NanoGUI Documentation

Release 0.1.0

Wenzel Jakob

Sep 20, 2017

Contents

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.

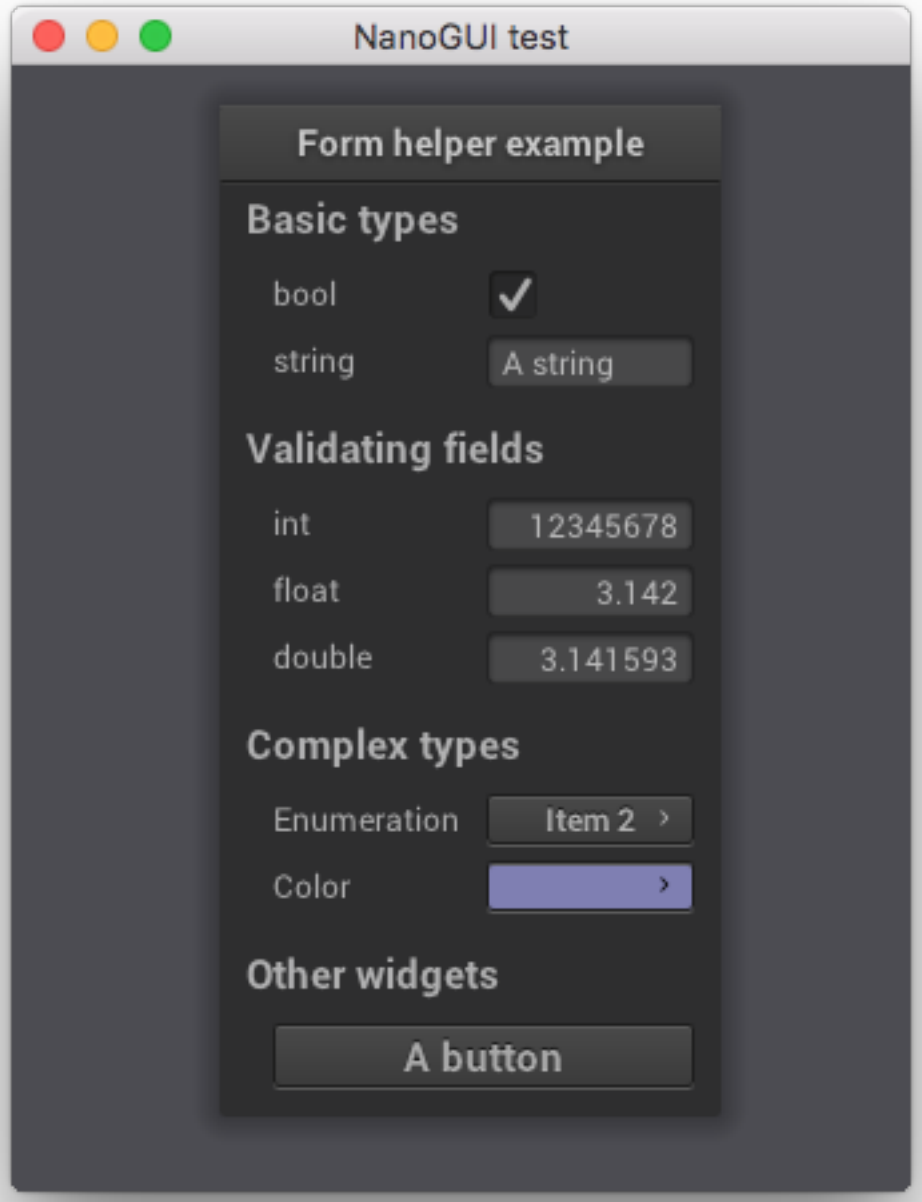
Note that the dependency library NanoVG already includes some basic example code to draw good-looking static widgets; what NanoGUI does is to flesh it out into a complete GUI toolkit with event handling, layout generation, etc.

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.


NanoGUI test
Form helper example
Basic types
bool
string
A string
Validating fields
int
12345678
float
3.142
double
3.141593
Complex types
Enumeration
Item 2
Color
Other widgets
A button

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.

NanoGUI uses Daniel Bruce's Entypo+ font for the icons used on various widgets. This work is licensed under a CC BY-SA 4.0 license. Commercial entities using NanoGUI should consult the proper legal counsel for how to best adhere to the attribution clause of the license.

Note: The CC BY-SA 4.0 license should not be an issue for most projects. However, you can adopt a different font for icons if you need. See *Including Custom Fonts*.

CHAPTER 5

Contents

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.  Execute `cmake -G` to get
# a listing of the available generators.
#
# 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 *shared* 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})
```

Advanced Compilation Details

NanoGUI and Python

Although it is 2017, you may still for example wish to build the Python bindings for Python 2.7. The variable you would set **before** `add_subdirectory` is `NANOGUI_PYTHON_VERSION`. For example,

```
set(NANOGUI_PYTHON_VERSION "2.7")
# can also use minor versions
set(NANOGUI_PYTHON_VERSION "3.6.2")
```

NanoGUI and Eigen

NanoGUI uses Eigen internally for various vector types. Eigen is an advanced header only template library, which NanoGUI vendors in the `ext` folder. It is important to understand the implication of Eigen being header only: **only one version of Eigen can be included**.

There is a CMake bypass variable available in NanoGUI: `NANOGUI_EIGEN_INCLUDE_DIR`. You would set this variable **before** `add_subdirectory`. Since you will want to provide the same kind of bypass for users of your library, the following snippet is a good starting point. For this example code:

1. The parent CMake project is called `myproj`. A good CMake practice to adopt is to prefix your project's name to any variables you intend to expose. This allows parent projects to know where the variable came from, and avoids name collisions.
2. First `find_package` is used to try and find Eigen. The philosophy is that the user is responsible for ensuring that the version of Eigen they want to use will be found.
3. Since NanoGUI needs to remain self-contained, the side-effect is that even if the user does *not* have Eigen installed, you can fallback and use the one vendored with NanoGUI.
4. The following directory structure:

```
myproj/
    CMakeLists.txt         <- Where this example code is
    ext/
        nanogui/
            CMakeLists.txt <- NanoGUI's build system
            ext/
                eigen/     <- NanoGUI's internal copy of Eigen
```

```
# `if NOT` is what enables the same bypass for your project
if(NOT MYPROJ_EIGEN3_INCLUDE_DIR)
  # Grab or find the Eigen3 include directory.
  find_package(Eigen3 QUIET)
  if(EIGEN3_INCLUDE_DIR)
    set(MYPROJ_EIGEN3_INCLUDE_DIR ${EIGEN3_INCLUDE_DIR})
  else()
    # use the internal NanoGUI copy of Eigen
    set(MYPROJ_EIGEN3_INCLUDE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/ext/nanogui/ext/eigen)
  endif()
endif()

message(STATUS "Using Eigen3 from directory: ${MYPROJ_EIGEN3_INCLUDE_DIR}")
set(NANOGUI_EIGEN_INCLUDE_DIR ${EIGEN3_INCLUDE_DIR} CACHE BOOL " " FORCE)
# set any other NanoGUI specific variables you need (shown in above sections)
add_subdirectory(ext/nanogui)

# include it for your project as well (or append to a list
# and include that list later, depending on your setup)
include_directories(${MYPROJ_EIGEN3_INCLUDE_DIR})
```

NanoGUI, GLFW, and Other Projects

Suppose you want to use NanoGUI as your GUI toolkit, but you also have another library you want to use that depends on `glfw`. Call the second library Foo. Generally speaking, it is unlikely that library Foo will provide you with mechanisms to explicitly specify where `glfw` comes from. You could try to work on a patch with the developers

of library Foo to allow this to be overridden, but you may need to maintain your own fork of library Foo. There is just as much justification to allow the bypass as there is to not want it in a build system.

Since NanoGUI merges the `glfw` objects into the library being built, you can actually just specify `nanogui` as the `glfw` dependency directly. So lets suppose that library Foo was looking for `glfw` like this:

```
find_package(GLFW3)
if(GLFW3_FOUND)
  include_directories(${GLFW3_INCLUDE_DIRS})
  target_link_libraries(foo ${GLFW3_LIBRARIES})
endif()
```

You can cheat around this pretty easily. For the modification to library Foo's build system, all we do is wrap `find_package`:

```
+ if(NOT GLFW3_FOUND)
    find_package(GLFW3)
+ endif()
  if(GLFW3_FOUND)
    include_directories(${GLFW3_INCLUDE_DIRS})
    target_link_libraries(foo ${GLFW3_LIBRARIES})
  endif()
```

Now that `find_package` will only execute if `NOT GLFW3_FOUND`, in your build system you make sure to set all three `glfw` variables (found, include, and libraries). It might look something like this:

```
# ... any other nanogui configs ...
# same directory structure as Eigen example
add_subdirectory(ext/nanogui)

# nanogui needs to be added first so the 'nanogui' target is defined
# and can be used in the generator expression for the libraries
set(GLFW3_FOUND ON)
set(GLFW3_INCLUDE_DIRS ${CMAKE_CURRENT_SOURCE_DIR}/ext/nanogui/ext/glfw/include)
set(GLFW3_LIBRARIES $<TARGET_FILE:nanogui>)

add_subdirectory(ext/foo)

# IMPORTANT! You need to force NanoGUI to build first
# Assuming their library target is called 'foo'
add_dependencies(foo nanogui)
```

Depending on what you need to do, the above may not be sufficient. But it is at least a starting point to being able to "share" NanoGUI as the vendor of `glfw`.

Including Custom Fonts

NanoGUI uses the Roboto font for text, and Entypo font for icons. If you wish to add your own custom font, all you need is a True Type file (a `.ttf` extension). NanoGUI will glob all fonts found in `resources` by expanding `resources/*.ttf`. So if you had the directory structure

```
myproject/
    CMakeLists.txt      <- where this code is
    fonts/
        superfont.ttf
    ext/
```

```
        nanogui/
            resources/
```

You simply need to copy the `superfont.ttf` to NanoGUI's resources directory:

