> • Some member methods and variables. • Explicit parameter documentation would be very useful.
popupbutton.h	<ul style="list-style-type: none"> • Almost everything.
progressbar.h	<ul style="list-style-type: none"> • Almost everything.
screen.h	<ul style="list-style-type: none"> • Documentation for the manual GLFW API. • All member variables.
slider.h	<ul style="list-style-type: none"> • Almost everything.
stackedwidget.h	<ul style="list-style-type: none"> • Almost everything.
tabheader.h	<ul style="list-style-type: none"> • Some member methods. • Some reformatting of existing documentation to use <code>\param</code> or <code>\return</code> etc.
5.5. Contributing	197
tabwidget.h	<ul style="list-style-type: none"> • Some member methods. • Some reformatting of existing documentation to use <code>\param</code> or <code>\return</code> etc.

Advanced Contribution Opportunity

Currently, all partial and full template specializations are skipped. Specifically, nearly everything in `include/nanogui/serializer/*.h`. According to the [Breathe documentation](#) this should be possible. The likely cause of this issue is that the version of Breathe packaged for use with `pip` is not up to date. Your task would be to find a way to use `docs/requirements.txt` to install the **current source** from the master branch of Breathe instead of using PyPi.

You can test locally by making sure you do not have Breathe installed with `pip`, and compiling it yourself (make sure you add it to your `PATH` so you can use it in Python).

Then try moving the `#ifndef DOXYGEN_SHOULD_SKIP_THIS` to expose a single template specialization in a file of your choice, and try and get the documentation to build. If you succeed with this, the next step will be to find a way to get Read the Docs to build the current source of Breathe rather than using PyPi.

In theory, all of these are possible.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

G

GL_HALF_FLOAT (C macro), 86

N

NAMESPACE_BEGIN (C macro), 87

NAMESPACE_END (C macro), 87

nanogui::__nanogui_get_image (C++ function), 82

nanogui::active (C++ function), 82

nanogui::AdvancedGridLayout (C++ class), 22

nanogui::AdvancedGridLayout::AdvancedGridLayout
(C++ function), 22

nanogui::AdvancedGridLayout::Anchor (C++ class), 19,
23

nanogui::AdvancedGridLayout::anchor (C++ function),
23

nanogui::AdvancedGridLayout::Anchor::align (C++
member), 20, 24

nanogui::AdvancedGridLayout::Anchor::Anchor (C++
function), 19, 23

nanogui::AdvancedGridLayout::Anchor::operator
std::string (C++ function), 19, 23

nanogui::AdvancedGridLayout::Anchor::pos (C++ mem-
ber), 20, 24

nanogui::AdvancedGridLayout::Anchor::size (C++ mem-
ber), 20, 24

nanogui::AdvancedGridLayout::appendCol (C++ func-
tion), 23

nanogui::AdvancedGridLayout::appendRow (C++ func-
tion), 22

nanogui::AdvancedGridLayout::colCount (C++ func-
tion), 22

nanogui::AdvancedGridLayout::computeLayout (C++
function), 23

nanogui::AdvancedGridLayout::mAnchor (C++ mem-
ber), 23

nanogui::AdvancedGridLayout::margin (C++ function),
22

nanogui::AdvancedGridLayout::mCols (C++ member),
23

nanogui::AdvancedGridLayout::mColStretch (C++ mem-
ber), 23

nanogui::AdvancedGridLayout::mMargin (C++ mem-
ber), 23

nanogui::AdvancedGridLayout::mRows (C++ member),
23

nanogui::AdvancedGridLayout::mRowStretch (C++
member), 23

nanogui::AdvancedGridLayout::performLayout (C++
function), 23

nanogui::AdvancedGridLayout::preferredSize (C++
function), 23

nanogui::AdvancedGridLayout::rowCount (C++ func-
tion), 22

nanogui::AdvancedGridLayout::setAnchor (C++ func-
tion), 23

nanogui::AdvancedGridLayout::setColStretch (C++
function), 23

nanogui::AdvancedGridLayout::setMargin (C++ func-
tion), 22

nanogui::AdvancedGridLayout::setRowStretch (C++
function), 23

nanogui::Alignment (C++ type), 80

nanogui::Arcball (C++ class), 20

nanogui::Arcball::active (C++ function), 20

nanogui::Arcball::Arcball (C++ function), 20

nanogui::Arcball::button (C++ function), 20

nanogui::Arcball::mActive (C++ member), 20

nanogui::Arcball::matrix (C++ function), 20

nanogui::Arcball::mIncr (C++ member), 20

nanogui::Arcball::mLastPos (C++ member), 20

nanogui::Arcball::motion (C++ function), 20

nanogui::Arcball::mQuat (C++ member), 20

nanogui::Arcball::mSize (C++ member), 20

nanogui::Arcball::mSpeedFactor (C++ member), 20

nanogui::Arcball::setSize (C++ function), 20

nanogui::Arcball::setSpeedFactor (C++ function), 20

nanogui::Arcball::setState (C++ function), 20

nanogui::Arcball::size (C++ function), 20

nanogui::Arcball::speedFactor (C++ function), 20

- nanogui::Arcball::state (C++ function), 20
- nanogui::Arrow (C++ class), 80
- nanogui::BoxLayout (C++ class), 24
- nanogui::BoxLayout::alignment (C++ function), 24
- nanogui::BoxLayout::BoxLayout (C++ function), 24
- nanogui::BoxLayout::mAlignment (C++ member), 25
- nanogui::BoxLayout::margin (C++ function), 24
- nanogui::BoxLayout::mMargin (C++ member), 25
- nanogui::BoxLayout::mOrientation (C++ member), 25
- nanogui::BoxLayout::mSpacing (C++ member), 25
- nanogui::BoxLayout::orientation (C++ function), 24
- nanogui::BoxLayout::performLayout (C++ function), 24
- nanogui::BoxLayout::preferredSize (C++ function), 24
- nanogui::BoxLayout::setAlignment (C++ function), 24
- nanogui::BoxLayout::setMargin (C++ function), 24
- nanogui::BoxLayout::setOrientation (C++ function), 24
- nanogui::BoxLayout::setSpacing (C++ function), 24
- nanogui::BoxLayout::spacing (C++ function), 24
- nanogui::Button (C++ class), 25
- nanogui::Button::backgroundColor (C++ function), 26
- nanogui::Button::Button (C++ function), 25
- nanogui::Button::buttonGroup (C++ function), 26
- nanogui::Button::callback (C++ function), 26
- nanogui::Button::caption (C++ function), 25
- nanogui::Button::changeCallback (C++ function), 26
- nanogui::Button::draw (C++ function), 26
- nanogui::Button::flags (C++ function), 26
- nanogui::Button::Flags (C++ type), 25, 79
- nanogui::Button::icon (C++ function), 26
- nanogui::Button::iconPosition (C++ function), 26
- nanogui::Button::IconPosition (C++ type), 25, 79
- nanogui::Button::Left (C++ class), 25, 79
- nanogui::Button::LeftCentered (C++ class), 25, 79
- nanogui::Button::load (C++ function), 26
- nanogui::Button::mBackgroundColor (C++ member), 27
- nanogui::Button::mButtonGroup (C++ member), 27
- nanogui::Button::mCallback (C++ member), 27
- nanogui::Button::mCaption (C++ member), 27
- nanogui::Button::mChangeCallback (C++ member), 27
- nanogui::Button::mFlags (C++ member), 27
- nanogui::Button::mIcon (C++ member), 27
- nanogui::Button::mIconPosition (C++ member), 27
- nanogui::Button::mouseButtonEvent (C++ function), 26
- nanogui::Button::mPushed (C++ member), 27
- nanogui::Button::mTextColor (C++ member), 27
- nanogui::Button::NormalButton (C++ class), 25, 79
- nanogui::Button::PopupButton (C++ class), 25, 79
- nanogui::Button::preferredSize (C++ function), 26
- nanogui::Button::pushed (C++ function), 26
- nanogui::Button::RadioButton (C++ class), 25, 79
- nanogui::Button::Right (C++ class), 25, 80
- nanogui::Button::RightCentered (C++ class), 25, 79
- nanogui::Button::save (C++ function), 26
- nanogui::Button::setBackgroundColor (C++ function), 26
- nanogui::Button::setButtonGroup (C++ function), 26
- nanogui::Button::setCallback (C++ function), 26
- nanogui::Button::setCaption (C++ function), 25
- nanogui::Button::setChangeCallback (C++ function), 26
- nanogui::Button::setFlags (C++ function), 26
- nanogui::Button::setIcon (C++ function), 26
- nanogui::Button::setIconPosition (C++ function), 26
- nanogui::Button::setPushed (C++ function), 26
- nanogui::Button::setTextColor (C++ function), 26
- nanogui::Button::textColor (C++ function), 26
- nanogui::Button::ToggleButton (C++ class), 25, 79
- nanogui::chdir_to_bundle_parent (C++ function), 82
- nanogui::CheckBox (C++ class), 27
- nanogui::CheckBox::callback (C++ function), 27
- nanogui::CheckBox::caption (C++ function), 27
- nanogui::CheckBox::CheckBox (C++ function), 27
- nanogui::CheckBox::checked (C++ function), 27
- nanogui::CheckBox::draw (C++ function), 27
- nanogui::CheckBox::load (C++ function), 28
- nanogui::CheckBox::mCallback (C++ member), 28
- nanogui::CheckBox::mCaption (C++ member), 28
- nanogui::CheckBox::mChecked (C++ member), 28
- nanogui::CheckBox::mouseButtonEvent (C++ function), 27
- nanogui::CheckBox::mPushed (C++ member), 28
- nanogui::CheckBox::preferredSize (C++ function), 27
- nanogui::CheckBox::pushed (C++ function), 27
- nanogui::CheckBox::save (C++ function), 28
- nanogui::CheckBox::setCallback (C++ function), 27
- nanogui::CheckBox::setCaption (C++ function), 27
- nanogui::CheckBox::setChecked (C++ function), 27
- nanogui::CheckBox::setPushed (C++ function), 27
- nanogui::Color (C++ class), 28
- nanogui::Color::b (C++ function), 30
- nanogui::Color::Color (C++ function), 28–30
- nanogui::Color::contrastingColor (C++ function), 30
- nanogui::Color::g (C++ function), 30
- nanogui::Color::operator const NVGcolor& (C++ function), 31
- nanogui::Color::operator= (C++ function), 30
- nanogui::Color::r (C++ function), 30
- nanogui::ColorPicker (C++ class), 31
- nanogui::ColorPicker::callback (C++ function), 31
- nanogui::ColorPicker::color (C++ function), 31
- nanogui::ColorPicker::ColorPicker (C++ function), 31
- nanogui::ColorPicker::mCallback (C++ member), 31
- nanogui::ColorPicker::mColorWheel (C++ member), 31
- nanogui::ColorPicker::mPickButton (C++ member), 31
- nanogui::ColorPicker::setCallback (C++ function), 31
- nanogui::ColorPicker::setColor (C++ function), 31
- nanogui::ColorWheel (C++ class), 31
- nanogui::ColorWheel::Both (C++ class), 81
- nanogui::ColorWheel::callback (C++ function), 32
- nanogui::ColorWheel::color (C++ function), 32

- nanogui::ColorWheel::ColorWheel (C++ function), 32
- nanogui::ColorWheel::draw (C++ function), 32
- nanogui::ColorWheel::InnerTriangle (C++ class), 81
- nanogui::ColorWheel::load (C++ function), 32
- nanogui::ColorWheel::mBlack (C++ member), 32
- nanogui::ColorWheel::mCallback (C++ member), 32
- nanogui::ColorWheel::mDragRegion (C++ member), 32
- nanogui::ColorWheel::mHue (C++ member), 32
- nanogui::ColorWheel::mouseButtonEvent (C++ function), 32
- nanogui::ColorWheel::mouseDragEvent (C++ function), 32
- nanogui::ColorWheel::mWhite (C++ member), 32
- nanogui::ColorWheel::None (C++ class), 81
- nanogui::ColorWheel::OuterCircle (C++ class), 81
- nanogui::ColorWheel::preferredSize (C++ function), 32
- nanogui::ColorWheel::Region (C++ type), 81
- nanogui::ColorWheel::save (C++ function), 32
- nanogui::ColorWheel::setCallback (C++ function), 32
- nanogui::ColorWheel::setColor (C++ function), 32
- nanogui::ComboBox (C++ class), 32
- nanogui::ComboBox::callback (C++ function), 33
- nanogui::ComboBox::ComboBox (C++ function), 33
- nanogui::ComboBox::items (C++ function), 33
- nanogui::ComboBox::itemsShort (C++ function), 33
- nanogui::ComboBox::load (C++ function), 33
- nanogui::ComboBox::mCallback (C++ member), 33
- nanogui::ComboBox::mItems (C++ member), 33
- nanogui::ComboBox::mItemsShort (C++ member), 33
- nanogui::ComboBox::mSelectedIndex (C++ member), 33
- nanogui::ComboBox::save (C++ function), 33
- nanogui::ComboBox::scrollEvent (C++ function), 33
- nanogui::ComboBox::selectedIndex (C++ function), 33
- nanogui::ComboBox::setCallback (C++ function), 33
- nanogui::ComboBox::setItems (C++ function), 33
- nanogui::ComboBox::setSelectedIndex (C++ function), 33
- nanogui::Crosshair (C++ class), 80
- nanogui::Cursor (C++ type), 80
- nanogui::CursorCount (C++ class), 80
- nanogui::detail::FormWidget (C++ class), 33
- nanogui::detail::serialization_helper (C++ class), 21
- nanogui::detail::serialization_traits (C++ class), 21
- nanogui::file_dialog (C++ function), 82
- nanogui::Fill (C++ class), 80
- nanogui::FloatBox (C++ class), 34
- nanogui::FloatBox::FloatBox (C++ function), 34
- nanogui::FloatBox::mouseButtonEvent (C++ function), 35
- nanogui::FloatBox::mouseDragEvent (C++ function), 35
- nanogui::FloatBox::numberFormat (C++ function), 34
- nanogui::FloatBox::scrollEvent (C++ function), 35
- nanogui::FloatBox::setCallback (C++ function), 35
- nanogui::FloatBox::setMaxValue (C++ function), 35
- nanogui::FloatBox::setMinMaxValues (C++ function), 35
- nanogui::FloatBox::setMinValue (C++ function), 35
- nanogui::FloatBox::setValue (C++ function), 35
- nanogui::FloatBox::setValueIncrement (C++ function), 35
- nanogui::FloatBox::value (C++ function), 35
- nanogui::FormHelper (C++ class), 35
- nanogui::FormHelper::addButton (C++ function), 36
- nanogui::FormHelper::addGroup (C++ function), 36
- nanogui::FormHelper::addVariable (C++ function), 36
- nanogui::FormHelper::addWidget (C++ function), 36
- nanogui::FormHelper::addWindow (C++ function), 36
- nanogui::FormHelper::fixedSize (C++ function), 36
- nanogui::FormHelper::FormHelper (C++ function), 36
- nanogui::FormHelper::groupFontName (C++ function), 36
- nanogui::FormHelper::groupFontSize (C++ function), 36
- nanogui::FormHelper::labelFontName (C++ function), 36
- nanogui::FormHelper::labelFontSize (C++ function), 36
- nanogui::FormHelper::mFixedSize (C++ member), 37
- nanogui::FormHelper::mGroupFontName (C++ member), 37
- nanogui::FormHelper::mGroupFontSize (C++ member), 37
- nanogui::FormHelper::mLabelFontName (C++ member), 37
- nanogui::FormHelper::mLabelFontSize (C++ member), 37
- nanogui::FormHelper::mLayout (C++ member), 37
- nanogui::FormHelper::mPostGroupSpacing (C++ member), 37
- nanogui::FormHelper::mPreGroupSpacing (C++ member), 37
- nanogui::FormHelper::mRefreshCallbacks (C++ member), 37
- nanogui::FormHelper::mScreen (C++ member), 37
- nanogui::FormHelper::mVariableSpacing (C++ member), 37
- nanogui::FormHelper::mWidgetFontSize (C++ member), 37
- nanogui::FormHelper::mWindow (C++ member), 37
- nanogui::FormHelper::refresh (C++ function), 36
- nanogui::FormHelper::setFixedSize (C++ function), 36
- nanogui::FormHelper::setGroupFontName (C++ function), 36
- nanogui::FormHelper::setGroupFontSize (C++ function), 36
- nanogui::FormHelper::setLabelFontName (C++ function), 36
- nanogui::FormHelper::setLabelFontSize (C++ function), 36
- nanogui::FormHelper::setWidgetFontSize (C++ function), 36
- nanogui::FormHelper::setWindow (C++ function), 36