```
file(
  COPY ${CMAKE_CURRENT_SOURCE_DIR}/fonts/superfont.ttf
  DESTINATION ${CMAKE_CURRENT_SOURCE_DIR}/ext/nanogui/resources/superfont.ttf
)
```

When you build the code, there should be a file `nanogui_resources.h` generated. If everything worked, your new font should have been included.

Note: Since NanoGUI can support images as icons, you will want to make sure that the *codepoint* for any *icon* fonts you create is greater than `1024`. See *nanogui::nvgIsImageIcon()*.

Tip: Some widgets allow you to set fonts directly, but if you want to apply the font globally, you should create a sub-class of *nanogui::Theme* and explicitly call *nanogui::Widget::setTheme()* for each widget you create.

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, Exhale, and the theme:

   ```
   pip3 install exhale 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
```

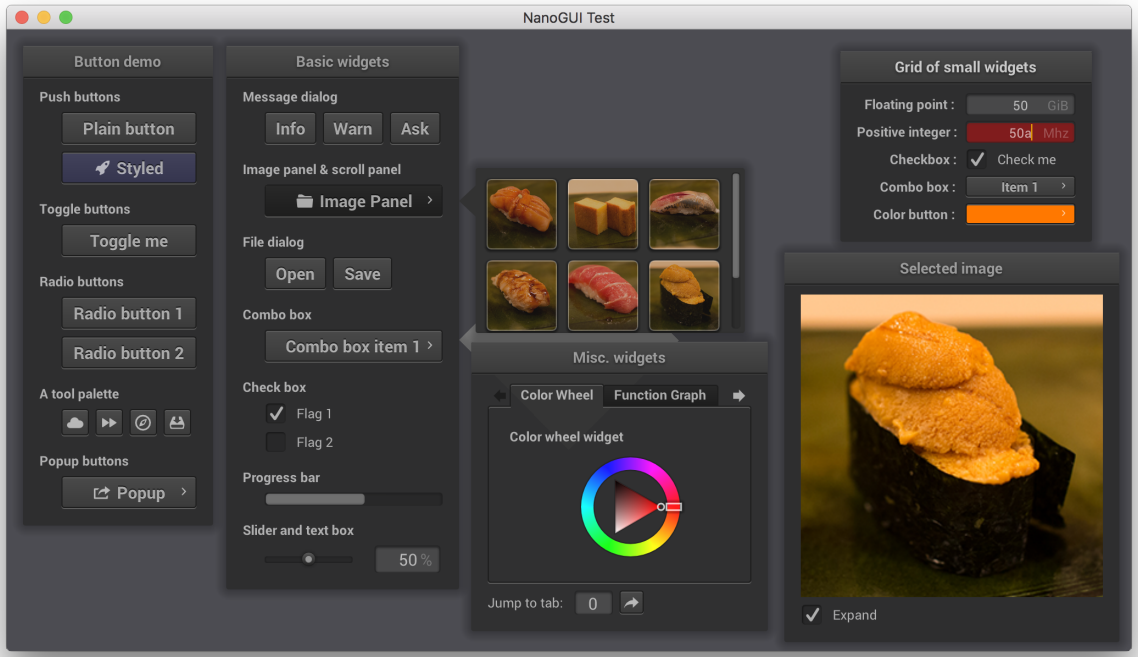
The output will be generated in `_build`, the root html document is located at `_build/html/index.html`.

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`.

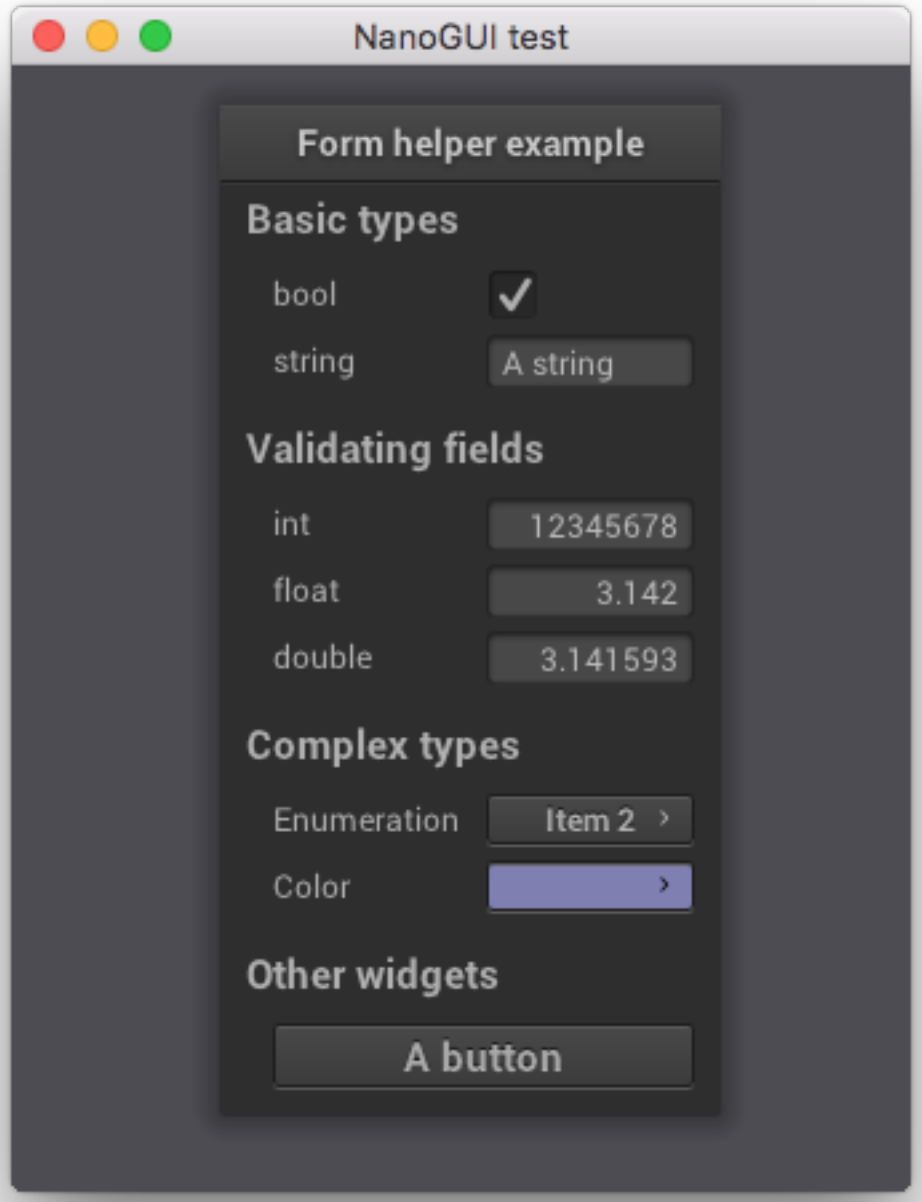
Example 1



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 Icons

NanoGUI includes various icons made available from *File entypo.h*, courtesy of Daniel Bruce's Entypo glyphs. The file level documentation is useful as a reference for selecting an icon, but the web rendering may be misleading — NanoGUI uses a dark background for widgets by default.

Run the `exampleIcon` executable to see what the icons look like in NanoGUI. The setup of this file may also be helpful in understanding how to control the *nanogui::VScrollPanel*.

- Example Icons in C++
- Example Icons 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

Page Contents

Namespaces

Classes

- *Class GLFramebuffer*
- *Class GLShader*
- *Class GLUniformBuffer*
- *Class Graph*
- *Class GridLayout*
- *Class GroupLayout*
- *Class ImagePanel*
- *Class ImageView*
- *Template Class IntBox*
- *Class Label*
- *Class Layout*
- *Class MessageDialog*
- *Class Object*
- *Class Popup*
- *Class PopupButton*
- *Class ProgressBar*
- *Template 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 Alignment*
- *Enum Cursor*
- *Enum Orientation*

Functions

Typedefs

Namespace nanogui::detail

Page Contents

Classes

Classes and Structs

Struct AdvancedGridLayout::Anchor

- Defined in *File layout.h*

Page Contents

Nested Relationships

This struct is a nested type of *Class AdvancedGridLayout*.

Struct Documentation

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

Public Functions

Anchor()
Creates a `0` *Anchor*.

Anchor(int *x*, int *y*, *Alignment* *horiz* = Alignment::Fill, *Alignment* *vert* = Alignment::Fill)
Create an *Anchor* at position `(x, y)` with specified Alignment.

Anchor(int *x*, int *y*, int *w*, int *h*, *Alignment* *horiz* = Alignment::Fill, *Alignment* *vert* = Alignment::Fill)
Create an *Anchor* at position `(x, y)` of size `(w, h)` with specified alignments.

operator std::string() **const**
Allows for printing out *Anchor* position, size, and alignment.

Public Members

uint8_t **pos**[2]
The `(x, y)` position.

uint8_t **size**[2]
The `(x, y)` size.

Alignment **align**[2]
The `(x, y)` Alignment.

Struct Arcball

- Defined in *File glutil.h*

Page Contents

Struct Documentation

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**

Template Struct serialization_helper

- Defined in *File core.h*

Page Contents

Struct Documentation

template <typename T>
struct `nanogui::detail::`**serialization_helper**

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

Template Struct serialization_traits

- Defined in *File core.h*

Page Contents

Struct Documentation

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 explicity be serialized.

Struct GLShader::Buffer

- Defined in *File glutil.h*

Page Contents

Nested Relationships

This struct is a nested type of *Class GLShader*.

Struct Documentation

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*

Page Contents

Nested Relationships

This struct is a nested type of *Class TabHeader::TabButton*.

Struct Documentation

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*

Page Contents

Nested Relationships

Nested Types

- *Struct AdvancedGridLayout::Anchor*

Inheritance Relationships

Base Type

- `public nanogui::Layout` (*Class Layout*)

Class Documentation

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)

Creates an *AdvancedGridLayout* with specified columns, rows, and margin.

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

The margin of this *AdvancedGridLayout*.

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

Sets the margin of this *AdvancedGridLayout*.

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**
See *Layout::preferredSize*.

virtual void **performLayout**(NVGcontext *ctx, *Widget* *widget) **const**
See *Layout::performLayout*.

Protected Functions

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

Protected Attributes

std::vector<int> **mCols**
The columns of this *AdvancedGridLayout*.

std::vector<int> **mRows**
The rows of this *AdvancedGridLayout*.

std::vector<float> **mColStretch**
The stretch for each column of this *AdvancedGridLayout*.

std::vector<float> **mRowStretch**
The stretch for each row of this *AdvancedGridLayout*.

std::unordered_map<**const** *Widget* *, *Anchor*> **mAnchor**
The mapping of widgets to their specified anchor points.

int **mMargin**
The margin around this *AdvancedGridLayout*.

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

Public Functions

Anchor()
Creates a `0` *Anchor*.

Anchor(int *x*, int *y*, *Alignment* *horiz* = Alignment::Fill, *Alignment* *vert* = Alignment::Fill)
Create an *Anchor* at position `(x, y)` with specified Alignment.

Anchor(int *x*, int *y*, int *w*, int *h*, *Alignment* *horiz* = Alignment::Fill, *Alignment* *vert* = Alignment::Fill)
Create an *Anchor* at position `(x, y)` of size `(w, h)` with specified alignments.

operator std::string() const
Allows for printing out *Anchor* position, size, and alignment.

Public Members

uint8_t **pos**[2]
The `(x, y)` position.

uint8_t **size**[2]
The `(x, y)` size.

Alignment **align**[2]
The `(x, y)` Alignment.

Class BoxLayout

- Defined in *File layout.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Layout` (*Class Layout*)

Class Documentation

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

- `orientation`: The Orientation this *BoxLayout* expands along

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

Orientation **`orientation`**`()` **const**
The Orientation this *BoxLayout* is using.

void **`setOrientation`**(*Orientation orientation*)
Sets the Orientation of this *BoxLayout*.

Alignment **`alignment`**`()` **const**
The Alignment of this *BoxLayout*.

void **`setAlignment`**(*Alignment alignment*)
Sets the Alignment of this *BoxLayout*.

int **`margin`**`()` **const**
The margin of this *BoxLayout*.

void **`setMargin`**(int *margin*)
Sets the margin of this *BoxLayout*.

int **`spacing`**`()` **const**
The spacing this *BoxLayout* is using to pad in between widgets.

void **`setSpacing`**(int *spacing*)
Sets the spacing of this *BoxLayout*.

virtual Vector2i **`preferredSize`**(NVGcontext **ctx*, **const** *Widget* **widget*) **const**
See *Layout::preferredSize*.

virtual void **`performLayout`**(NVGcontext **ctx*, *Widget* **widget*) **const**
See *Layout::performLayout*.

Protected Attributes

Orientation **`mOrientation`**
The Orientation of this *BoxLayout*.

Alignment **`mAlignment`**
The Alignment of this *BoxLayout*.

int **`mMargin`**
The margin of this *BoxLayout*.

int **`mSpacing`**
The spacing between widgets of this *BoxLayout*.

Class Button

- Defined in *File button.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Derived Types

- `public nanogui::PopupButton` (*Class PopupButton*)
- `public nanogui::ToolButton` (*Class ToolButton*)

Class Documentation

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)
A normal *Button*.

`RadioButton` = (1 << 1)
A radio *Button*.

`ToggleButton` = (1 << 2)
A toggle *Button*.

`PopupButton` = (1 << 3)
A popup *Button*.

enum `IconPosition`

The available icon positions.

Values:

`Left`
Button icon on the far left.

LeftCentered
Button icon on the left, centered (depends on caption text length).

RightCentered
Button icon on the right, centered (depends on caption text length).

Right
Button icon on the far right.

Public Functions

Button(*Widget* *parent, **const** std::string &caption = "Untitled", int icon = 0)
Creates a button attached to the specified parent.

Parameters

- `parent`: The *nanogui::Widget* this *Button* will be attached to.
- `caption`: The name of the button (default `"Untitled"`).
- `icon`: The icon to display with this *Button*. See *nanogui::Button::mIcon*.

const std::string &**caption**() **const**
Returns the caption of this *Button*.

void **setCaption**(**const** std::string &caption)
Sets the caption of this *Button*.

const *Color* &**backgroundColor**() **const**
Returns the background color of this *Button*.

void **setBackgroundColor**(**const** *Color* &backgroundColor)
Sets the background color of this *Button*.

const *Color* &**textColor**() **const**
Returns the text color of the caption of this *Button*.

void **setTextColor**(**const** *Color* &textColor)
Sets the text color of the caption of this *Button*.

int **icon**() **const**
Returns the icon of this *Button*. See *nanogui::Button::mIcon*.

void **setIcon**(int icon)
Sets the icon of this *Button*. See *nanogui::Button::mIcon*.

int **flags**() **const**
The current flags of this *Button* (see *nanogui::Button::Flags* for options).

void **setFlags**(int buttonFlags)
Sets the flags of this *Button* (see *nanogui::Button::Flags* for options).

IconPosition **iconPosition**() **const**
The position of the icon for this *Button*.

void **setIconPosition**(*IconPosition* iconPosition)
Sets the position of the icon for this *Button*.

bool **pushed**() **const**
Whether or not this *Button* is currently pushed.

void **setPushed**(bool *pushed*)
Sets whether or not this *Button* is currently pushed.

std::function<void()> **callback**
constThe current callback to execute (for any type of button).

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

std::function<void(bool)> **changeCallback**
constThe current callback to execute (for toggle buttons).

void **setChangeCallback**(**const** std::function<void() bool
> &*callback*Set the change callback (for toggle buttons).

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

const std::vector<*Button* *> &**buttonGroup**() **const**
The current button group (for radio buttons).

virtual Vector2i **preferredSize**(NVGcontext **ctx*) **const**
The preferred size of this *Button*.

virtual bool **mouseButtonEvent**(**const** Vector2i &*p*, int *button*, bool *down*, int *modifiers*)
The callback that is called when any type of mouse button event is issued to this *Button*.

virtual void **draw**(NVGcontext **ctx*)
Responsible for drawing the *Button*.

virtual void **save**(*Serializer* &*s*) **const**
Saves the state of this *Button* provided the given *Serializer*.

virtual bool **load**(*Serializer* &*s*)
Sets the state of this *Button* provided the given *Serializer*.

Protected Attributes

std::string **mCaption**
The caption of this *Button*.

int **mIcon**
The icon of this *Button* (`0` means no icon).

The icon to display with this Button. If not `0`, may either be a picture icon, or one of the icons enumerated in *File entypo.h*. The kind of icon (image or Entypo) is determined by the functions `nanogui::nvgIsImageIcon()` and its reciprocal counterpart `nanogui::nvgIsFontIcon()`.

IconPosition **mIconPosition**
The position to draw the icon at.

bool **mPushed**
Whether or not this *Button* is currently pushed.

int **mFlags**
The current flags of this button (see *nanogui::Button::Flags* for options).

Color **mBackgroundColor**
The background color of this *Button*.

Color **mTextColor**
The color of the caption text of this *Button*.

std::function<void ()> **mCallback**
The callback issued for all types of buttons.

std::function<void (bool)> **mChangeCallback**
The callback issued for toggle buttons.

std::vector<*Button* *> **mButtonGroup**
The button group for radio buttons.

Class CheckBox

- Defined in *File checkbox.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Derived Type

- `public nanogui::detail::FormWidget< bool, std::true_type >` (*Template Class FormWidget< bool, std::true_type >*)

Class Documentation

class `nanogui::CheckBox`
Two-state check box widget.

Remark This class overrides *nanogui::Widget::mIconExtraScale* to be `1.2f`, which affects all subclasses of this *Widget*. Subclasses must explicitly set a different value if needed (e.g., in their constructor).

Inherits from *nanogui::Widget*

Subclassed by *nanogui::detail::FormWidget< bool, std::true_type >*

Public Functions

CheckBox(*Widget* *parent, **const** std::string &caption = "Untitled", **const** std::function<void(bool)> &callback = std::function< void(bool)>()Adds a *CheckBox* to the specified `parent`.

Parameters

- `parent`: The *Widget* to add this *CheckBox* to.
- `caption`: The caption text of the *CheckBox* (default `"Untitled"`).
- `callback`: If provided, the callback to execute when the *CheckBox* is checked or unchecked. Default parameter function does nothing. See *nanogui::CheckBox::mPushed* for the difference between "pushed" and "checked".

const std::string &**caption**() **const**
The caption of this *CheckBox*.

void **setCaption**(**const** std::string &*caption*)
Sets the caption of this *CheckBox*.

const bool &**checked**() **const**
Whether or not this *CheckBox* is currently checked.

void **setChecked**(**const** bool &*checked*)
Sets whether or not this *CheckBox* is currently checked.

const bool &**pushed**() **const**
Whether or not this *CheckBox* is currently pushed. See *nanogui::CheckBox::mPushed*.

void **setPushed**(**const** bool &*pushed*)
Sets whether or not this *CheckBox* is currently pushed. See *nanogui::CheckBox::mPushed*.

std::function<void(bool)> **callback**
constReturns the current callback of this *CheckBox*.

void **setCallback**(**const** std::function<void(bool)> &*callback*Sets the callback to be executed when this *CheckBox* is checked / unchecked.

virtual bool **mouseButtonEvent**(**const** Vector2i &*p*, int *button*, bool *down*, int *modifiers*)
The mouse button callback will return `true` when all three conditions are met:

1. This *CheckBox* is "enabled" (see *nanogui::Widget::mEnabled*).
2. `p` is inside this *CheckBox*.
3. `button` is `GLFW_MOUSE_BUTTON_1` (left mouse click).

Since a mouse button event is issued for both when the mouse is pressed, as well as released, this function sets *nanogui::CheckBox::mPushed* to `true` when parameter `down == true`. When the second event (`down == false`) is fired, *nanogui::CheckBox::mChecked* is inverted and *nanogui::CheckBox::mCallback* is called.

That is, the callback provided is only called when the mouse button is released, **and** the click location remains within the *CheckBox* boundaries. If the user clicks on the *CheckBox* and releases away from the bounds of the *CheckBox*, *nanogui::CheckBox::mPushed* is simply set back to `false`.

virtual Vector2i **preferredSize**(NVGcontext *ctx) **const**
The preferred size of this *CheckBox*.

virtual void **draw**(NVGcontext *ctx)
Draws this *CheckBox*.

virtual void **save**(*Serializer* &s) **const**
Saves this *CheckBox* to the specified *Serializer*.

virtual bool **load**(*Serializer* &s)
Loads the state of the specified *Serializer* to this *CheckBox*.

Protected Attributes

std::string **mCaption**
The caption text of this *CheckBox*.

bool **mPushed**
Internal tracking variable to distinguish between mouse click and release. *nanogui::CheckBox::mCallback* is only called upon release. See *nanogui::CheckBox::mouseButtonEvent* for specific conditions.

bool **mChecked**
Whether or not this *CheckBox* is currently checked or unchecked.

std::function<void(bool)> **mCallback**
The function to execute when *nanogui::CheckBox::mChecked* is changed.

Class Color

- Defined in *File common.h*

Page Contents

Inheritance Relationships

Base Type

- `public Vector4f`

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::PopupButton` (*Class PopupButton*)

Derived Type

- `public nanogui::detail::FormWidget< Color, std::true_type >` (*Template Class FormWidget< Color, std::true_type >*)

Class Documentation

class `nanogui::ColorPicker`

Push button with a popup to tweak a color value. This widget was contributed by Christian Schueller.

Inherits from *nanogui::PopupButton*

Subclassed by *nanogui::detail::FormWidget< Color, std::true_type >*

Public Functions

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

Attaches a *ColorPicker* to the specified parent.

Parameters

- `parent`: The *Widget* to add this *ColorPicker* to.
- `color`: The color initially selected by this *ColorPicker* (default: Red).

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

constThe callback executed when the *ColorWheel* changes.

void **setCallback**(**const** std::function<void) **const** *Color*& > &*callback*Sets the callback is executed as the *ColorWheel* itself is changed. Set this callback if you need to receive updates for the *ColorWheel* changing before the user clicks *nanogui::ColorPicker::mPickButton* or *nanogui::ColorPicker::mPickButton*.

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

constThe callback to execute when a new *Color* is selected on the *ColorWheel* **and** the user clicks the *nanogui::ColorPicker::mPickButton* or *nanogui::ColorPicker::mResetButton*.

void **setFinalCallback**(**const** std::function<void) **const** *Color*& > &*callback*The callback to execute when a new *Color* is selected on the *ColorWheel* **and** the user clicks the *nanogui::ColorPicker::mPickButton* or *nanogui::ColorPicker::mResetButton*.

Color **color**() **const**

Get the current *Color* selected for this *ColorPicker*.

void **setColor**(**const** *Color* &*color*)

Set the current *Color* selected for this *ColorPicker*.

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

The current caption of the *nanogui::ColorPicker::mPickButton*.

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

Sets the current caption of the *nanogui::ColorPicker::mPickButton*.