nanogui::FormHelper::widgetFontSize (C++ function), 36

nanogui::FormHelper::window (C++ function), 36

nanogui::frustum (C++ function), 82

nanogui::GLCanvas (C++ class), 37

nanogui::GLCanvas::backgroundColor (C++ function), 37

nanogui::GLCanvas::draw (C++ function), 37

nanogui::GLCanvas::drawBorder (C++ function), 37

nanogui::GLCanvas::drawGL (C++ function), 38

nanogui::GLCanvas::drawWidgetBorder (C++ function), 38

nanogui::GLCanvas::GLCanvas (C++ function), 37

nanogui::GLCanvas::load (C++ function), 38

nanogui::GLCanvas::mBackgroundColor (C++ member), 38

nanogui::GLCanvas::mDrawBorder (C++ member), 38

nanogui::GLCanvas::save (C++ function), 38

nanogui::GLCanvas::setBackgroundColor (C++ function), 37

nanogui::GLCanvas::setDrawBorder (C++ function), 37

nanogui::GLFramebuffer (C++ class), 38

nanogui::GLFramebuffer::bind (C++ function), 38

nanogui::GLFramebuffer::blit (C++ function), 38

nanogui::GLFramebuffer::downloadTGA (C++ function), 38

nanogui::GLFramebuffer::free (C++ function), 38

nanogui::GLFramebuffer::GLFramebuffer (C++ function), 38

nanogui::GLFramebuffer::init (C++ function), 38

nanogui::GLFramebuffer::mColor (C++ member), 39

nanogui::GLFramebuffer::mDepth (C++ member), 39

nanogui::GLFramebuffer::mFramebuffer (C++ member), 39

nanogui::GLFramebuffer::mSamples (C++ member), 39

nanogui::GLFramebuffer::mSize (C++ member), 39

nanogui::GLFramebuffer::ready (C++ function), 38

nanogui::GLFramebuffer::release (C++ function), 38

nanogui::GLFramebuffer::samples (C++ function), 38

nanogui::GLShader (C++ class), 39

nanogui::GLShader::attrib (C++ function), 40

nanogui::GLShader::attribVersion (C++ function), 40

nanogui::GLShader::bind (C++ function), 40

nanogui::GLShader::Buffer (C++ class), 21, 41

nanogui::GLShader::Buffer::compSize (C++ member), 21, 42

nanogui::GLShader::Buffer::dim (C++ member), 21, 42

nanogui::GLShader::Buffer::glType (C++ member), 21, 42

nanogui::GLShader::Buffer::id (C++ member), 21, 42

nanogui::GLShader::Buffer::size (C++ member), 21, 42

nanogui::GLShader::Buffer::version (C++ member), 21, 42

nanogui::GLShader::bufferSize (C++ function), 41

nanogui::GLShader::define (C++ function), 40

nanogui::GLShader::downloadAttrib (C++ function), 40, 41

nanogui::GLShader::drawArray (C++ function), 40

nanogui::GLShader::drawIndexed (C++ function), 40

nanogui::GLShader::free (C++ function), 40

nanogui::GLShader::freeAttrib (C++ function), 40

nanogui::GLShader::GLShader (C++ function), 39

nanogui::GLShader::hasAttrib (C++ function), 40

nanogui::GLShader::init (C++ function), 39

nanogui::GLShader::initFromFiles (C++ function), 39

nanogui::GLShader::invalidateAttribs (C++ function), 40

nanogui::GLShader::mBufferObjects (C++ member), 41

nanogui::GLShader::mDefinitions (C++ member), 41

nanogui::GLShader::mFragmentShader (C++ member), 41

nanogui::GLShader::mGeometryShader (C++ member), 41

nanogui::GLShader::mName (C++ member), 41

nanogui::GLShader::mProgramShader (C++ member), 41

nanogui::GLShader::mVertexArrayObject (C++ member), 41

nanogui::GLShader::mVertexShader (C++ member), 41

nanogui::GLShader::name (C++ function), 40

nanogui::GLShader::resetAttribVersion (C++ function), 40

nanogui::GLShader::setUniform (C++ function), 40, 41

nanogui::GLShader::shareAttrib (C++ function), 40

nanogui::GLShader::uniform (C++ function), 40

nanogui::GLShader::uploadAttrib (C++ function), 40, 41

nanogui::GLShader::uploadIndices (C++ function), 40

nanogui::GLUniformBuffer (C++ class), 42

nanogui::GLUniformBuffer::bind (C++ function), 42

nanogui::GLUniformBuffer::free (C++ function), 42

nanogui::GLUniformBuffer::getBindingPoint (C++ function), 42

nanogui::GLUniformBuffer::GLUniformBuffer (C++ function), 42

nanogui::GLUniformBuffer::init (C++ function), 42

nanogui::GLUniformBuffer::release (C++ function), 42

nanogui::GLUniformBuffer::update (C++ function), 42

nanogui::Graph (C++ class), 42

nanogui::Graph::backgroundColor (C++ function), 43

nanogui::Graph::caption (C++ function), 43

nanogui::Graph::draw (C++ function), 43

nanogui::Graph::footer (C++ function), 43

nanogui::Graph::foregroundColor (C++ function), 43

nanogui::Graph::Graph (C++ function), 43

nanogui::Graph::header (C++ function), 43

nanogui::Graph::load (C++ function), 43

nanogui::Graph::mBackgroundColor (C++ member), 43

nanogui::Graph::mCaption (C++ member), 43

nanogui::Graph::mFooter (C++ member), 43

- nanogui::Graph::mForegroundColor (C++ member), 43
- nanogui::Graph::mHeader (C++ member), 43
- nanogui::Graph::mTextColor (C++ member), 43
- nanogui::Graph::mValues (C++ member), 43
- nanogui::Graph::preferredSize (C++ function), 43
- nanogui::Graph::save (C++ function), 43
- nanogui::Graph::setBackgroundColor (C++ function), 43
- nanogui::Graph::setCaption (C++ function), 43
- nanogui::Graph::setFooter (C++ function), 43
- nanogui::Graph::setForegroundColor (C++ function), 43
- nanogui::Graph::setHeader (C++ function), 43
- nanogui::Graph::setTextColor (C++ function), 43
- nanogui::Graph::setValues (C++ function), 43
- nanogui::Graph::textColor (C++ function), 43
- nanogui::Graph::values (C++ function), 43
- nanogui::GridLayout (C++ class), 44
- nanogui::GridLayout::alignment (C++ function), 44
- nanogui::GridLayout::computeLayout (C++ function), 44
- nanogui::GridLayout::GridLayout (C++ function), 44
- nanogui::GridLayout::mAlignment (C++ member), 45
- nanogui::GridLayout::margin (C++ function), 44
- nanogui::GridLayout::mDefaultAlignment (C++ member), 45
- nanogui::GridLayout::mMargin (C++ member), 45
- nanogui::GridLayout::mOrientation (C++ member), 45
- nanogui::GridLayout::mResolution (C++ member), 45
- nanogui::GridLayout::mSpacing (C++ member), 45
- nanogui::GridLayout::orientation (C++ function), 44
- nanogui::GridLayout::performLayout (C++ function), 44
- nanogui::GridLayout::preferredSize (C++ function), 44
- nanogui::GridLayout::resolution (C++ function), 44
- nanogui::GridLayout::setColAlignment (C++ function), 44
- nanogui::GridLayout::setMargin (C++ function), 44
- nanogui::GridLayout::setOrientation (C++ function), 44
- nanogui::GridLayout::setResolution (C++ function), 44
- nanogui::GridLayout::setRowAlignment (C++ function), 44
- nanogui::GridLayout::setSpacing (C++ function), 44
- nanogui::GridLayout::spacing (C++ function), 44
- nanogui::GroupLayout (C++ class), 45
- nanogui::GroupLayout::groupIndent (C++ function), 45
- nanogui::GroupLayout::GroupLayout (C++ function), 45
- nanogui::GroupLayout::groupSpacing (C++ function), 45
- nanogui::GroupLayout::margin (C++ function), 45
- nanogui::GroupLayout::mGroupIndent (C++ member), 46
- nanogui::GroupLayout::mGroupSpacing (C++ member), 46
- nanogui::GroupLayout::mMargin (C++ member), 46
- nanogui::GroupLayout::mSpacing (C++ member), 46
- nanogui::GroupLayout::performLayout (C++ function), 45
- nanogui::GroupLayout::preferredSize (C++ function), 45
- nanogui::GroupLayout::setGroupIndent (C++ function), 45
- nanogui::GroupLayout::setGroupSpacing (C++ function), 45
- nanogui::GroupLayout::setMargin (C++ function), 45
- nanogui::GroupLayout::setSpacing (C++ function), 45
- nanogui::GroupLayout::spacing (C++ function), 45
- nanogui::Hand (C++ class), 80
- nanogui::Horizontal (C++ class), 80
- nanogui::HResize (C++ class), 80
- nanogui::IBeam (C++ class), 80
- nanogui::ImagePanel (C++ class), 46
- nanogui::ImagePanel::callback (C++ function), 46
- nanogui::ImagePanel::draw (C++ function), 46
- nanogui::ImagePanel::gridSize (C++ function), 46
- nanogui::ImagePanel::ImagePanel (C++ function), 46
- nanogui::ImagePanel::images (C++ function), 46
- nanogui::ImagePanel::Images (C++ type), 46
- nanogui::ImagePanel::indexOfPosition (C++ function), 46
- nanogui::ImagePanel::mCallback (C++ member), 47
- nanogui::ImagePanel::mImages (C++ member), 47
- nanogui::ImagePanel::mMargin (C++ member), 47
- nanogui::ImagePanel::mMouseIndex (C++ member), 47
- nanogui::ImagePanel::mouseButtonEvent (C++ function), 46
- nanogui::ImagePanel::mouseMotionEvent (C++ function), 46
- nanogui::ImagePanel::mSpacing (C++ member), 47
- nanogui::ImagePanel::mThumbSize (C++ member), 47
- nanogui::ImagePanel::preferredSize (C++ function), 46
- nanogui::ImagePanel::setCallback (C++ function), 46
- nanogui::ImagePanel::setImages (C++ function), 46
- nanogui::ImageView (C++ class), 47
- nanogui::ImageView::~~ImageView (C++ function), 47
- nanogui::ImageView::bindImage (C++ function), 47
- nanogui::ImageView::center (C++ function), 48
- nanogui::ImageView::clampedImageCoordinateAt (C++ function), 48
- nanogui::ImageView::draw (C++ function), 49
- nanogui::ImageView::fit (C++ function), 48
- nanogui::ImageView::fixedOffset (C++ function), 47
- nanogui::ImageView::fixedScale (C++ function), 47
- nanogui::ImageView::fontScaleFactor (C++ function), 48
- nanogui::ImageView::gridThreshold (C++ function), 48
- nanogui::ImageView::gridVisible (C++ function), 49
- nanogui::ImageView::helpersVisible (C++ function), 49
- nanogui::ImageView::imageCoordinateAt (C++ function), 48
- nanogui::ImageView::imageShader (C++ function), 47
- nanogui::ImageView::imageSize (C++ function), 47
- nanogui::ImageView::imageSizeF (C++ function), 47
- nanogui::ImageView::ImageView (C++ function), 47

nanogui::ImageView::keyboardCharacterEvent (C++ function), 48
 nanogui::ImageView::keyboardEvent (C++ function), 48
 nanogui::ImageView::mouseDragEvent (C++ function), 48
 nanogui::ImageView::moveOffset (C++ function), 48
 nanogui::ImageView::offset (C++ function), 47
 nanogui::ImageView::performLayout (C++ function), 49
 nanogui::ImageView::pixelInfoThreshold (C++ function), 48
 nanogui::ImageView::pixelInfoVisible (C++ function), 49
 nanogui::ImageView::positionF (C++ function), 47
 nanogui::ImageView::positionForCoordinate (C++ function), 48
 nanogui::ImageView::preferredSize (C++ function), 49
 nanogui::ImageView::scale (C++ function), 47
 nanogui::ImageView::scaledImageSize (C++ function), 47
 nanogui::ImageView::scaledImageSizeF (C++ function), 47
 nanogui::ImageView::scrollEvent (C++ function), 48
 nanogui::ImageView::setFixedOffset (C++ function), 47
 nanogui::ImageView::setFixedScale (C++ function), 47
 nanogui::ImageView::setFontScaleFactor (C++ function), 48
 nanogui::ImageView::setGridThreshold (C++ function), 48
 nanogui::ImageView::setImageCoordinateAt (C++ function), 48
 nanogui::ImageView::setOffset (C++ function), 47
 nanogui::ImageView::setPixelInfoThreshold (C++ function), 48
 nanogui::ImageView::setScale (C++ function), 47
 nanogui::ImageView::setScaleCentered (C++ function), 48
 nanogui::ImageView::setZoomSensitivity (C++ function), 48
 nanogui::ImageView::sizeF (C++ function), 47
 nanogui::ImageView::zoom (C++ function), 48
 nanogui::ImageView::zoomSensitivity (C++ function), 48
 nanogui::init (C++ function), 83
 nanogui::IntBox (C++ class), 49
 nanogui::IntBox::IntBox (C++ function), 49
 nanogui::IntBox::mouseButtonEvent (C++ function), 49
 nanogui::IntBox::mouseDragEvent (C++ function), 49
 nanogui::IntBox::scrollEvent (C++ function), 49
 nanogui::IntBox::setCallback (C++ function), 49
 nanogui::IntBox::setMaxValue (C++ function), 49
 nanogui::IntBox::setMinMaxValues (C++ function), 49
 nanogui::IntBox::setMinValue (C++ function), 49
 nanogui::IntBox::setValue (C++ function), 49
 nanogui::IntBox::setValueIncrement (C++ function), 49
 nanogui::IntBox::value (C++ function), 49
 nanogui::Label (C++ class), 50
 nanogui::Label::caption (C++ function), 50
 nanogui::Label::color (C++ function), 50
 nanogui::Label::draw (C++ function), 50
 nanogui::Label::font (C++ function), 50
 nanogui::Label::Label (C++ function), 50
 nanogui::Label::load (C++ function), 50
 nanogui::Label::mCaption (C++ member), 50
 nanogui::Label::mColor (C++ member), 50
 nanogui::Label::mFont (C++ member), 50
 nanogui::Label::preferredSize (C++ function), 50
 nanogui::Label::save (C++ function), 50
 nanogui::Label::setCaption (C++ function), 50
 nanogui::Label::setColor (C++ function), 50
 nanogui::Label::setFont (C++ function), 50
 nanogui::Label::setTheme (C++ function), 50
 nanogui::Layout (C++ class), 51
 nanogui::Layout::~~Layout (C++ function), 51
 nanogui::Layout::performLayout (C++ function), 51
 nanogui::Layout::preferredSize (C++ function), 51
 nanogui::leave (C++ function), 83
 nanogui::loadImageDirectory (C++ function), 83
 nanogui::lookAt (C++ function), 83
 nanogui::mainloop (C++ function), 84
 nanogui::MatrixXu (C++ type), 89
 nanogui::Maximum (C++ class), 80
 nanogui::MessageDialog (C++ class), 51
 nanogui::MessageDialog::callback (C++ function), 52
 nanogui::MessageDialog::Information (C++ class), 51, 81
 nanogui::MessageDialog::mCallback (C++ member), 52
 nanogui::MessageDialog::MessageDialog (C++ function), 51
 nanogui::MessageDialog::messageLabel (C++ function), 51, 52
 nanogui::MessageDialog::mMessageLabel (C++ member), 52
 nanogui::MessageDialog::Question (C++ class), 51, 81
 nanogui::MessageDialog::setCallback (C++ function), 52
 nanogui::MessageDialog::Type (C++ type), 51, 81
 nanogui::MessageDialog::Warning (C++ class), 51, 81
 nanogui::Middle (C++ class), 80
 nanogui::Minimum (C++ class), 80
 nanogui::nvglSetFontIcon (C++ function), 84
 nanogui::nvglsetImageIcon (C++ function), 84
 nanogui::Object (C++ class), 52
 nanogui::Object::~~Object (C++ function), 52
 nanogui::Object::decRef (C++ function), 52
 nanogui::Object::getRefCount (C++ function), 52
 nanogui::Object::incRef (C++ function), 52
 nanogui::Object::Object (C++ function), 52
 nanogui::Orientation (C++ type), 80
 nanogui::ortho (C++ function), 84