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

The current caption of the *nanogui::ColorPicker::mResetButton*.

void **setResetButtonCaption**(**const** std::string &*caption*)
Sets the current caption of the *nanogui::ColorPicker::mResetButton*.

Protected Attributes

std::function<void(**const** *Color*&)> **mCallback**
The "fast" callback executed when the *ColorWheel* has changed.

std::function<void(**const** *Color*&)> **mFinalCallback**
The callback to execute when a new *Color* is selected on the *ColorWheel* **and** the user clicks the *nanogui::ColorPicker::mPickButton* or *nanogui::ColorPicker::mResetButton*.

ColorWheel ***mColorWheel**
The *ColorWheel* for this *ColorPicker* (the actual widget allowing selection).

Button ***mPickButton**
The *Button* used to signal that the current value on the *ColorWheel* is the desired color to be chosen. The default value for the caption of this *Button* is `"Pick"`. You can change it using *nanogui::ColorPicker::setPickButtonCaption* if you need.

The color of this *Button* will not affect *nanogui::ColorPicker::color* until the user has actively selected by clicking this pick button. Similarly, the *nanogui::ColorPicker::mCallback* function is only called when a user selects a new *Color* using by clicking this *Button*.

Button ***mResetButton**
Remains the *Color* of the active color selection, until the user picks a new *Color* on the *ColorWheel* **and** selects the *nanogui::ColorPicker::mPickButton*. The default value for the caption of this *Button* is `"Reset"`. You can change it using *nanogui::ColorPicker::setResetButtonCaption* if you need.

Class ColorWheel

- Defined in *File colorwheel.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

class `nanogui::ColorWheel`
Fancy analog widget to select a color value. This widget was contributed by Dmitriy Morozov.

Inherits from *nanogui::Widget*

Public Functions

ColorWheel(*Widget* *parent, **const** *Color* &color = Color (1.0f, 0.0f, 0.0f, 1.0f))
Adds a *ColorWheel* to the specified parent.

Parameters

- `parent`: The *Widget* to add this *ColorWheel* to.
- `color`: The initial color of the *ColorWheel* (default: Red).

std::function<void (**const** *Color*&) > **callback**
constThe callback to execute when a user changes the *ColorWheel* value.

void **setCallback**(**const** std::function<void) **const** *Color*&
> &*callback*Sets the callback to execute when a user changes the *ColorWheel* value.

Color **color**() **const**
The current *Color* this *ColorWheel* has selected.

void **setColor**(**const** *Color* &*color*)
Sets the current *Color* this *ColorWheel* has selected.

virtual Vector2i **preferredSize**(NVGcontext *ctx) **const**
The preferred size of this *ColorWheel*.

virtual void **draw**(NVGcontext *ctx)
Draws the *ColorWheel*.

virtual bool **mouseButtonEvent**(**const** Vector2i &*p*, int *button*, bool *down*, int *modifiers*)
Handles mouse button click events for the *ColorWheel*.

virtual bool **mouseDragEvent**(**const** Vector2i &*p*, **const** Vector2i &*rel*, int *button*, int *modifiers*)
Handles mouse drag events for the *ColorWheel*.

virtual void **save**(*Serializer* &*s*) **const**
Saves the current state of this *ColorWheel* to the specified *Serializer*.

virtual bool **load**(*Serializer* &*s*)
Sets the state of this *ColorWheel* using the specified *Serializer*.

Protected Attributes

float **mHue**
The current Hue in the HSV color model.

float **mWhite**
The implicit Value component of the HSV color model. See implementation *nanogui::ColorWheel::color* for its usage. Valid values are in the range `[0, 1]`.

float **mBlack**
The implicit Saturation component of the HSV color model. See implementation *nanogui::ColorWheel::color* for its usage. Valid values are in the range `[0, 1]`.

Region **mDragRegion**
The current region the mouse is interacting with.

std::function<void (**const** *Color*&)> **mCallback**
The current callback to execute when the color value has changed.

Class ComboBox

- Defined in *File combobox.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::PopupButton` (*Class PopupButton*)

Derived Type

- `public nanogui::detail::FormWidget< T, typename std::is_enum< T >::type >` (*Template Class FormWidget< T, typename std::is_enum< T >::type >*)

Class Documentation

class `nanogui::ComboBox`
Simple combo box widget based on a popup button.

Inherits from *nanogui::PopupButton*

Subclassed by *nanogui::detail::FormWidget< T, typename std::is_enum< T >::type >*

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**
constThe callback to execute for this *ComboBox*.

void **setCallback** (**const** std::function<void) int
> &*callback*Sets the callback to execute for this *ComboBox*.

int **selectedIndex** () **const**
The current index this *ComboBox* has selected.

void **setSelectedIndex** (int *idx*)
Sets the current index this *ComboBox* has selected.

void **setItems** (**const** std::vector<std::string> &*items*, **const** std::vector<std::string> &*itemsShort*)
Sets the items for this *ComboBox*, providing both short and long descriptive lables for each item.

void **setItems** (**const** std::vector<std::string> &*items*)
Sets the items for this *ComboBox*.

const std::vector<std::string> &**items** () **const**
The items associated with this *ComboBox*.

const std::vector<std::string> &**itemsShort** () **const**
The short descriptions associated with this *ComboBox*.

virtual bool **scrollEvent** (**const** Vector2i &*p*, **const** Vector2f &*rel*)
Handles mouse scrolling events for this *ComboBox*.

virtual void **save** (*Serializer* &*s*) **const**
Saves the state of this *ComboBox* to the specified *Serializer*.

virtual bool **load** (*Serializer* &*s*)
Sets the state of this *ComboBox* from the specified *Serializer*.

Protected Attributes

std::vector<std::string> **mItems**
The items associated with this *ComboBox*.

std::vector<std::string> **mItemsShort**
The short descriptions of items associated with this *ComboBox*.

std::function<void (int) > **mCallback**
The callback for this *ComboBox*.

int **mSelectedIndex**
The current index this *ComboBox* has selected.

Template Class FormWidget

- Defined in *File formhelper.h*

Page Contents

Class Documentation

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 *nanogui::ComboBox* for `enum` types:

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

•Inheritance from *nanogui::IntBox* for integral types:

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

•Inheritance from *nanogui::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 *nanogui::CheckBox* for booleans:

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

•Inheritance from *nanogui::TextBox* for strings:

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

•Inheritance from *nanogui::ColorPicker* for *nanogui::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.

Template Class FormWidget< bool, std::true_type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::CheckBox` (*Class CheckBox*)

Class Documentation

template <>
template<>
class `nanogui::detail::FormWidget`<bool, std::true_type>
A specialization for adding a *CheckBox* to a *FormHelper*.

Inherits from *nanogui::CheckBox*

Public Functions

FormWidget(*Widget* *p)
Creates a new *FormWidget* with underlying type *CheckBox*.

void **setValue**(bool *v*)
Pass-through function for *nanogui::CheckBox::setChecked*.

void **setEditable**(bool *e*)
Pass-through function for *nanogui::Widget::setEnabled*.

bool **value**() **const**
Returns the value of *nanogui::CheckBox::checked*.

Template Class FormWidget< Color, std::true_type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::ColorPicker` (*Class ColorPicker*)

Class Documentation

template <>
template<>
class `nanogui::detail::FormWidget`<*Color*, std::true_type>
A specialization for adding a *ColorPicker* to a *FormHelper*.

Inherits from *nanogui::ColorPicker*

Public Functions

FormWidget(*Widget* *p)
Creates a new *FormWidget* with underlying type *ColorPicker*.

void **setValue**(**const** *Color* &c)
Pass-through function for *nanogui::ColorPicker::setColor*.

void **setEditable**(bool e)
Pass-through function for *nanogui::Widget::setEnabled*.

Color **value**() **const**
Returns the value of *nanogui::ColorPicker::color*.

Template Class FormWidget< std::string, std::true_type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::TextBox` (*Class TextBox*)

Class Documentation

template <>
template<>
class `nanogui::detail::FormWidget`<std::string, std::true_type>
A specialization for adding a *TextBox* to a *FormHelper*.

Inherits from *nanogui::TextBox*

Public Functions

FormWidget(*Widget* *p)
Creates a new *FormWidget* with underlying type *TextBox*.

void **setCallback**(**const** std::function<void) **const** std::string&
> &*cb*Pass-through function for *nanogui::TextBox::setCallback*.

Template Class FormWidget< T, typename std::is_enum< T >::type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::ComboBox` (*Class ComboBox*)

Class Documentation

template <**typename** T>
template<>
class `nanogui::detail::`**FormWidget**<T, **typename** std::is_enum<T>::type>
A specialization for adding a *ComboBox* to a *FormHelper*.

Template Parameters

- `T`: The type being used inside the *ComboBox*.

Inherits from *nanogui::ComboBox*

Public Functions

FormWidget(*Widget* *p)
Creates a new *FormWidget* with underlying type *ComboBox*.

T **value**() **const**
Pass-through function for *nanogui::ComboBox::selectedIndex*.

void **setValue**(T *value*)
Pass-through function for *nanogui::ComboBox::setSelectedIndex*.

void **setCallback**(**const** std::function<void) **const** T&
> &*cb*Pass-through function for *nanogui::ComboBox::setCallback*.

void **setEditable**(bool *e*)
Pass-through function for *nanogui::Widget::setEnabled*.

Template Class FormWidget< T, typename std::is_floating_point< T >::type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::FloatBox< T >` (*Template Class FloatBox*)

Class Documentation

template <**typename** T>
template<>
class `nanogui::detail::`**`FormWidget`**<T, **typename** std::is_floating_point<T>::type>
A specialization for adding a *FloatBox* to a *FormHelper*.

Template Parameters

- `T`: The **floating point** type being used for the *FloatBox*.

Inherits from *nanogui::FloatBox< T >*

Public Functions

`FormWidget`(*Widget* *p)
Creates a new *FormWidget* with underlying type *FloatBox*.

Template Class FormWidget< T, typename std::is_integral< T >::type >

- Defined in *File formhelper.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::IntBox< T >` (*Template Class IntBox*)

Class Documentation

template <typename T>
template<>
class `nanogui::detail::FormWidget`<T, **typename** std::is_integral<T>::type>
A specialization for adding an *IntBox* to a *FormHelper*.

Template Parameters

- `T`: The **integral** type being used for the *IntBox*.

Inherits from *nanogui::IntBox< T >*

Public Functions

`FormWidget`(*Widget *p*)
Creates a new *FormWidget* with underlying type *IntBox*.

Template Class FloatBox

- Defined in *File textbox.h*

Inheritance Relationships

Base Type

- `public nanogui::TextBox` (*Class TextBox*)

Class Documentation

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*

Page Contents

Class Documentation

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* = trueAdd 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*)
Set the active *Window* instance.

void **setFixedSize**(**const** Vector2i &*fw*)
Specify a fixed size for newly added widgets.

Vector2i **fixedSize**()
The current fixed size being used for newly added widgets.

const std::string &**groupFontName**() **const**
The font name being used for group headers.

void **setGroupFontName**(**const** std::string &*name*)
Sets the font name to be used for group headers.

const std::string &**labelFontName**() **const**
The font name being used for labels.

void **setLabelFontName**(**const** std::string &*name*)
Sets the font name being used for labels.

int **groupFontSize**() **const**
The size of the font being used for group headers.

void **setGroupFontSize**(int *value*)
Sets the size of the font being used for group headers.

int **labelFontSize**() **const**
The size of the font being used for labels.

void **setLabelFontSize**(int *value*)
Sets the size of the font being used for labels.

int **widgetFontSize**() **const**
The size of the font being used for non-group / non-label widgets.

void **setWidgetFontSize**(int *value*)
Sets the size of the font being used for non-group / non-label widgets.

Protected Attributes

ref<*Screen*> **mScreen**
A reference to the *nanogui::Screen* this *FormHelper* is assisting.

ref<*Window*> **mWindow**
A reference to the *nanogui::Window* this *FormHelper* is controlling.

ref<*AdvancedGridLayout*> **mLayout**
A reference to the *nanogui::AdvancedGridLayout* this *FormHelper* is using.

std::vector<std::function<void()>> **mRefreshCallbacks**
The callbacks associated with all widgets this *FormHelper* is managing.

std::string **mGroupFontName** = "sans-bold"
The group header font name.

std::string **mLabelFontName** = "sans"
The label font name.

Vector2i **mFixedSize** = *Vector2i*(0, 20)
The fixed size for newly added widgets.

int **mGroupFontSize** = 20
The font size for group headers.

int **mLabelFontSize** = 16
The font size for labels.

int **mWidgetFontSize** = 16
The font size for non-group / non-label widgets.

int **mPreGroupSpacing** = 15
The spacing used **before** new groups.

int **mPostGroupSpacing** = 5
The spacing used **after** each group.

int **mVariableSpacing** = 5
The spacing between all other widgets.

Class GLCanvas

- Defined in *File glcanvas.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

class `nanogui::GLCanvas`
Canvas widget for rendering OpenGL content. This widget was contributed by Jan Winkler.

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 `nanogui::GLCanvas::drawGL()` in subclasses to provide custom drawing code. See *Example 4*.

Inherits from *nanogui::Widget*

Public Functions

GLCanvas(*Widget* *parent)
Creates a *GLCanvas* attached to the specified parent.

Parameters

- `parent`: The *Widget* to attach this *GLCanvas* to.

const *Color* &**backgroundColor**() **const**
Returns the background color.

void **setBackgroundColor**(**const** *Color* &*backgroundColor*)
Sets 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 the state of this *GLCanvas* to the specified *Serializer*.

virtual bool **load**(*Serializer* &*s*)
Set the state of this *GLCanvas* from the specified *Serializer*.

Protected Functions

void **drawWidgetBorder**(NVGcontext *ctx) **const**
Internal helper function for drawing the widget border.

Protected Attributes

Color **mBackgroundColor**
The background color (what is used with `glClearColor`).

bool **mDrawBorder**
Whether to draw the widget border or not.

Class GLFramebuffer

- Defined in *File glutil.h*

Page Contents

Class Documentation

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*

Page Contents

Nested Relationships

Nested Types

- *Struct GLShader::Buffer*

Class Documentation

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 unitialized 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>
void **setUniform**(**const** std::string &*name*, **const** Eigen::Transform<T, 3, 3> &*affine*, bool *warn* = true)
Initialize a uniform parameter with a 3x3 affine transform (float)

template <**typename** T>
void **setUniform**(**const** std::string &*name*, **const** Eigen::Matrix<T, 3, 3> &*mat*, bool *warn* = true)
Initialize a uniform parameter with a 3x3 matrix (float)

template <**typename** T>
void **setUniform**(**const** std::string &*name*, **const** Eigen::Transform<T, 2, 2> &*affine*, bool *warn* = true)
Initialize a uniform parameter with a 2x2 affine transform (float)

void **setUniform**(**const** std::string &*name*, bool *value*, bool *warn* = true)
Initialize a uniform parameter with a boolean value.

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*

Page Contents

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

- *Inheritance Relationships*

Inheritance Relationships

Base Type

- `public nanogui::Layout` (*Class Layout*)

Class Documentation

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.

Parameters

- `orientation`: The fixed dimension of this *GridLayout*.
- `resolution`: The number of rows or columns in the grid (depending on the Orientation).
- `alignment`: How widgets should be aligned within each grid cell.
- `margin`: The amount of spacing to add around the border of the grid.
- `spacing`: The amount of spacing between widgets added to the grid.

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

The Orientation of this *GridLayout*.

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

Sets the Orientation of this *GridLayout*.

int **resolution**() **const**

The number of rows or columns (depending on the Orientation) of this *GridLayout*.

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

Sets the number of rows or columns (depending on the Orientation) of this *GridLayout*.

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

The spacing at the specified axis (row or column number, depending on the Orientation).

void **setSpacing**(int *axis*, int *spacing*)
Sets the spacing for a specific axis.

void **setSpacing**(int *spacing*)
Sets the spacing for all axes.

int **margin**() **const**
The margin around this *GridLayout*.

void **setMargin**(int *margin*)
Sets the margin of this *GridLayout*.

Alignment **alignment**(int *axis*, int *item*) **const**
The Alignment of the specified axis (row or column number, depending on the Orientation) at the specified index of that row or column.

void **setColAlignment**(*Alignment value*)
Sets the Alignment of the columns.

void **setRowAlignment**(*Alignment value*)
Sets the Alignment of the rows.

void **setColAlignment**(**const** std::vector<*Alignment*> &*value*)
Use this to set variable Alignment for columns.

void **setRowAlignment**(**const** std::vector<*Alignment*> &*value*)
Use this to set variable Alignment for rows.

virtual Vector2i **preferredSize**(NVGcontext **ctx*, **const** *Widget* **widget*) **const**
See *Layout::preferredSize*.

virtual void **performLayout**(NVGcontext **ctx*, *Widget* **widget*) **const**
See *Layout::performLayout*.

Protected Functions