- nanogui::Popup (C++ class), 53
- nanogui::Popup::anchorHeight (C++ function), 53
- nanogui::Popup::anchorPos (C++ function), 53
- nanogui::Popup::draw (C++ function), 53
- nanogui::Popup::Left (C++ class), 53, 81
- nanogui::Popup::load (C++ function), 54
- nanogui::Popup::mAnchorHeight (C++ member), 54
- nanogui::Popup::mAnchorPos (C++ member), 54
- nanogui::Popup::mParentWindow (C++ member), 54
- nanogui::Popup::mSide (C++ member), 54
- nanogui::Popup::parentWindow (C++ function), 53
- nanogui::Popup::performLayout (C++ function), 53
- nanogui::Popup::Popup (C++ function), 53
- nanogui::Popup::refreshRelativePlacement (C++ function), 54
- nanogui::Popup::Right (C++ class), 53, 81
- nanogui::Popup::save (C++ function), 54
- nanogui::Popup::setAnchorHeight (C++ function), 53
- nanogui::Popup::setAnchorPos (C++ function), 53
- nanogui::Popup::setSide (C++ function), 53
- nanogui::Popup::side (C++ function), 53
- nanogui::Popup::Side (C++ type), 53, 81
- nanogui::PopupButton (C++ class), 54
- nanogui::PopupButton::chevronIcon (C++ function), 54
- nanogui::PopupButton::draw (C++ function), 54
- nanogui::PopupButton::load (C++ function), 55
- nanogui::PopupButton::mChevronIcon (C++ member), 55
- nanogui::PopupButton::mPopup (C++ member), 55
- nanogui::PopupButton::performLayout (C++ function), 55
- nanogui::PopupButton::popup (C++ function), 54
- nanogui::PopupButton::PopupButton (C++ function), 54
- nanogui::PopupButton::preferredSize (C++ function), 54
- nanogui::PopupButton::save (C++ function), 55
- nanogui::PopupButton::setChevronIcon (C++ function), 54
- nanogui::PopupButton::setSide (C++ function), 54
- nanogui::PopupButton::side (C++ function), 54
- nanogui::ProgressBar (C++ class), 55
- nanogui::ProgressBar::draw (C++ function), 55
- nanogui::ProgressBar::load (C++ function), 55
- nanogui::ProgressBar::mValue (C++ member), 55
- nanogui::ProgressBar::preferredSize (C++ function), 55
- nanogui::ProgressBar::ProgressBar (C++ function), 55
- nanogui::ProgressBar::save (C++ function), 55
- nanogui::ProgressBar::setValue (C++ function), 55
- nanogui::ProgressBar::value (C++ function), 55
- nanogui::project (C++ function), 85
- nanogui::ref (C++ class), 56
- nanogui::ref::~~ref (C++ function), 56
- nanogui::ref::get (C++ function), 57
- nanogui::ref::operator
= (C++ function), 56
- nanogui::ref::operator bool (C++ function), 57
- nanogui::ref::operator T * (C++ function), 57
- nanogui::ref::operator* (C++ function), 57
- nanogui::ref::operator-> (C++ function), 56
- nanogui::ref::operator= (C++ function), 56
- nanogui::ref::operator== (C++ function), 56
- nanogui::ref::ref (C++ function), 56
- nanogui::scale (C++ function), 85
- nanogui::Screen (C++ class), 57
- nanogui::Screen::~~Screen (C++ function), 58
- nanogui::Screen::background (C++ function), 58
- nanogui::Screen::caption (C++ function), 58
- nanogui::Screen::centerWindow (C++ function), 59
- nanogui::Screen::charCallbackEvent (C++ function), 59
- nanogui::Screen::cursorPosCallbackEvent (C++ function), 59
- nanogui::Screen::disposeWindow (C++ function), 59
- nanogui::Screen::drawAll (C++ function), 58
- nanogui::Screen::drawContents (C++ function), 58
- nanogui::Screen::drawWidgets (C++ function), 59
- nanogui::Screen::dropCallbackEvent (C++ function), 59
- nanogui::Screen::dropEvent (C++ function), 58
- nanogui::Screen::glfwWindow (C++ function), 59
- nanogui::Screen::initialize (C++ function), 59
- nanogui::Screen::keyboardCharacterEvent (C++ function), 58
- nanogui::Screen::keyboardEvent (C++ function), 58
- nanogui::Screen::keyCallbackEvent (C++ function), 59
- nanogui::Screen::mBackground (C++ member), 60
- nanogui::Screen::mCaption (C++ member), 60
- nanogui::Screen::mCursor (C++ member), 59
- nanogui::Screen::mCursors (C++ member), 59
- nanogui::Screen::mDragActive (C++ member), 60
- nanogui::Screen::mDragWidget (C++ member), 60
- nanogui::Screen::mFBSize (C++ member), 60
- nanogui::Screen::mFocusPath (C++ member), 60
- nanogui::Screen::mFullscreen (C++ member), 60
- nanogui::Screen::mGLFWWindow (C++ member), 59
- nanogui::Screen::mLastInteraction (C++ member), 60
- nanogui::Screen::mModifiers (C++ member), 60
- nanogui::Screen::mMousePos (C++ member), 60
- nanogui::Screen::mMouseState (C++ member), 60
- nanogui::Screen::mNVGContext (C++ member), 59
- nanogui::Screen::mouseButtonCallbackEvent (C++ function), 59
- nanogui::Screen::mousePos (C++ function), 58
- nanogui::Screen::moveWindowToFront (C++ function), 59
- nanogui::Screen::mPixelRatio (C++ member), 60
- nanogui::Screen::mProcessEvent (C++ member), 60
- nanogui::Screen::mResizeCallback (C++ member), 60
- nanogui::Screen::mShutdownGLFWOnDestruct (C++ member), 60
- nanogui::Screen::nvgContext (C++ function), 59

- nanogui::Screen::performLayout (C++ function), 59
- nanogui::Screen::pixelRatio (C++ function), 58
- nanogui::Screen::resizeCallback (C++ function), 58
- nanogui::Screen::resizeCallbackEvent (C++ function), 59
- nanogui::Screen::resizeEvent (C++ function), 58
- nanogui::Screen::Screen (C++ function), 57, 59
- nanogui::Screen::scrollCallbackEvent (C++ function), 59
- nanogui::Screen::setBackground (C++ function), 58
- nanogui::Screen::setCaption (C++ function), 58
- nanogui::Screen::setResizeCallback (C++ function), 58
- nanogui::Screen::setShutdownGLFWOnDestruct (C++ function), 59
- nanogui::Screen::setSize (C++ function), 58
- nanogui::Screen::setVisible (C++ function), 58
- nanogui::Screen::shutdownGLFWOnDestruct (C++ function), 59
- nanogui::Screen::updateFocus (C++ function), 59
- nanogui::Serializer (C++ class), 60
- nanogui::Serializer::~~Serializer (C++ function), 60
- nanogui::Serializer::compatibility (C++ function), 61
- nanogui::Serializer::get (C++ function), 61
- nanogui::Serializer::get_base (C++ function), 61
- nanogui::Serializer::isSerializedFile (C++ function), 61
- nanogui::Serializer::keys (C++ function), 61
- nanogui::Serializer::pop (C++ function), 60
- nanogui::Serializer::push (C++ function), 60
- nanogui::Serializer::read (C++ function), 61
- nanogui::Serializer::readTOC (C++ function), 61
- nanogui::Serializer::seek (C++ function), 61
- nanogui::Serializer::Serializer (C++ function), 60
- nanogui::Serializer::set (C++ function), 61
- nanogui::Serializer::set_base (C++ function), 61
- nanogui::Serializer::setCompatibility (C++ function), 61
- nanogui::Serializer::size (C++ function), 60
- nanogui::Serializer::write (C++ function), 61
- nanogui::Serializer::writeTOC (C++ function), 61
- nanogui::shutdown (C++ function), 85
- nanogui::Slider (C++ class), 61
- nanogui::Slider::callback (C++ function), 62
- nanogui::Slider::draw (C++ function), 62
- nanogui::Slider::finalCallback (C++ function), 62
- nanogui::Slider::highlightColor (C++ function), 62
- nanogui::Slider::highlightedRange (C++ function), 62
- nanogui::Slider::load (C++ function), 62
- nanogui::Slider::mCallback (C++ member), 62
- nanogui::Slider::mFinalCallback (C++ member), 62
- nanogui::Slider::mHighlightColor (C++ member), 63
- nanogui::Slider::mHighlightedRange (C++ member), 62
- nanogui::Slider::mouseButtonEvent (C++ function), 62
- nanogui::Slider::mouseDragEvent (C++ function), 62
- nanogui::Slider::mRange (C++ member), 62
- nanogui::Slider::mValue (C++ member), 62
- nanogui::Slider::preferredSize (C++ function), 62
- nanogui::Slider::range (C++ function), 62
- nanogui::Slider::save (C++ function), 62
- nanogui::Slider::setCallback (C++ function), 62
- nanogui::Slider::setFinalCallback (C++ function), 62
- nanogui::Slider::setHighlightColor (C++ function), 62
- nanogui::Slider::setHighlightedRange (C++ function), 62
- nanogui::Slider::setRange (C++ function), 62
- nanogui::Slider::setValue (C++ function), 62
- nanogui::Slider::Slider (C++ function), 62
- nanogui::Slider::value (C++ function), 62
- nanogui::StackedWidget (C++ class), 63
- nanogui::StackedWidget::addChild (C++ function), 63
- nanogui::StackedWidget::performLayout (C++ function), 63
- nanogui::StackedWidget::preferredSize (C++ function), 63
- nanogui::StackedWidget::selectedIndex (C++ function), 63
- nanogui::StackedWidget::setSelectedIndex (C++ function), 63
- nanogui::StackedWidget::StackedWidget (C++ function), 63
- nanogui::TabHeader (C++ class), 63
- nanogui::TabHeader::activeButtonArea (C++ function), 64
- nanogui::TabHeader::activeTab (C++ function), 64
- nanogui::TabHeader::addTab (C++ function), 64
- nanogui::TabHeader::callback (C++ function), 64
- nanogui::TabHeader::ClickLocation (C++ type), 79
- nanogui::TabHeader::draw (C++ function), 65
- nanogui::TabHeader::ensureTabVisible (C++ function), 64
- nanogui::TabHeader::font (C++ function), 63
- nanogui::TabHeader::isTabVisible (C++ function), 64
- nanogui::TabHeader::LeftControls (C++ class), 79
- nanogui::TabHeader::mouseButtonEvent (C++ function), 64
- nanogui::TabHeader::overflowing (C++ function), 64
- nanogui::TabHeader::performLayout (C++ function), 64
- nanogui::TabHeader::preferredSize (C++ function), 64
- nanogui::TabHeader::removeTab (C++ function), 64
- nanogui::TabHeader::RightControls (C++ class), 79
- nanogui::TabHeader::setActiveTab (C++ function), 64
- nanogui::TabHeader::setCallback (C++ function), 64
- nanogui::TabHeader::setFont (C++ function), 63
- nanogui::TabHeader::TabButton (C++ class), 65
- nanogui::TabHeader::TabButton::calculateVisibleString (C++ function), 65
- nanogui::TabHeader::TabButton::dots (C++ member), 65
- nanogui::TabHeader::TabButton::drawActiveBorderAt (C++ function), 65
- nanogui::TabHeader::TabButton::drawAtPosition (C++ function), 65
- nanogui::TabHeader::TabButton::drawInactiveBorderAt (C++ function), 65