void **computeLayout**(NVGcontext **ctx*, **const** *Widget* **widget*, std::vector<int> **grid*) **const**
Compute the maximum row and column sizes.

Protected Attributes

Orientation **mOrientation**
The Orientation defining this *GridLayout*.

Alignment **mDefaultAlignment**[2]
The default Alignment for this *GridLayout*.

std::vector<*Alignment*> **mAlignment**[2]
The actual Alignment being used.

int **mResolution**
The number of rows or columns before starting a new one, depending on the Orientation.

Vector2i **mSpacing**
The spacing used for each dimension.

int **mMargin**
The margin around this *GridLayout*.

Class GroupLayout

- Defined in *File layout.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Layout` (*Class Layout*)

Class Documentation

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)

Creates a *GroupLayout*.

Parameters

- `margin`: The margin around the widgets added.
- `spacing`: The spacing between widgets added.
- `groupSpacing`: The spacing between groups (groups are defined by each *Label* added).
- `groupIndent`: The amount to indent widgets in a group (underneath a *Label*).

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

The margin of this *GroupLayout*.

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

Sets the margin of this *GroupLayout*.

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

The spacing between widgets of this *GroupLayout*.

void **setSpacing**(int *spacing*)
Sets the spacing between widgets of this *GroupLayout*.

int **groupIndent**() **const**
The indent of widgets in a group (underneath a *Label*) of this *GroupLayout*.

void **setGroupIndent**(int *groupIndent*)
Sets the indent of widgets in a group (underneath a *Label*) of this *GroupLayout*.

int **groupSpacing**() **const**
The spacing between groups of this *GroupLayout*.

void **setGroupSpacing**(int *groupSpacing*)
Sets the spacing between groups of this *GroupLayout*.

virtual Vector2i **preferredSize**(NVGcontext **ctx*, **const** *Widget* **widget*) **const**
See *Layout::preferredSize*.

virtual void **performLayout**(NVGcontext **ctx*, *Widget* **widget*) **const**
See *Layout::performLayout*.

Protected Attributes

int **mMargin**
The margin of this *GroupLayout*.

int **mSpacing**
The spacing between widgets of this *GroupLayout*.

int **mGroupSpacing**
The spacing between groups of this *GroupLayout*.

int **mGroupIndent**
The indent amount of a group under its defining *Label* of this *GroupLayout*.

Class ImagePanel

- Defined in *File imagepanel.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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)

Template Class IntBox

- Defined in *File textbox.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::TextBox` (*Class TextBox*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Object` (*Class Object*)

Derived Types

- `public nanogui::AdvancedGridLayout` (*Class AdvancedGridLayout*)
- `public nanogui::BoxLayout` (*Class BoxLayout*)
- `public nanogui::GridLayout` (*Class GridLayout*)
- `public nanogui::GroupLayout` (*Class GroupLayout*)

Class Documentation

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

Performs any and all resizing applicable.

Parameters

- `ctx`: The `NanoVG` context being used for drawing.
- `widget`: The *Widget* this layout is controlling sizing for.

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

The preferred size for this layout.

Return The preferred size, accounting for things such as spacing, padding for icons, etc.

Parameters

- `ctx`: The `NanoVG` context being used for drawing.
- `widget`: The *Widget* this layout's preferred size is considering.

Protected Functions

virtual ~Layout()
Default destructor (exists for inheritance).

Class MessageDialog

- Defined in *File messagedialog.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Window` (*Class Window*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Derived Types

- `public nanogui::Layout` (*Class Layout*)
- `public nanogui::Theme` (*Class Theme*)
- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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 deconstructor. (Will only be called by *ref*)

Class Popup

- Defined in *File popup.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Window` (*Class Window*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Button` (*Class Button*)

Derived Types

- `public nanogui::ColorPicker` (*Class ColorPicker*)
- `public nanogui::ComboBox` (*Class ComboBox*)

Class Documentation

class `nanogui::PopupButton`
Button which launches a popup widget.

Remark This class overrides *nanogui::Widget::mIconExtraScale* to be `0.8f`, which affects all subclasses of this *Widget*. Subclasses must explicitly set a different value if needed (e.g., in their constructor).

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)
Responsible for drawing the *Button*.

virtual Vector2i **preferredSize**(NVGcontext *ctx) **const**
The preferred size of this *Button*.

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

virtual void **save**(*Serializer* &s) **const**
Saves the state of this *Button* provided the given *Serializer*.

virtual bool **load**(*Serializer* &s)
Sets the state of this *Button* provided the given *Serializer*.

Protected Attributes

Popup ***mPopup**

int **mChevronIcon**

Class ProgressBar

- Defined in *File progressbar.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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**

Template Class ref

- Defined in *File object.h*

Page Contents

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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**
constSet 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 initalize() 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 **mProcessEvents**

Color **mBackground**

std::string **mCaption**

bool **mShutdownGLFWOnDestruct**

bool **mFullscreen**

std::function<void (Vector2i) > **mResizeCallback**

Class Serializer

- Defined in *File core.h*

Page Contents

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

Nested Relationships

Nested Types

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

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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 (tabsCount) 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*

Page Contents

Nested Relationships

This class is a nested type of *Class TabHeader*.

Nested Types

- *Struct TabButton::StringView*

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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 the 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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Derived Types

- `public nanogui::FloatBox< T >` (*Template Class FloatBox*)
- `public nanogui::IntBox< T >` (*Template Class IntBox*)
- `public nanogui::detail::FormWidget< std::string, std::true_type >` (*Template Class FormWidget< std::string, std::true_type >*)
- `public nanogui::FloatBox< Scalar >` (*Template Class FloatBox*)
- `public nanogui::IntBox< Scalar >` (*Template Class IntBox*)

Class Documentation

class `nanogui::TextBox`
Fancy text box with builtin regular expression-based validation.

Remark This class overrides *nanogui::Widget::mIconExtraScale* to be `0.8f`, which affects all subclasses of this *Widget*. Subclasses must explicitly set a different value if needed (e.g., in their constructor).

Inherits from *nanogui::Widget*

Subclassed by *nanogui::FloatBox< T >*, *nanogui::IntBox< T >*, *nanogui::detail::FormWidget< std::string, std::true_type >*, *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**
constThe callback to execute when the value of this *TextBox* has changed.

void **setCallback**(**const** std::function<bool) **const** std::string &str
> &*callback*Sets the callback to execute when the value of this *TextBox* has changed.

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 **mMouseDragPos**

int **mMouseDownModifier**

float **mTextOffset**

double **mLastClick**

Class Theme

- Defined in *File theme.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Object` (*Class Object*)

Class Documentation

class `nanogui::Theme`

Storage class for basic theme-related properties.

Inherits from *nanogui::Object*

Public Functions

Theme (NVGcontext **ctx*)

Public Members

int **mFontNormal**

The standard font face (default: `"sans"` from `resources/roboto_regular.ttf`).

int **mFontBold**

The bold font face (default: `"sans-bold"` from `resources/roboto_regular.ttf`).

int **mFontIcons**

The icon font face (default: `"icons"` from `resources/entypo.ttf`).

float **mIconScale**

The amount of scaling that is applied to each icon to fit the size of NanoGUI widgets. The default value is `0.77f`, setting to e.g. higher than `1.0f` is generally discouraged.

int **mStandardFontSize**

The font size for all widgets other than buttons and textboxes (default: `16`).

int **mButtonFontSize**

The font size for buttons (default: `20`).

int **mTextBoxFontSize**

The font size for text boxes (default: `20`).

int **mWindowCornerRadius**

Rounding radius for *Window* widget corners (default: `2`).

int **mWindowHeaderHeight**

Default size of *Window* widget titles (default: `30`).

int **mWindowDropShadowSize**

Size of drop shadow rendered behind the *Window* widgets (default: `10`).

int **mButtonCornerRadius**

Rounding radius for *Button* (and derived types) widgets (default: `2`).

float **mTabBorderWidth**

The border width for *TabHeader* widgets (default: `0.75f`).

int **mTabInnerMargin**
The inner margin on a *TabHeader* widget (default: `5`).

int **mTabMinButtonWidth**
The minimum size for buttons on a *TabHeader* widget (default: `20`).

int **mTabMaxButtonWidth**
The maximum size for buttons on a *TabHeader* widget (default: `160`).

int **mTabControlWidth**
Used to help specify what lies "in bound" for a *TabHeader* widget (default: `20`).

int **mTabButtonHorizontalPadding**
The amount of horizontal padding for a *TabHeader* widget (default: `10`).

int **mTabButtonVerticalPadding**
The amount of vertical padding for a *TabHeader* widget (default: `2`).

Color **mDropShadow**
The color of the drop shadow drawn behind widgets (default: intensity=`0`, alpha=`128`; see *nanogui::Color::Color(int,int)*).

Color **mTransparent**
The transparency color (default: intensity=`0`, alpha=`0`; see *nanogui::Color::Color(int,int)*).

Color **mBorderDark**
The dark border color (default: intensity=`29`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mBorderLight**
The light border color (default: intensity=`92`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mBorderMedium**
The medium border color (default: intensity=`35`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mTextColor**
The text color (default: intensity=`255`, alpha=`160`; see *nanogui::Color::Color(int,int)*).

Color **mDisabledTextColor**
The disable dtext color (default: intensity=`255`, alpha=`80`; see *nanogui::Color::Color(int,int)*).

Color **mTextColorShadow**
The text shadow color (default: intensity=`0`, alpha=`160`; see *nanogui::Color::Color(int,int)*).

Color **mIconColor**
The icon color (default: *nanogui::Theme::mTextColor*).

Color **mButtonGradientTopFocused**
The top gradient color for buttons in focus (default: intensity=`64`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mButtonGradientBotFocused**
The bottom gradient color for buttons in focus (default: intensity=`48`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mButtonGradientTopUnfocused**
The top gradient color for buttons not in focus (default: intensity=`74`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mButtonGradientBotUnfocused**
The bottom gradient color for buttons not in focus (default: intensity=`58`, alpha=`255`; see *nanogui::Color::Color(int,int)*).

Color **mButtonGradientTopPushed**
The top gradient color for buttons currently pushed (default: intensity=41, alpha=255; see *nanogui::Color::Color(int,int)*).

Color **mButtonGradientBotPushed**
The bottom gradient color for buttons currently pushed (default: intensity=29, alpha=255; see *nanogui::Color::Color(int,int)*).

Color **mWindowFillUnfocused**
The fill color for a *Window* that is not in focus (default: intensity=43, alpha=230; see *nanogui::Color::Color(int,int)*).

Color **mWindowFillFocused**
The fill color for a *Window* that is in focus (default: intensity=45, alpha=230; see *nanogui::Color::Color(int,int)*).

Color **mWindowTitleUnfocused**
The title color for a *Window* that is not in focus (default: intensity=220, alpha=160; see *nanogui::Color::Color(int,int)*).

Color **mWindowTitleFocused**
The title color for a *Window* that is in focus (default: intensity=255, alpha=190; see *nanogui::Color::Color(int,int)*).

Color **mWindowHeaderGradientTop**
The top gradient color for *Window* headings (default: *nanogui::Theme::mButtonGradientTopUnfocused*).

Color **mWindowHeaderGradientBot**
The bottom gradient color for *Window* headings (default: *nanogui::Theme::mButtonGradientBotUnfocused*).

Color **mWindowHeaderSepTop**
The *Window* header top separation color (default: *nanogui::Theme::mBorderLight*).

Color **mWindowHeaderSepBot**
The *Window* header bottom separation color (default: *nanogui::Theme::mBorderDark*).

Color **mWindowPopup**
The popup window color (default: intensity=50, alpha=255; see *nanogui::Color::Color(int,int)*)).

Color **mWindowPopupTransparent**
The transparent popup window color (default: intensity=50, alpha=0; see *nanogui::Color::Color(int,int)*)).

int **mCheckBoxIcon**
Icon to use for *CheckBox* widgets (default: ENTYPO_ICON_CHECK).

int **mMessageInformationIcon**
Icon to use for informational *MessageDialog* widgets (default: ENTYPO_ICON_INFO_WITH_CIRCLE).

int **mMessageQuestionIcon**
Icon to use for interrogative *MessageDialog* widgets (default: ENTYPO_ICON_HELP_WITH_CIRCLE).

int **mMessageWarningIcon**
Icon to use for warning *MessageDialog* widgets (default: ENTYPO_ICON_WARNING).

int **mMessageAltButtonIcon**
Icon to use on *MessageDialog* alt button (default: ENTYPO_ICON_CIRCLE_WITH_CROSS).

int **mMessagePrimaryButtonIcon**
Icon to use on *MessageDialog* primary button (default: ENTYPO_ICON_CHECK).

int **mPopupChevronRightIcon**
Icon to use for *PopupButton* widgets opening to the right (default: ENTYPO_ICON_CHEVRON_RIGHT).

int **mPopupChevronLeftIcon**
Icon to use for *PopupButton* widgets opening to the left (default: `ENTYPO_ICON_CHEVRON_LEFT`).

int **mTabHeaderLeftIcon**
Icon to indicate hidden tabs to the left on a *TabHeader* (default: `ENTYPO_ICON_ARROW_BOLD_LEFT`).

int **mTabHeaderRightIcon**
Icon to indicate hidden tabs to the right on a *TabHeader* (default: `ENTYPO_ICON_ARROW_BOLD_RIGHT`).

int **mTextBoxUpIcon**
Icon to use when a *TextBox* has an up toggle (e.g. *IntBox*) (default: `ENTYPO_ICON_CHEVRON_UP`).

int **mTextBoxDownIcon**
Icon to use when a *TextBox* has a down toggle (e.g. *IntBox*) (default: `ENTYPO_ICON_CHEVRON_DOWN`).

Protected Functions

virtual ~Theme()
Default destructor does nothing; allows for inheritance.

Class ToolButton

- Defined in *File toolbutton.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Button` (*Class Button*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public std::vector< uint8_t >`

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Class Documentation

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*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Object` (*Class Object*)

Derived Types

- `public nanogui::Button` (*Class Button*)
- `public nanogui::CheckBox` (*Class CheckBox*)
- `public nanogui::ColorWheel` (*Class ColorWheel*)
- `public nanogui::GLCanvas` (*Class GLCanvas*)
- `public nanogui::Graph` (*Class Graph*)
- `public nanogui::ImagePanel` (*Class ImagePanel*)
- `public nanogui::ImageView` (*Class ImageView*)
- `public nanogui::Label` (*Class Label*)
- `public nanogui::ProgressBar` (*Class ProgressBar*)
- `public nanogui::Screen` (*Class Screen*)
- `public nanogui::Slider` (*Class Slider*)
- `public nanogui::StackedWidget` (*Class StackedWidget*)
- `public nanogui::TabHeader` (*Class TabHeader*)
- `public nanogui::TabWidget` (*Class TabWidget*)
- `public nanogui::TextBox` (*Class TextBox*)
- `public nanogui::VScrollPanel` (*Class VScrollPanel*)
- `public nanogui::Window` (*Class Window*)

Class Documentation

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.

Screen ***screen**()
Walk up the hierarchy and return the parent screen.

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.

float **iconExtraScale**() **const**
The amount of extra scaling applied to *icon* fonts. See *nanogui::Widget::mIconExtraScale*.

void **setIconExtraScale**(float *scale*)
Sets the amount of extra scaling applied to *icon* fonts. See *nanogui::Widget::mIconExtraScale*.

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.

float **icon_scale**() **const**
Convenience definition for subclasses to get the full icon scale for this class of *Widget*. It simple returns the value `mTheme->mIconScale * this->mIconExtraScale`.

Remark See also: *nanogui::Theme::mIconScale* and *nanogui::Widget::mIconExtraScale*. This tiered scaling strategy may not be appropriate with fonts other than `entypo.ttf`.

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**
Whether or not this *Widget* is currently visible. When a *Widget* is not currently visible, no time is wasted executing its drawing method.

bool **mEnabled**
Whether or not this *Widget* is currently enabled. Various different kinds of derived types use this to determine whether or not user input will be accepted. For example, when `mEnabled == false`, the state of a *CheckBox* cannot be changed, or a *TextBox* will not allow new input.

bool **mFocused**

bool **mMouseFocus**

std::string **mTooltip**

int **mFontSize**

float **mIconExtraScale**
The amount of extra icon scaling used in addition the the theme's default icon font scale. Default value is `1.0`, which implies that *nanogui::Widget::icon_scale* simply returns the value of *nanogui::Theme::mIconScale*.

Most widgets do not need extra scaling, but some (e.g., *CheckBox*, *TextBox*) need to adjust the *Theme*'s default icon scaling (*nanogui::Theme::mIconScale*) to properly display icons within their bounds (upscale, or downscale).

Note: When using `nvgFontSize` for icons in subclasses, make sure to call the *`nanogui::Widget::icon_scale()`* function. Expected usage when *drawing* icon fonts is something like:

```
virtual void draw(NVGcontext *ctx) {
    // fontSize depends on the kind of Widget.  Search for `FontSize`
    // in the Theme class (e.g., standard vs button)
    float ih = fontSize;
    // assuming your Widget has a declared `mIcon`
    if (nvgIsFontIcon(mIcon)) {
        ih *= icon_scale();
        nvgFontFace(ctx, "icons");
        nvgFontSize(ctx, ih);
        /// remaining drawing code (see button.cpp for more)
    }
}
```

Cursor **mCursor**

Class Window

- Defined in *File window.h*

Page Contents

Inheritance Relationships

Base Type

- `public nanogui::Widget` (*Class Widget*)

Derived Types

- `public nanogui::MessageDialog` (*Class MessageDialog*)
- `public nanogui::Popup` (*Class Popup*)

Class Documentation

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 model 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 Alignment

- Defined in *File layout.h*

Enum Documentation

enum nanogui::**Alignment**

The different kinds of alignments a layout can perform.

Values:

Minimum = 0

Take only as much space as is required.

Middle

Center align.

Maximum

Take as much space as is allowed.

Fill

Fill according to preferred sizes.

Enum Cursor

- Defined in *File common.h*

Enum Documentation

enum nanogui::**Cursor**

Cursor shapes available to use in GLFW. Shape of actual cursor determined by Operating System.

Values:

Arrow = 0

The arrow cursor.

IBeam

The I-beam cursor.

Crosshair

The crosshair cursor.

Hand
The hand cursor.

HResize
The horizontal resize cursor.

VResize
The vertical resize cursor.

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

Enum Orientation

- Defined in *File layout.h*

Enum Documentation

enum nanogui::**Orientation**
The direction of data flow for a layout.

Values:

Horizontal = 0
Layout expands on horizontal axis.

Vertical
Layout expands on vertical axis.

Functions

Function __nanogui_get_image

- Defined in *File common.h*

Function Documentation

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

Function active

- Defined in *File common.h*

Function Documentation

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

Function chdir_to_bundle_parent

- Defined in *File common.h*

Function Documentation

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*

Function Documentation

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*

Function Documentation

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*

Function Documentation

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*

Function Documentation

void `nanogui::leave()`
Request the application main loop to terminate (e.g. if you detached mainloop).

Function loadImageDirectory

- Defined in *File common.h*

Function Documentation

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*

Function Documentation

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*

Function Documentation

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 nvgIsFontIcon

- Defined in *File opengl.h*

Function Documentation

bool `nanogui::nvgIsFontIcon`(int *value*)
Determine whether an icon ID is a font-based icon (e.g. from `entypo.ttf`).

See *`nanogui::nvgIsImageIcon()`* for details.

Return Whether or not this is a font icon (from `entypo.ttf`).

Parameters

- `value`: The integral value of the icon.

Function nvgIsImageIcon

- Defined in *File opengl.h*

Function Documentation

bool `nanogui::nvgIsImageIcon`(int *value*)

Determine whether an icon ID is a texture loaded via `nvgImageIcon`.

The implementation defines all `value < 1024` as image icons, and everything `>= 1024` as an Entypo icon (see *File entypo.h*). The value `1024` exists to provide a generous buffer on how many images may have been loaded by NanoVG.

Return Whether or not this is an image icon.

Parameters

- `value`: The integral value of the icon.

Function ortho

- Defined in *File glutil.h*

Function Documentation

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*

Function Documentation

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*

Function Documentation

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*

Function Documentation

void `nanogui::shutdown`()

Static shutdown; should be called before the application terminates.

Function translate

- Defined in *File glutil.h*

Function Documentation

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*

Function Documentation

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*

Function Documentation

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*

Define Documentation

`GL_HALF_FLOAT`

Ensures that `GL_HALF_FLOAT` is defined properly for all platforms.

Define NAMESPACE_BEGIN

- Defined in *File common.h*

Define Documentation

NAMESPACE_BEGIN(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*

Define Documentation

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*

Define Documentation

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*

Define Documentation

NANOGUI_FORCE_DISCRETE_GPU

On Windows, exports `AmdPowerXpressRequestHighPerformance` and `NvOptimusEnablement` as `1`.

Define NANOGUI_LAYOUT_OVERLOADS

- Defined in *File python.h*

Define Documentation

NANOGUI_LAYOUT_OVERLOADS (Parent)
Provides a `PYBIND11_OVERLOAD` for any relevant Layout items that need to be bound.

Define NANOGUI_SCREEN_OVERLOADS

- Defined in *File python.h*

Define Documentation

NANOGUI_SCREEN_OVERLOADS (Parent)
Provides a `PYBIND11_OVERLOAD` for any relevant Screen items that need to be bound.

Define NANOGUI_SNPRINTF

- Defined in *File compat.h*

Define Documentation

NANOGUI_SNPRINTF
Platform dependent snprintf (`_snprintf` for MSVC, `snprintf` otherwise).

Define NANOGUI_WIDGET_OVERLOADS

- Defined in *File python.h*

Define Documentation

NANOGUI_WIDGET_OVERLOADS (Parent)
Provides a `PYBIND11_OVERLOAD` for any relevant Widget items that need to be bound.

Define nvgImageIcon

- Defined in *File common.h*

Define Documentation

nvgImageIcon (ctx, name)
Convenience function for instanting a PNG icon from the application's data segment (via bin2c)

Define SYSTEM_COMMAND_MOD

- Defined in *File common.h*

Define Documentation

SYSTEM_COMMAND_MOD
If on OSX, maps to `GLFW_MOD_SUPER`. Otherwise, maps to `GLFW_MOD_CONTROL`.

Typedefs

Typedef nanogui::Matrix3f

- Defined in *File common.h*

Typedef Documentation

using nanogui::Matrix3f = typedef Eigen::Matrix3f
Type alias to allow `Eigen::Matrix3f` to be used as `nanogui::Matrix3f`.

Typedef nanogui::Matrix4f

- Defined in *File common.h*

Typedef Documentation

using nanogui::Matrix4f = typedef Eigen::Matrix4f
Type alias to allow `Eigen::Matrix4f` to be used as `nanogui::Matrix4f`.

Typedef nanogui::MatrixXf

- Defined in *File common.h*

Typedef Documentation

using nanogui::MatrixXf = typedef Eigen::MatrixXf
Type alias to allow `Eigen::MatrixXf` to be used as `nanogui::MatrixXf`.

Typedef nanogui::MatrixXu

- Defined in *File common.h*

Typedef Documentation

using nanogui::MatrixXu = typedef Eigen::Matrix<uint32_t, Eigen::Dynamic, Eigen::Dynamic>
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`.

Typedef nanogui::Vector2f

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector2f = typedef Eigen::Vector2f
Type alias to allow `Eigen::Vector2f` to be used as `nanogui::Vector2f`.

Typedef nanogui::Vector2i

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector2i = typedef Eigen::Vector2i
Type alias to allow `Eigen::Vector2i` to be used as `nanogui::Vector2i`.

Typedef nanogui::Vector3f

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector3f = typedef Eigen::Vector3f
Type alias to allow `Eigen::Vector3f` to be used as `nanogui::Vector3f`.

Typedef nanogui::Vector3i

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector3i = typedef Eigen::Vector3i
Type alias to allow `Eigen::Vector3i` to be used as `nanogui::Vector3i`.

Typedef nanogui::Vector4f

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector4f = typedef Eigen::Vector4f
Type alias to allow `Eigen::Vector4f` to be used as `nanogui::Vector4f`.

Typedef nanogui::Vector4i

- Defined in *File common.h*

Typedef Documentation

using nanogui::Vector4i = typedef Eigen::Vector4i
Type alias to allow `Eigen::Vector4i` to be used as `nanogui::Vector4i`.

Typedef nanogui::VectorXf

- Defined in *File common.h*

Typedef Documentation

using nanogui::VectorXf = typedef Eigen::VectorXf
Type alias to allow `Eigen::VectorXf` to be used as `nanogui::VectorXf`.

Directories

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

Defines the [Normal/Toggle/Radio/Popup] *Class Button* widget.

Page Contents

Definition (nanogui/button.h)

Program Listing for File button.h

- Return to documentation for *File button.h*