- nanogui::TabHeader::TabButton::label (C++ function), 65
- nanogui::TabHeader::TabButton::preferredSize (C++ function), 65
- nanogui::TabHeader::TabButton::setLabel (C++ function), 65
- nanogui::TabHeader::TabButton::setSize (C++ function), 65
- nanogui::TabHeader::TabButton::size (C++ function), 65
- nanogui::TabHeader::TabButton::StringView (C++ class), 21
- nanogui::TabHeader::TabButton::StringView::first (C++ member), 22
- nanogui::TabHeader::TabButton::StringView::last (C++ member), 22
- nanogui::TabHeader::TabButton::TabButton (C++ function), 65
- nanogui::TabHeader::TabButtons (C++ class), 79
- nanogui::TabHeader::tabCount (C++ function), 64
- nanogui::TabHeader::TabHeader (C++ function), 63
- nanogui::TabHeader::tabIndex (C++ function), 64
- nanogui::TabHeader::tabLabelAt (C++ function), 64
- nanogui::TabHeader::visibleButtonArea (C++ function), 64
- nanogui::TabWidget (C++ class), 65
- nanogui::TabWidget::activeTab (C++ function), 66
- nanogui::TabWidget::addTab (C++ function), 66
- nanogui::TabWidget::callback (C++ function), 66
- nanogui::TabWidget::createTab (C++ function), 66
- nanogui::TabWidget::draw (C++ function), 67
- nanogui::TabWidget::ensureTabVisible (C++ function), 66
- nanogui::TabWidget::performLayout (C++ function), 66
- nanogui::TabWidget::preferredSize (C++ function), 66
- nanogui::TabWidget::removeTab (C++ function), 66
- nanogui::TabWidget::setActiveTab (C++ function), 66
- nanogui::TabWidget::setCallback (C++ function), 66
- nanogui::TabWidget::tab (C++ function), 66
- nanogui::TabWidget::tabCount (C++ function), 66
- nanogui::TabWidget::tabIndex (C++ function), 66
- nanogui::TabWidget::tabLabelAt (C++ function), 66
- nanogui::TabWidget::tabLabelIndex (C++ function), 66
- nanogui::TabWidget::TabWidget (C++ function), 66
- nanogui::TextBox (C++ class), 67
- nanogui::TextBox::alignment (C++ function), 67
- nanogui::TextBox::Alignment (C++ type), 67, 78
- nanogui::TextBox::Bottom (C++ class), 69, 81
- nanogui::TextBox::callback (C++ function), 68
- nanogui::TextBox::Center (C++ class), 67, 79
- nanogui::TextBox::checkFormat (C++ function), 69
- nanogui::TextBox::copySelection (C++ function), 69
- nanogui::TextBox::cursorIndex2Position (C++ function), 69
- nanogui::TextBox::defaultValue (C++ function), 67
- nanogui::TextBox::deleteSelection (C++ function), 69
- nanogui::TextBox::draw (C++ function), 68
- nanogui::TextBox::editable (C++ function), 67
- nanogui::TextBox::focusEvent (C++ function), 68
- nanogui::TextBox::format (C++ function), 68
- nanogui::TextBox::keyboardCharacterEvent (C++ function), 68
- nanogui::TextBox::keyboardEvent (C++ function), 68
- nanogui::TextBox::Left (C++ class), 67, 79
- nanogui::TextBox::load (C++ function), 68
- nanogui::TextBox::mAlignment (C++ member), 69
- nanogui::TextBox::mCallback (C++ member), 69
- nanogui::TextBox::mCommitted (C++ member), 69
- nanogui::TextBox::mCursorPos (C++ member), 69
- nanogui::TextBox::mDefaultValue (C++ member), 69
- nanogui::TextBox::mEditable (C++ member), 69
- nanogui::TextBox::mFormat (C++ member), 69
- nanogui::TextBox::mLastClick (C++ member), 69
- nanogui::TextBox::mMouseDownModifier (C++ member), 69
- nanogui::TextBox::mMouseDownPos (C++ member), 69
- nanogui::TextBox::mMouseDragPos (C++ member), 69
- nanogui::TextBox::mMousePos (C++ member), 69
- nanogui::TextBox::mouseButtonEvent (C++ function), 68
- nanogui::TextBox::mouseDragEvent (C++ function), 68
- nanogui::TextBox::mouseMotionEvent (C++ function), 68
- nanogui::TextBox::mSelectionPos (C++ member), 69
- nanogui::TextBox::mSpinnable (C++ member), 69
- nanogui::TextBox::mTextOffset (C++ member), 69
- nanogui::TextBox::mUnits (C++ member), 69
- nanogui::TextBox::mUnitsImage (C++ member), 69
- nanogui::TextBox::mValidFormat (C++ member), 69
- nanogui::TextBox::mValue (C++ member), 69
- nanogui::TextBox::mValueTemp (C++ member), 69
- nanogui::TextBox::None (C++ class), 68, 81
- nanogui::TextBox::pasteFromClipboard (C++ function), 69
- nanogui::TextBox::position2CursorIndex (C++ function), 69
- nanogui::TextBox::preferredSize (C++ function), 68
- nanogui::TextBox::Right (C++ class), 67, 79
- nanogui::TextBox::save (C++ function), 68
- nanogui::TextBox::setAlignment (C++ function), 67
- nanogui::TextBox::setCallback (C++ function), 68
- nanogui::TextBox::setDefaultValue (C++ function), 67
- nanogui::TextBox::setEditable (C++ function), 67
- nanogui::TextBox::setFormat (C++ function), 68
- nanogui::TextBox::setSpinnable (C++ function), 67
- nanogui::TextBox::setTheme (C++ function), 68
- nanogui::TextBox::setUnits (C++ function), 67
- nanogui::TextBox::setUnitsImage (C++ function), 68
- nanogui::TextBox::setValue (C++ function), 67

nanogui::TextBox::spinArea (C++ function), 69
 nanogui::TextBox::SpinArea (C++ type), 68, 81
 nanogui::TextBox::spinnable (C++ function), 67
 nanogui::TextBox::TextBox (C++ function), 67
 nanogui::TextBox::Top (C++ class), 69, 81
 nanogui::TextBox::units (C++ function), 67
 nanogui::TextBox::unitsImage (C++ function), 68
 nanogui::TextBox::updateCursor (C++ function), 69
 nanogui::TextBox::value (C++ function), 67
 nanogui::Theme (C++ class), 70
 nanogui::Theme::~~Theme (C++ function), 71
 nanogui::Theme::mBorderDark (C++ member), 70
 nanogui::Theme::mBorderLight (C++ member), 70
 nanogui::Theme::mBorderMedium (C++ member), 70
 nanogui::Theme::mButtonCornerRadius (C++ member), 70
 nanogui::Theme::mButtonFontSize (C++ member), 70
 nanogui::Theme::mButtonGradientBotFocused (C++ member), 71
 nanogui::Theme::mButtonGradientBotPushed (C++ member), 71
 nanogui::Theme::mButtonGradientBotUnfocused (C++ member), 71
 nanogui::Theme::mButtonGradientTopFocused (C++ member), 71
 nanogui::Theme::mButtonGradientTopPushed (C++ member), 71
 nanogui::Theme::mButtonGradientTopUnfocused (C++ member), 71
 nanogui::Theme::mDisabledTextColor (C++ member), 70
 nanogui::Theme::mDropShadow (C++ member), 70
 nanogui::Theme::mFontBold (C++ member), 70
 nanogui::Theme::mFontIcons (C++ member), 70
 nanogui::Theme::mFontNormal (C++ member), 70
 nanogui::Theme::mIconColor (C++ member), 70
 nanogui::Theme::mStandardFontSize (C++ member), 70
 nanogui::Theme::mTabBorderWidth (C++ member), 70
 nanogui::Theme::mTabButtonHorizontalPadding (C++ member), 70
 nanogui::Theme::mTabButtonVerticalPadding (C++ member), 70
 nanogui::Theme::mTabControlWidth (C++ member), 70
 nanogui::Theme::mTabInnerMargin (C++ member), 70
 nanogui::Theme::mTabMaxButtonWidth (C++ member), 70
 nanogui::Theme::mTabMinButtonWidth (C++ member), 70
 nanogui::Theme::mTextBoxFontSize (C++ member), 70
 nanogui::Theme::mTextColor (C++ member), 70
 nanogui::Theme::mTextColorShadow (C++ member), 70
 nanogui::Theme::mTransparent (C++ member), 70
 nanogui::Theme::mWindowCornerRadius (C++ member), 70
 nanogui::Theme::mWindowDropShadowSize (C++ member), 70
 nanogui::Theme::mWindowFillFocused (C++ member), 71
 nanogui::Theme::mWindowFillUnfocused (C++ member), 71
 nanogui::Theme::mWindowHeaderGradientBot (C++ member), 71
 nanogui::Theme::mWindowHeaderGradientTop (C++ member), 71
 nanogui::Theme::mWindowHeaderHeight (C++ member), 70
 nanogui::Theme::mWindowHeaderSepBot (C++ member), 71
 nanogui::Theme::mWindowHeaderSepTop (C++ member), 71
 nanogui::Theme::mWindowPopup (C++ member), 71
 nanogui::Theme::mWindowPopupTransparent (C++ member), 71
 nanogui::Theme::mWindowTitleFocused (C++ member), 71
 nanogui::Theme::mWindowTitleUnfocused (C++ member), 71
 nanogui::Theme::Theme (C++ function), 70
 nanogui::ToolButton (C++ class), 71
 nanogui::ToolButton::ToolButton (C++ function), 71
 nanogui::translate (C++ function), 86
 nanogui::UniformBufferStd140 (C++ class), 71
 nanogui::UniformBufferStd140::Parent (C++ type), 72
 nanogui::UniformBufferStd140::push_back (C++ function), 72
 nanogui::unproject (C++ function), 86
 nanogui::utf8 (C++ function), 86
 nanogui::Vertical (C++ class), 80
 nanogui::VResize (C++ class), 80
 nanogui::VScrollPanel (C++ class), 72
 nanogui::VScrollPanel::draw (C++ function), 72
 nanogui::VScrollPanel::load (C++ function), 72
 nanogui::VScrollPanel::mChildPreferredHeight (C++ member), 72
 nanogui::VScrollPanel::mouseDragEvent (C++ function), 72
 nanogui::VScrollPanel::mScroll (C++ member), 72
 nanogui::VScrollPanel::mUpdateLayout (C++ member), 72
 nanogui::VScrollPanel::performLayout (C++ function), 72
 nanogui::VScrollPanel::preferredSize (C++ function), 72
 nanogui::VScrollPanel::save (C++ function), 72
 nanogui::VScrollPanel::scrollEvent (C++ function), 72
 nanogui::VScrollPanel::VScrollPanel (C++ function), 72
 nanogui::Widget (C++ class), 73
 nanogui::Widget::~~Widget (C++ function), 76
 nanogui::Widget::absolutePosition (C++ function), 73

nanogui::Widget::add (C++ function), 75
 nanogui::Widget::addChild (C++ function), 74
 nanogui::Widget::childAt (C++ function), 75
 nanogui::Widget::childCount (C++ function), 74
 nanogui::Widget::childIndex (C++ function), 75
 nanogui::Widget::children (C++ function), 74
 nanogui::Widget::contains (C++ function), 76
 nanogui::Widget::cursor (C++ function), 75
 nanogui::Widget::draw (C++ function), 76
 nanogui::Widget::enabled (C++ function), 75
 nanogui::Widget::findWidget (C++ function), 76
 nanogui::Widget::fixedHeight (C++ function), 74
 nanogui::Widget::fixedSize (C++ function), 74
 nanogui::Widget::fixedWidth (C++ function), 74
 nanogui::Widget::focused (C++ function), 75
 nanogui::Widget::focusEvent (C++ function), 76
 nanogui::Widget::fontSize (C++ function), 75
 nanogui::Widget::hasFontSize (C++ function), 75
 nanogui::Widget::height (C++ function), 74
 nanogui::Widget::id (C++ function), 75
 nanogui::Widget::keyboardCharacterEvent (C++ function), 76
 nanogui::Widget::keyboardEvent (C++ function), 76
 nanogui::Widget::layout (C++ function), 73
 nanogui::Widget::load (C++ function), 76
 nanogui::Widget::mChildren (C++ member), 77
 nanogui::Widget::mCursor (C++ member), 77
 nanogui::Widget::mEnabled (C++ member), 77
 nanogui::Widget::mFixedSize (C++ member), 77
 nanogui::Widget::mFocused (C++ member), 77
 nanogui::Widget::mFontSize (C++ member), 77
 nanogui::Widget::mId (C++ member), 77
 nanogui::Widget::mLayout (C++ member), 77
 nanogui::Widget::mMouseFocus (C++ member), 77
 nanogui::Widget::mouseButtonEvent (C++ function), 76
 nanogui::Widget::mouseDragEvent (C++ function), 76
 nanogui::Widget::mouseEnterEvent (C++ function), 76
 nanogui::Widget::mouseMotionEvent (C++ function), 76
 nanogui::Widget::mParent (C++ member), 77
 nanogui::Widget::mPos (C++ member), 77
 nanogui::Widget::mSize (C++ member), 77
 nanogui::Widget::mTheme (C++ member), 77
 nanogui::Widget::mTooltip (C++ member), 77
 nanogui::Widget::mVisible (C++ member), 77
 nanogui::Widget::parent (C++ function), 73
 nanogui::Widget::performLayout (C++ function), 76
 nanogui::Widget::position (C++ function), 73
 nanogui::Widget::preferredSize (C++ function), 76
 nanogui::Widget::removeChild (C++ function), 75
 nanogui::Widget::requestFocus (C++ function), 75
 nanogui::Widget::save (C++ function), 76
 nanogui::Widget::scrollEvent (C++ function), 76
 nanogui::Widget::setCursor (C++ function), 76
 nanogui::Widget::setEnabled (C++ function), 75
 nanogui::Widget::setFixedHeight (C++ function), 74
 nanogui::Widget::setFixedSize (C++ function), 74
 nanogui::Widget::setFixedWidth (C++ function), 74
 nanogui::Widget::setFocused (C++ function), 75
 nanogui::Widget::setFontSize (C++ function), 75
 nanogui::Widget::setHeight (C++ function), 74
 nanogui::Widget::setId (C++ function), 75
 nanogui::Widget::setLayout (C++ function), 73
 nanogui::Widget::setParent (C++ function), 73
 nanogui::Widget::setPosition (C++ function), 73
 nanogui::Widget::setSize (C++ function), 74
 nanogui::Widget::setTheme (C++ function), 73
 nanogui::Widget::setTooltip (C++ function), 75
 nanogui::Widget::setVisible (C++ function), 74
 nanogui::Widget::setWidth (C++ function), 74
 nanogui::Widget::size (C++ function), 73
 nanogui::Widget::theme (C++ function), 73
 nanogui::Widget::tooltip (C++ function), 75
 nanogui::Widget::visible (C++ function), 74
 nanogui::Widget::visibleRecursive (C++ function), 74
 nanogui::Widget::Widget (C++ function), 73
 nanogui::Widget::width (C++ function), 74
 nanogui::Widget::window (C++ function), 75
 nanogui::Window (C++ class), 77
 nanogui::Window::buttonPanel (C++ function), 77
 nanogui::Window::center (C++ function), 78
 nanogui::Window::dispose (C++ function), 77
 nanogui::Window::draw (C++ function), 78
 nanogui::Window::load (C++ function), 78
 nanogui::Window::mButtonPanel (C++ member), 78
 nanogui::Window::mDrag (C++ member), 78
 nanogui::Window::mModal (C++ member), 78
 nanogui::Window::modal (C++ function), 77
 nanogui::Window::mouseButtonEvent (C++ function), 78
 nanogui::Window::mouseDragEvent (C++ function), 78
 nanogui::Window::mTitle (C++ member), 78
 nanogui::Window::performLayout (C++ function), 78
 nanogui::Window::preferredSize (C++ function), 78
 nanogui::Window::refreshRelativePlacement (C++ function), 78
 nanogui::Window::save (C++ function), 78
 nanogui::Window::scrollEvent (C++ function), 78
 nanogui::Window::setModal (C++ function), 77
 nanogui::Window::setTitle (C++ function), 77
 nanogui::Window::title (C++ function), 77
 nanogui::Window::Window (C++ function), 77
 NANOGUI_EXPORT (C macro), 87
 NANOGUI_FORCE_DISCRETE_GPU (C macro), 87
 NANOGUI_LAYOUT_OVERLOADS (C macro), 87
 NANOGUI_SCREEN_OVERLOADS (C macro), 88
 NANOGUI_SNPRINTF (C macro), 88
 NANOGUI_WIDGET_OVERLOADS (C macro), 88
 nvglImageIcon (C macro), 88

S

SYSTEM_COMMAND_MOD (C macro), [88](#)