```
/*
    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),
        RadioButton  = (1 << 1),
        ToggleButton = (1 << 2),
        PopupButton  = (1 << 3)
    };

    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*

File checkbox.h

Two-state check box Widget.

Page Contents

- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`nanogui/checkbox.h`)

Program Listing for File checkbox.h

- Return to documentation for *File checkbox.h*

```
/*
    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;

    bool 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

Push button with a popup to tweak a color value. This widget was contributed by Christian Schueller.

Page Contents

- *Namespaces*
- *Classes*

Definition (`nanogui/colorpicker.h`)

Program Listing for File colorpicker.h

- Return to documentation for *File colorpicker.h*

```
/*
    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;
        mCallback(backgroundColor());
    }

    std::function<void(const Color &)> finalCallback() const { return mFinalCallback;␣
↪}

    void setFinalCallback(const std::function<void(const Color &)> &callback) {␣
↪mFinalCallback = callback; }

    Color color() const;

    void setColor(const Color& color);

    const std::string &pickButtonCaption() { return mPickButton->caption(); }

    void setPickButtonCaption(const std::string &caption) { mPickButton->
↪setCaption(caption); }

    const std::string &resetButtonCaption() { return mResetButton->caption(); }

    void setResetButtonCaption(const std::string &caption) { mResetButton->
↪setCaption(caption); }
```

```
protected:
    std::function<void(const Color &)> mCallback;

    std::function<void(const Color &)> mFinalCallback;

    ColorWheel *mColorWheel;

    Button *mPickButton;

    Button *mResetButton;

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

Fancy analog widget to select a color value. This widget was contributed by Dmitriy Morozov.

Page Contents

Definition (nanogui/colorwheel.h)

Program Listing for File colorwheel.h

- Return to documentation for *File colorwheel.h*

```
/*
    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:
    // Used to describe where the mouse is interacting
    enum Region {
        None = 0,
        InnerTriangle = 1,
        OuterCircle = 2,
        Both = 3
    };

    // Converts a specified hue (with saturation = value = 1) to RGB space.
```

```
    Color hue2rgb(float h) const;

    // Manipulates the positioning of the different regions of the ColorWheel.
    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*

File combobox.h

Simple combo box widget based on a popup button.

Page Contents

- *Namespaces*
- *Classes*

Definition (`nanogui/combobox.h`)

Program Listing for File combobox.h

- Return to documentation for *File combobox.h*

```
/*
    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;

    std::vector<std::string> 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

Common definitions used by NanoGUI.

Page Contents

- *Enums*
- *Functions*
- *Defines*
- *Typedefs*

Definition (`nanogui/common.h`)

Program Listing for File common.h

- Return to documentation for *File common.h*

```
/*
    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

#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
#  pragma warning(disable : 4127) // warning C4127: conditional expression is constant
#endif

#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 {
#endif
#if !defined(NAMESPACE_END) || defined(DOXYGEN_DOCUMENTATION_BUILD)

    #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) && !defined(DOXYGEN_DOCUMENTATION_BUILD)
#define NANOGUI_FORCE_DISCRETE_GPU() \
    extern "C" { \
        __declspec(dllexport) int AmdPowerXpressRequestHighPerformance = 1; \
        __declspec(dllexport) int NvOptimusEnablement = 1; \
    }
#else

#define NANOGUI_FORCE_DISCRETE_GPU()
#endif

// These will produce broken links in the docs build
#if !defined(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
#if defined(__APPLE__) || defined(DOXYGEN_DOCUMENTATION_BUILD)
    #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 Vector2f = Eigen::Vector2f;
using Vector3f = Eigen::Vector3f;
using Vector4f = Eigen::Vector4f;
using Vector2i = Eigen::Vector2i;
using Vector3i = Eigen::Vector3i;
using Vector4i = Eigen::Vector4i;
using Matrix3f = Eigen::Matrix3f;
using Matrix4f = Eigen::Matrix4f;
using VectorXf = Eigen::VectorXf;
using MatrixXf = Eigen::MatrixXf;

using MatrixXu = Eigen::Matrix<uint32_t, Eigen::Dynamic, Eigen::Dynamic>;

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
#ifndef 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);

#if defined(__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 nvgImageIcon*
- *Define SYSTEM_COMMAND_MOD*

Typedefs

- *Typedef nanogui::Matrix3f*
- *Typedef nanogui::Matrix4f*
- *Typedef nanogui::MatrixXf*
- *Typedef nanogui::MatrixXu*
- *Typedef nanogui::Vector2f*
- *Typedef nanogui::Vector2i*
- *Typedef nanogui::Vector3f*

- *Typedef nanogui::Vector3i*
- *Typedef nanogui::Vector4f*
- *Typedef nanogui::Vector4i*
- *Typedef nanogui::VectorXf*

File compat.h

Compatibility layer for `snprintf` across platforms.

Page Contents

Definition (nanogui/compat.h)

Program Listing for File compat.h

- Return to documentation for *File compat.h*

```
/*
    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>

#if defined(_MSC_VER) && !defined(DOXYGEN_DOCUMENTATION_BUILD)
    #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

Page Contents

Definition (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
#ifndef 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;
        }
    }
};

#endif // DOXYGEN_SHOULD_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

- *Template Struct serialization_helper*
- *Template Struct serialization_traits*
- *Class Serializer*

File entypo.h

This is a list of icon codes for the entypo.ttf font by Daniel Bruce.

Page Contents

Definition (nanogui/entypo.h)

Program Listing for File entypo.h

- Return to documentation for *File entypo.h*

```
/*
    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.
 */
```

```
/* Developer note: need to make a change to this file?
 * Please raise an Issue on GitHub describing what needs to change.  This file
 * was generated, so the scripts that generated it needs to update as well.
 */

#pragma once

// prevent individual pages from being generated for all of these
#if !defined(DOXYGEN_SHOULD_SKIP_THIS)

#define ENTYPO_ICON_500PX                       0x0000F100
#define ENTYPO_ICON_500PX_WITH_CIRCLE           0x0000F101
#define ENTYPO_ICON_ADD_TO_LIST                 0x0000F102
#define ENTYPO_ICON_ADD_USER                    0x0000F103
#define ENTYPO_ICON_ADDRESS                     0x0000F104
#define ENTYPO_ICON_ADJUST                      0x0000F105
#define ENTYPO_ICON_AIR                         0x0000F106
#define ENTYPO_ICON_AIRCRAFT                    0x0000F107
#define ENTYPO_ICON_AIRCRAFT_LANDING            0x0000F108
#define ENTYPO_ICON_AIRCRAFT_TAKE_OFF           0x0000F109
#define ENTYPO_ICON_ALIGN_BOTTOM                0x0000F10A
#define ENTYPO_ICON_ALIGN_HORIZONTAL_MIDDLE     0x0000F10B
#define ENTYPO_ICON_ALIGN_LEFT                  0x0000F10C
#define ENTYPO_ICON_ALIGN_RIGHT                 0x0000F10D
#define ENTYPO_ICON_ALIGN_TOP                   0x0000F10E
#define ENTYPO_ICON_ALIGN_VERTICAL_MIDDLE       0x0000F10F
#define ENTYPO_ICON_APP_STORE                   0x0000F110
#define ENTYPO_ICON_ARCHIVE                     0x0000F111
#define ENTYPO_ICON_AREA_GRAPH                  0x0000F112
#define ENTYPO_ICON_ARROW_BOLD_DOWN             0x0000F113
#define ENTYPO_ICON_ARROW_BOLD_LEFT             0x0000F114
#define ENTYPO_ICON_ARROW_BOLD_RIGHT            0x0000F115
#define ENTYPO_ICON_ARROW_BOLD_UP               0x0000F116
#define ENTYPO_ICON_ARROW_DOWN                  0x0000F117
#define ENTYPO_ICON_ARROW_LEFT                  0x0000F118
#define ENTYPO_ICON_ARROW_LONG_DOWN             0x0000F119
#define ENTYPO_ICON_ARROW_LONG_LEFT             0x0000F11A
#define ENTYPO_ICON_ARROW_LONG_RIGHT            0x0000F11B
#define ENTYPO_ICON_ARROW_LONG_UP               0x0000F11C
#define ENTYPO_ICON_ARROW_RIGHT                 0x0000F11D
#define ENTYPO_ICON_ARROW_UP                    0x0000F11E
#define ENTYPO_ICON_ARROW_WITH_CIRCLE_DOWN      0x0000F11F
#define ENTYPO_ICON_ARROW_WITH_CIRCLE_LEFT      0x0000F120
#define ENTYPO_ICON_ARROW_WITH_CIRCLE_RIGHT     0x0000F121
#define ENTYPO_ICON_ARROW_WITH_CIRCLE_UP        0x0000F122
#define ENTYPO_ICON_ATTACHMENT                  0x0000F123
#define ENTYPO_ICON_AWARENESS_RIBBON            0x0000F124
#define ENTYPO_ICON_BACK                        0x0000F125
#define ENTYPO_ICON_BACK_IN_TIME                0x0000F126
#define ENTYPO_ICON_BAIDU                       0x0000F127
#define ENTYPO_ICON_BAR_GRAPH                   0x0000F128
#define ENTYPO_ICON_BASECAMP                    0x0000F129
#define ENTYPO_ICON_BATTERY                     0x0000F12A
#define ENTYPO_ICON_BEAMED_NOTE                 0x0000F12B
#define ENTYPO_ICON_BEHANCE                     0x0000F12C
#define ENTYPO_ICON_BELL                        0x0000F12D
#define ENTYPO_ICON_BLACKBOARD                  0x0000F12E
```

```
#define ENTYPO_ICON_BLOCK                           0x0000F12F
#define ENTYPO_ICON_BOOK                            0x0000F130
#define ENTYPO_ICON_BOOKMARK                        0x0000F131
#define ENTYPO_ICON_BOOKMARKS                       0x0000F132
#define ENTYPO_ICON_BOWL                            0x0000F133
#define ENTYPO_ICON_BOX                             0x0000F134
#define ENTYPO_ICON_BRIEFCASE                       0x0000F135
#define ENTYPO_ICON_BROWSER                         0x0000F136
#define ENTYPO_ICON_BRUSH                           0x0000F137
#define ENTYPO_ICON_BUCKET                          0x0000F138
#define ENTYPO_ICON_BUG                             0x0000F139
#define ENTYPO_ICON_CAKE                            0x0000F13A
#define ENTYPO_ICON_CALCULATOR                      0x0000F13B
#define ENTYPO_ICON_CALENDAR                        0x0000F13C
#define ENTYPO_ICON_CAMERA                          0x0000F13D
#define ENTYPO_ICON_CCW                             0x0000F13E
#define ENTYPO_ICON_CHAT                            0x0000F13F
#define ENTYPO_ICON_CHECK                           0x0000F140
#define ENTYPO_ICON_CHEVRON_DOWN                    0x0000F141
#define ENTYPO_ICON_CHEVRON_LEFT                    0x0000F142
#define ENTYPO_ICON_CHEVRON_RIGHT                   0x0000F143
#define ENTYPO_ICON_CHEVRON_SMALL_DOWN              0x0000F144
#define ENTYPO_ICON_CHEVRON_SMALL_LEFT              0x0000F145
#define ENTYPO_ICON_CHEVRON_SMALL_RIGHT             0x0000F146
#define ENTYPO_ICON_CHEVRON_SMALL_UP                0x0000F147
#define ENTYPO_ICON_CHEVRON_THIN_DOWN               0x0000F148
#define ENTYPO_ICON_CHEVRON_THIN_LEFT               0x0000F149
#define ENTYPO_ICON_CHEVRON_THIN_RIGHT              0x0000F14A
#define ENTYPO_ICON_CHEVRON_THIN_UP                 0x0000F14B
#define ENTYPO_ICON_CHEVRON_UP                      0x0000F14C
#define ENTYPO_ICON_CHEVRON_WITH_CIRCLE_DOWN        0x0000F14D
#define ENTYPO_ICON_CHEVRON_WITH_CIRCLE_LEFT        0x0000F14E
#define ENTYPO_ICON_CHEVRON_WITH_CIRCLE_RIGHT       0x0000F14F
#define ENTYPO_ICON_CHEVRON_WITH_CIRCLE_UP          0x0000F150
#define ENTYPO_ICON_CIRCLE                          0x0000F151
#define ENTYPO_ICON_CIRCLE_WITH_CROSS               0x0000F152
#define ENTYPO_ICON_CIRCLE_WITH_MINUS               0x0000F153
#define ENTYPO_ICON_CIRCLE_WITH_PLUS                0x0000F154
#define ENTYPO_ICON_CIRCULAR_GRAPH                  0x0000F155
#define ENTYPO_ICON_CLAPPERBOARD                    0x0000F156
#define ENTYPO_ICON_CLASSIC_COMPUTER                0x0000F157
#define ENTYPO_ICON_CLIPBOARD                       0x0000F158
#define ENTYPO_ICON_CLOCK                           0x0000F159
#define ENTYPO_ICON_CLOUD                           0x0000F15A
#define ENTYPO_ICON_CODE                            0x0000F15B
#define ENTYPO_ICON_COG                             0x0000F15C
#define ENTYPO_ICON_COLOURS                         0x0000F15D
#define ENTYPO_ICON_COMPASS                         0x0000F15E
#define ENTYPO_ICON_CONTROLLER_FAST_BACKWARD        0x0000F15F
#define ENTYPO_ICON_CONTROLLER_FAST_FORWARD         0x0000F160
#define ENTYPO_ICON_CONTROLLER_JUMP_TO_START        0x0000F161
#define ENTYPO_ICON_CONTROLLER_NEXT                 0x0000F162
#define ENTYPO_ICON_CONTROLLER_PAUS                 0x0000F163
#define ENTYPO_ICON_CONTROLLER_PLAY                 0x0000F164
#define ENTYPO_ICON_CONTROLLER_RECORD               0x0000F165
#define ENTYPO_ICON_CONTROLLER_STOP                 0x0000F166
#define ENTYPO_ICON_CONTROLLER_VOLUME               0x0000F167
#define ENTYPO_ICON_COPY                            0x0000F168
```

```
#define ENTYPO_ICON_CREATIVE_CLOUD                      0x0000F169
#define ENTYPO_ICON_CREATIVE_COMMONS                    0x0000F16A
#define ENTYPO_ICON_CREATIVE_COMMONS_ATTRIBUTION        0x0000F16B
#define ENTYPO_ICON_CREATIVE_COMMONS_NODERIVS           0x0000F16C
#define ENTYPO_ICON_CREATIVE_COMMONS_NONCOMMERCIAL_EU   0x0000F16D
#define ENTYPO_ICON_CREATIVE_COMMONS_NONCOMMERCIAL_US   0x0000F16E
#define ENTYPO_ICON_CREATIVE_COMMONS_PUBLIC_DOMAIN      0x0000F16F
#define ENTYPO_ICON_CREATIVE_COMMONS_REMIX              0x0000F170
#define ENTYPO_ICON_CREATIVE_COMMONS_SHARE              0x0000F171
#define ENTYPO_ICON_CREATIVE_COMMONS_SHAREALIKE         0x0000F172
#define ENTYPO_ICON_CREDIT                              0x0000F173
#define ENTYPO_ICON_CREDIT_CARD                         0x0000F174
#define ENTYPO_ICON_CROP                                0x0000F175
#define ENTYPO_ICON_CROSS                               0x0000F176
#define ENTYPO_ICON_CUP                                 0x0000F177
#define ENTYPO_ICON_CW                                  0x0000F178
#define ENTYPO_ICON_CYCLE                               0x0000F179
#define ENTYPO_ICON_DATABASE                            0x0000F17A
#define ENTYPO_ICON_DIAL_PAD                            0x0000F17B
#define ENTYPO_ICON_DIRECTION                           0x0000F17C
#define ENTYPO_ICON_DOCUMENT                            0x0000F17D
#define ENTYPO_ICON_DOCUMENT_LANDSCAPE                  0x0000F17E
#define ENTYPO_ICON_DOCUMENTS                           0x0000F17F
#define ENTYPO_ICON_DOT_SINGLE                          0x0000F180
#define ENTYPO_ICON_DOTS_THREE_HORIZONTAL               0x0000F181
#define ENTYPO_ICON_DOTS_THREE_VERTICAL                 0x0000F182
#define ENTYPO_ICON_DOTS_TWO_HORIZONTAL                 0x0000F183
#define ENTYPO_ICON_DOTS_TWO_VERTICAL                   0x0000F184
#define ENTYPO_ICON_DOWNLOAD                            0x0000F185
#define ENTYPO_ICON_DRIBBBLE                            0x0000F186
#define ENTYPO_ICON_DRIBBBLE_WITH_CIRCLE                0x0000F187
#define ENTYPO_ICON_DRINK                               0x0000F188
#define ENTYPO_ICON_DRIVE                               0x0000F189
#define ENTYPO_ICON_DROP                                0x0000F18A
#define ENTYPO_ICON_DROPBOX                             0x0000F18B
#define ENTYPO_ICON_EDIT                                0x0000F18C
#define ENTYPO_ICON_EMAIL                               0x0000F18D
#define ENTYPO_ICON_EMOJI_FLIRT                         0x0000F18E
#define ENTYPO_ICON_EMOJI_HAPPY                         0x0000F18F
#define ENTYPO_ICON_EMOJI_NEUTRAL                       0x0000F190
#define ENTYPO_ICON_EMOJI_SAD                           0x0000F191
#define ENTYPO_ICON_ERASE                               0x0000F192
#define ENTYPO_ICON_ERASER                              0x0000F193
#define ENTYPO_ICON_EVERNOTE                            0x0000F194
#define ENTYPO_ICON_EXPORT                              0x0000F195
#define ENTYPO_ICON_EYE                                 0x0000F196
#define ENTYPO_ICON_EYE_WITH_LINE                       0x0000F197
#define ENTYPO_ICON_FACEBOOK                            0x0000F198
#define ENTYPO_ICON_FACEBOOK_WITH_CIRCLE                0x0000F199
#define ENTYPO_ICON_FEATHER                             0x0000F19A
#define ENTYPO_ICON_FINGERPRINT                         0x0000F19B
#define ENTYPO_ICON_FLAG                                0x0000F19C
#define ENTYPO_ICON_FLASH                               0x0000F19D
#define ENTYPO_ICON_FLASHLIGHT                          0x0000F19E
#define ENTYPO_ICON_FLAT_BRUSH                          0x0000F19F
#define ENTYPO_ICON_FLATTR                              0x0000F1A0
#define ENTYPO_ICON_FLICKR                              0x0000F1A1
#define ENTYPO_ICON_FLICKR_WITH_CIRCLE                  0x0000F1A2
```

```
#define ENTYPO_ICON_FLOW_BRANCH                 0x0000F1A3
#define ENTYPO_ICON_FLOW_CASCADE                0x0000F1A4
#define ENTYPO_ICON_FLOW_LINE                   0x0000F1A5
#define ENTYPO_ICON_FLOW_PARALLEL               0x0000F1A6
#define ENTYPO_ICON_FLOW_TREE                   0x0000F1A7
#define ENTYPO_ICON_FLOWER                      0x0000F1A8
#define ENTYPO_ICON_FOLDER                      0x0000F1A9
#define ENTYPO_ICON_FOLDER_IMAGES               0x0000F1AA
#define ENTYPO_ICON_FOLDER_MUSIC                0x0000F1AB
#define ENTYPO_ICON_FOLDER_VIDEO                0x0000F1AC
#define ENTYPO_ICON_FORWARD                     0x0000F1AD
#define ENTYPO_ICON_FOURSQUARE                  0x0000F1AE
#define ENTYPO_ICON_FUNNEL                      0x0000F1AF
#define ENTYPO_ICON_GAME_CONTROLLER             0x0000F1B0
#define ENTYPO_ICON_GAUGE                       0x0000F1B1
#define ENTYPO_ICON_GITHUB                      0x0000F1B2
#define ENTYPO_ICON_GITHUB_WITH_CIRCLE          0x0000F1B3
#define ENTYPO_ICON_GLOBE                       0x0000F1B4
#define ENTYPO_ICON_GOOGLE_DRIVE                0x0000F1B5
#define ENTYPO_ICON_GOOGLE_HANGOUTS             0x0000F1B6
#define ENTYPO_ICON_GOOGLE_PLAY                 0x0000F1B7
#define ENTYPO_ICON_GOOGLE_PLUS                 0x0000F1B8
#define ENTYPO_ICON_GOOGLE_PLUS_WITH_CIRCLE     0x0000F1B9
#define ENTYPO_ICON_GRADUATION_CAP              0x0000F1BA
#define ENTYPO_ICON_GRID                        0x0000F1BB
#define ENTYPO_ICON_GROOVESHARK                 0x0000F1BC
#define ENTYPO_ICON_HAIR_CROSS                  0x0000F1BD
#define ENTYPO_ICON_HAND                        0x0000F1BE
#define ENTYPO_ICON_HEART                       0x0000F1BF
#define ENTYPO_ICON_HEART_OUTLINED              0x0000F1C0
#define ENTYPO_ICON_HELP                        0x0000F1C1
#define ENTYPO_ICON_HELP_WITH_CIRCLE            0x0000F1C2
#define ENTYPO_ICON_HOME                        0x0000F1C3
#define ENTYPO_ICON_HOUR_GLASS                  0x0000F1C4
#define ENTYPO_ICON_HOUZZ                       0x0000F1C5
#define ENTYPO_ICON_ICLOUD                      0x0000F1C6
#define ENTYPO_ICON_IMAGE                       0x0000F1C7
#define ENTYPO_ICON_IMAGE_INVERTED              0x0000F1C8
#define ENTYPO_ICON_IMAGES                      0x0000F1C9
#define ENTYPO_ICON_INBOX                       0x0000F1CA
#define ENTYPO_ICON_INFINITY                    0x0000F1CB
#define ENTYPO_ICON_INFO                        0x0000F1CC
#define ENTYPO_ICON_INFO_WITH_CIRCLE            0x0000F1CD
#define ENTYPO_ICON_INSTAGRAM                   0x0000F1CE
#define ENTYPO_ICON_INSTAGRAM_WITH_CIRCLE       0x0000F1CF
#define ENTYPO_ICON_INSTALL                     0x0000F1D0
#define ENTYPO_ICON_KEY                         0x0000F1D1
#define ENTYPO_ICON_KEYBOARD                    0x0000F1D2
#define ENTYPO_ICON_LAB_FLASK                   0x0000F1D3
#define ENTYPO_ICON_LANDLINE                    0x0000F1D4
#define ENTYPO_ICON_LANGUAGE                    0x0000F1D5
#define ENTYPO_ICON_LAPTOP                      0x0000F1D6
#define ENTYPO_ICON_LASTFM                      0x0000F1D7
#define ENTYPO_ICON_LASTFM_WITH_CIRCLE          0x0000F1D8
#define ENTYPO_ICON_LAYERS                      0x0000F1D9
#define ENTYPO_ICON_LEAF                        0x0000F1DA
#define ENTYPO_ICON_LEVEL_DOWN                  0x0000F1DB
#define ENTYPO_ICON_LEVEL_UP                    0x0000F1DC
```

```
#define ENTYPO_ICON_LIFEBUOY                    0x0000F1DD
#define ENTYPO_ICON_LIGHT_BULB                  0x0000F1DE
#define ENTYPO_ICON_LIGHT_DOWN                  0x0000F1DF
#define ENTYPO_ICON_LIGHT_UP                    0x0000F1E0
#define ENTYPO_ICON_LINE_GRAPH                  0x0000F1E1
#define ENTYPO_ICON_LINK                        0x0000F1E2
#define ENTYPO_ICON_LINKEDIN                    0x0000F1E3
#define ENTYPO_ICON_LINKEDIN_WITH_CIRCLE        0x0000F1E4
#define ENTYPO_ICON_LIST                        0x0000F1E5
#define ENTYPO_ICON_LOCATION                    0x0000F1E6
#define ENTYPO_ICON_LOCATION_PIN                0x0000F1E7
#define ENTYPO_ICON_LOCK                        0x0000F1E8
#define ENTYPO_ICON_LOCK_OPEN                   0x0000F1E9
#define ENTYPO_ICON_LOG_OUT                     0x0000F1EA
#define ENTYPO_ICON_LOGIN                       0x0000F1EB
#define ENTYPO_ICON_LOOP                        0x0000F1EC
#define ENTYPO_ICON_MAGNET                      0x0000F1ED
#define ENTYPO_ICON_MAGNIFYING_GLASS            0x0000F1EE
#define ENTYPO_ICON_MAIL                        0x0000F1EF
#define ENTYPO_ICON_MAIL_WITH_CIRCLE            0x0000F1F0
#define ENTYPO_ICON_MAN                         0x0000F1F1
#define ENTYPO_ICON_MAP                         0x0000F1F2
#define ENTYPO_ICON_MASK                        0x0000F1F3
#define ENTYPO_ICON_MEDAL                       0x0000F1F4
#define ENTYPO_ICON_MEDIUM                      0x0000F1F5
#define ENTYPO_ICON_MEDIUM_WITH_CIRCLE          0x0000F1F6
#define ENTYPO_ICON_MEGAPHONE                   0x0000F1F7
#define ENTYPO_ICON_MENU                        0x0000F1F8
#define ENTYPO_ICON_MERGE                       0x0000F1F9
#define ENTYPO_ICON_MESSAGE                     0x0000F1FA
#define ENTYPO_ICON_MIC                         0x0000F1FB
#define ENTYPO_ICON_MINUS                       0x0000F1FC
#define ENTYPO_ICON_MIXI                        0x0000F1FD
#define ENTYPO_ICON_MOBILE                      0x0000F1FE
#define ENTYPO_ICON_MODERN_MIC                  0x0000F1FF
#define ENTYPO_ICON_MOON                        0x0000F200
#define ENTYPO_ICON_MOUSE                       0x0000F201
#define ENTYPO_ICON_MOUSE_POINTER               0x0000F202
#define ENTYPO_ICON_MUSIC                       0x0000F203
#define ENTYPO_ICON_NETWORK                     0x0000F204
#define ENTYPO_ICON_NEW                         0x0000F205
#define ENTYPO_ICON_NEW_MESSAGE                 0x0000F206
#define ENTYPO_ICON_NEWS                        0x0000F207
#define ENTYPO_ICON_NEWSLETTER                  0x0000F208
#define ENTYPO_ICON_NOTE                        0x0000F209
#define ENTYPO_ICON_NOTIFICATION                0x0000F20A
#define ENTYPO_ICON_NOTIFICATIONS_OFF           0x0000F20B
#define ENTYPO_ICON_OLD_MOBILE                  0x0000F20C
#define ENTYPO_ICON_OLD_PHONE                   0x0000F20D
#define ENTYPO_ICON_ONEDRIVE                    0x0000F20E
#define ENTYPO_ICON_OPEN_BOOK                   0x0000F20F
#define ENTYPO_ICON_PALETTE                     0x0000F210
#define ENTYPO_ICON_PAPER_PLANE                 0x0000F211
#define ENTYPO_ICON_PAYPAL                      0x0000F212
#define ENTYPO_ICON_PENCIL                      0x0000F213
#define ENTYPO_ICON_PHONE                       0x0000F214
#define ENTYPO_ICON_PICASA                      0x0000F215
#define ENTYPO_ICON_PIE_CHART                   0x0000F216
```

```
#define ENTYPO_ICON_PIN                         0x0000F217
#define ENTYPO_ICON_PINTEREST                   0x0000F218
#define ENTYPO_ICON_PINTEREST_WITH_CIRCLE       0x0000F219
#define ENTYPO_ICON_PLUS                        0x0000F21A
#define ENTYPO_ICON_POPUP                       0x0000F21B
#define ENTYPO_ICON_POWER_PLUG                  0x0000F21C
#define ENTYPO_ICON_PRICE_RIBBON                0x0000F21D
#define ENTYPO_ICON_PRICE_TAG                   0x0000F21E
#define ENTYPO_ICON_PRINT                       0x0000F21F
#define ENTYPO_ICON_PROGRESS_EMPTY              0x0000F220
#define ENTYPO_ICON_PROGRESS_FULL               0x0000F221
#define ENTYPO_ICON_PROGRESS_ONE                0x0000F222
#define ENTYPO_ICON_PROGRESS_TWO                0x0000F223
#define ENTYPO_ICON_PUBLISH                     0x0000F224
#define ENTYPO_ICON_QQ                          0x0000F225
#define ENTYPO_ICON_QQ_WITH_CIRCLE              0x0000F226
#define ENTYPO_ICON_QUOTE                       0x0000F227
#define ENTYPO_ICON_RADIO                       0x0000F228
#define ENTYPO_ICON_RAFT                        0x0000F229
#define ENTYPO_ICON_RAFT_WITH_CIRCLE            0x0000F22A
#define ENTYPO_ICON_RAINBOW                     0x0000F22B
#define ENTYPO_ICON_RDIO                        0x0000F22C
#define ENTYPO_ICON_RDIO_WITH_CIRCLE            0x0000F22D
#define ENTYPO_ICON_REMOVE_USER                 0x0000F22E
#define ENTYPO_ICON_RENREN                      0x0000F22F
#define ENTYPO_ICON_REPLY                       0x0000F230
#define ENTYPO_ICON_REPLY_ALL                   0x0000F231
#define ENTYPO_ICON_RESIZE_100_PERCENT          0x0000F232
#define ENTYPO_ICON_RESIZE_FULL_SCREEN          0x0000F233
#define ENTYPO_ICON_RETWEET                     0x0000F234
#define ENTYPO_ICON_ROCKET                      0x0000F235
#define ENTYPO_ICON_ROUND_BRUSH                 0x0000F236
#define ENTYPO_ICON_RSS                         0x0000F237
#define ENTYPO_ICON_RULER                       0x0000F238
#define ENTYPO_ICON_SAVE                        0x0000F239
#define ENTYPO_ICON_SCISSORS                    0x0000F23A
#define ENTYPO_ICON_SCRIBD                      0x0000F23B
#define ENTYPO_ICON_SELECT_ARROWS               0x0000F23C
#define ENTYPO_ICON_SHARE                       0x0000F23D
#define ENTYPO_ICON_SHARE_ALTERNATIVE           0x0000F23E
#define ENTYPO_ICON_SHAREABLE                   0x0000F23F
#define ENTYPO_ICON_SHIELD                      0x0000F240
#define ENTYPO_ICON_SHOP                        0x0000F241
#define ENTYPO_ICON_SHOPPING_BAG                0x0000F242
#define ENTYPO_ICON_SHOPPING_BASKET             0x0000F243
#define ENTYPO_ICON_SHOPPING_CART               0x0000F244
#define ENTYPO_ICON_SHUFFLE                     0x0000F245
#define ENTYPO_ICON_SIGNAL                      0x0000F246
#define ENTYPO_ICON_SINA_WEIBO                  0x0000F247
#define ENTYPO_ICON_SKYPE                       0x0000F248
#define ENTYPO_ICON_SKYPE_WITH_CIRCLE           0x0000F249
#define ENTYPO_ICON_SLIDESHARE                  0x0000F24A
#define ENTYPO_ICON_SMASHING                    0x0000F24B
#define ENTYPO_ICON_SOUND                       0x0000F24C
#define ENTYPO_ICON_SOUND_MIX                   0x0000F24D
#define ENTYPO_ICON_SOUND_MUTE                  0x0000F24E
#define ENTYPO_ICON_SOUNDCLOUD                  0x0000F24F
#define ENTYPO_ICON_SPORTS_CLUB                 0x0000F250
```

```
#define ENTYPO_ICON_SPOTIFY                         0x0000F251
#define ENTYPO_ICON_SPOTIFY_WITH_CIRCLE             0x0000F252
#define ENTYPO_ICON_SPREADSHEET                     0x0000F253
#define ENTYPO_ICON_SQUARED_CROSS                   0x0000F254
#define ENTYPO_ICON_SQUARED_MINUS                   0x0000F255
#define ENTYPO_ICON_SQUARED_PLUS                    0x0000F256
#define ENTYPO_ICON_STAR                            0x0000F257
#define ENTYPO_ICON_STAR_OUTLINED                   0x0000F258
#define ENTYPO_ICON_STOPWATCH                       0x0000F259
#define ENTYPO_ICON_STUMBLEUPON                     0x0000F25A
#define ENTYPO_ICON_STUMBLEUPON_WITH_CIRCLE         0x0000F25B
#define ENTYPO_ICON_SUITCASE                        0x0000F25C
#define ENTYPO_ICON_SWAP                            0x0000F25D
#define ENTYPO_ICON_SWARM                           0x0000F25E
#define ENTYPO_ICON_SWEDEN                          0x0000F25F
#define ENTYPO_ICON_SWITCH                          0x0000F260
#define ENTYPO_ICON_TABLET                          0x0000F261
#define ENTYPO_ICON_TABLET_MOBILE_COMBO             0x0000F262
#define ENTYPO_ICON_TAG                             0x0000F263
#define ENTYPO_ICON_TEXT                            0x0000F264
#define ENTYPO_ICON_TEXT_DOCUMENT                   0x0000F265
#define ENTYPO_ICON_TEXT_DOCUMENT_INVERTED          0x0000F266
#define ENTYPO_ICON_THERMOMETER                     0x0000F267
#define ENTYPO_ICON_THUMBS_DOWN                     0x0000F268
#define ENTYPO_ICON_THUMBS_UP                       0x0000F269
#define ENTYPO_ICON_THUNDER_CLOUD                   0x0000F26A
#define ENTYPO_ICON_TICKET                          0x0000F26B
#define ENTYPO_ICON_TIME_SLOT                       0x0000F26C
#define ENTYPO_ICON_TOOLS                           0x0000F26D
#define ENTYPO_ICON_TRAFFIC_CONE                    0x0000F26E
#define ENTYPO_ICON_TRASH                           0x0000F26F
#define ENTYPO_ICON_TREE                            0x0000F270
#define ENTYPO_ICON_TRIANGLE_DOWN                   0x0000F271
#define ENTYPO_ICON_TRIANGLE_LEFT                   0x0000F272
#define ENTYPO_ICON_TRIANGLE_RIGHT                  0x0000F273
#define ENTYPO_ICON_TRIANGLE_UP                     0x0000F274
#define ENTYPO_ICON_TRIPADVISOR                     0x0000F275
#define ENTYPO_ICON_TROPHY                          0x0000F276
#define ENTYPO_ICON_TUMBLR                          0x0000F277
#define ENTYPO_ICON_TUMBLR_WITH_CIRCLE              0x0000F278
#define ENTYPO_ICON_TV                              0x0000F279
#define ENTYPO_ICON_TWITTER                         0x0000F27A
#define ENTYPO_ICON_TWITTER_WITH_CIRCLE             0x0000F27B
#define ENTYPO_ICON_TYPING                          0x0000F27C
#define ENTYPO_ICON_UNINSTALL                       0x0000F27D
#define ENTYPO_ICON_UNREAD                          0x0000F27E
#define ENTYPO_ICON_UNTAG                           0x0000F27F
#define ENTYPO_ICON_UPLOAD                          0x0000F280
#define ENTYPO_ICON_UPLOAD_TO_CLOUD                 0x0000F281
#define ENTYPO_ICON_USER                            0x0000F282
#define ENTYPO_ICON_USERS                           0x0000F283
#define ENTYPO_ICON_V_CARD                          0x0000F284
#define ENTYPO_ICON_VIDEO                           0x0000F285
#define ENTYPO_ICON_VIDEO_CAMERA                    0x0000F286
#define ENTYPO_ICON_VIMEO                           0x0000F287
#define ENTYPO_ICON_VIMEO_WITH_CIRCLE               0x0000F288
#define ENTYPO_ICON_VINE                            0x0000F289
#define ENTYPO_ICON_VINE_WITH_CIRCLE                0x0000F28A
```

```
#define ENTYPO_ICON_VINYL                                0x0000F28B
#define ENTYPO_ICON_VK                                   0x0000F28C
#define ENTYPO_ICON_VK_ALTERNITIVE                       0x0000F28D
#define ENTYPO_ICON_VK_WITH_CIRCLE                       0x0000F28E
#define ENTYPO_ICON_VOICEMAIL                            0x0000F28F
#define ENTYPO_ICON_WALLET                               0x0000F290
#define ENTYPO_ICON_WARNING                              0x0000F291
#define ENTYPO_ICON_WATER                                0x0000F292
#define ENTYPO_ICON_WINDOWS_STORE                        0x0000F293
#define ENTYPO_ICON_XING                                 0x0000F294
#define ENTYPO_ICON_XING_WITH_CIRCLE                     0x0000F295
#define ENTYPO_ICON_YELP                                 0x0000F296
#define ENTYPO_ICON_YOUKO                                0x0000F297
#define ENTYPO_ICON_YOUKO_WITH_CIRCLE                    0x0000F298
#define ENTYPO_ICON_YOUTUBE                              0x0000F299
#define ENTYPO_ICON_YOUTUBE_WITH_CIRCLE                  0x0000F29A

#endif // DOXYGEN_SHOULD_SKIP_THIS
```

Detailed Description

This file defines the full listing of Entypo icons available in NanoGUI. Please note that if viewing the documentation on the web, your browser may display the icons differently than what they look like in NanoGUI. Run the one of the *Example Icons* executables to see what they all look like in NanoGUI.

Note: Constants you may have used in the past may no longer exist, e.g. the name may have changed slightly. For example, `ENTYPO_ICON_CIRCLED_HELP` is renamed to `ENTYPO_ICON_HELP_WITH_CIRCLE`.

Warning: Some icons have a "small" variant, e.g. `ENTYPO_ICON_CHEVRON_SMALL_LEFT` is smaller than `ENTYPO_ICON_CHEVRON_LEFT`. While these "small" icons can be used, they may not be positioned correctly. If you experience this you can, instead of using the "small" variant, use the "regular" icon and call the *`nanogui::Widget::setIconExtraScale()`* function with a value of **less than** `1.0` to scale the icon size down.

Tip: In C++, `#include <nanogui/entypo.h>` to gain access to the `#define` shown in these docs. In Python, `from nanogui import entypo`. So in the below table, when you see `ENTYPO_ICON_FLOW_TREE` that is for C++, and when you see `ICON_FLOW_TREE`, that is for Python, and you would access it using `entypo.ICON_FLOW_TREE`.

The following icons are available:

Included By

- *File nanogui.h*
- *File popupbutton.h*

File formhelper.h

Helper class to construct forms for editing a set of variables of various types.

Page Contents

Definition (nanogui/formhelper.h)

Program Listing for File formhelper.h

- Return to documentation for *File formhelper.h*

```
/*
    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; }

    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 <> 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((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
};

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

- *Template Class FormWidget*
- *Template Class FormWidget< bool, std::true_type >*
- *Template Class FormWidget< Color, std::true_type >*
- *Template Class FormWidget< std::string, std::true_type >*
- *Template Class FormWidget< T, typename std::is_enum< T >::type >*
- *Template Class FormWidget< T, typename std::is_floating_point< T >::type >*
- *Template Class FormWidget< T, typename std::is_integral< T >::type >*
- *Class FormHelper*

File glcanvas.h

Canvas widget for rendering OpenGL content. This widget was contributed by Jan Winkler.

Page Contents

Definition (nanogui/glcanvas.h)

Program Listing for File glcanvas.h

- Return to documentation for *File glcanvas.h*

```
/*
    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

Page Contents

- *Included By*
- *Namespaces*
- *Classes*
- *Functions*
- *Defines*

Definition (`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>

#ifndef DOXYGEN_SHOULD_SKIP_THIS
namespace half_float { class half; }
#endif

#if !defined(GL_HALF_FLOAT) || defined(DOXYGEN_DOCUMENTATION_BUILD)
    #define GL_HALF_FLOAT 0x140B
#endif

NAMESPACE_BEGIN(nanogui)

// bypass template specializations
#ifndef 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
#ifndef 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>
    void setUniform(const std::string &name, const Eigen::Transform<T, 3, 3> &affine,␣
↪bool warn = true) {
```

```
        glUniformMatrix4fv(uniform(name, warn), 1, GL_FALSE, affine.template cast
↪<float>().data());
    }

    template <typename T>
    void setUniform(const std::string &name, const Eigen::Matrix<T, 3, 3> &mat, bool␣
↪warn = true) {
        glUniformMatrix3fv(uniform(name, warn), 1, GL_FALSE, mat.template cast<float>
↪().data());
    }

    template <typename T>
    void setUniform(const std::string &name, const Eigen::Transform<T, 2, 2> &affine,␣
↪bool warn = true) {
        glUniformMatrix3fv(uniform(name, warn), 1, GL_FALSE, affine.template cast
↪<float>().data());
    }

    void setUniform(const std::string &name, bool value, bool warn = true) {
        glUniform1i(uniform(name, warn), (int)value);
    }

    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

Page Contents

Definition (`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

Page Contents

Definition (`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 indexForPosition(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

Page Contents

Definition (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

Page Contents

Definition (`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

A collection of useful layout managers. The *Class GridLayout* was contributed by Christian Schueller.

Page Contents

Definition (`nanogui/layout.h`)

Program Listing for File layout.h

- Return to documentation for *File layout.h*

```
/*
    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:
    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;

    std::vector<int> mRows;

    std::vector<float> mColStretch;

    std::vector<float> 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

Page Contents

Definition (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*

File nanogui.h

Page Contents

- *Definition (nanogui/nanogui.h)*

- *Includes*

Definition (`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

Page Contents

- *Classes*

Definition (`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*
- *Template Class ref*

File opengl.h

Page Contents

Definition (`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

    #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_BEGIN(nanogui)

inline Color::operator const NVGcolor &() const {
    return reinterpret_cast<const NVGcolor &>(*this->data());
}

inline bool nvgIsImageIcon(int value) { return value < 1024; }
```

```
inline bool nvgIsFontIcon(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 nvgIsFontIcon*
- *Function nvgIsImageIcon*

File opengl.h

Page Contents

Definition (`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 popup.h

Page Contents

Definition (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*

File popupbutton.h

Page Contents

Definition (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

Page Contents

Definition (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

Page Contents

Definition (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

Page Contents

Definition (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 mProcessEvents;
    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

Page Contents

Definition (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

Page Contents

Definition (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

Page Contents

Definition (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

Page Contents

Definition (`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*

File tabwidget.h

Page Contents

Definition (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

Page Contents

Definition (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 mMouseDragPos;
    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

- *Template Class FloatBox*
- *Template Class IntBox*
- *Class TextBox*

File theme.h

Storage class for basic theme-related properties.

Page Contents

Definition (nanogui/theme.h)

Program Listing for File theme.h

- Return to documentation for *File theme.h*

```
/*
    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;
    float mIconScale;

    /* 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;

    int mCheckBoxIcon;
    int mMessageInformationIcon;
    int mMessageQuestionIcon;
    int mMessageWarningIcon;
    int mMessageAltButtonIcon;
    int mMessagePrimaryButtonIcon;
    int mPopupChevronRightIcon;
    int mPopupChevronLeftIcon;
    int mTabHeaderLeftIcon;
    int mTabHeaderRightIcon;
    int mTextBoxUpIcon;
    int mTextBoxDownIcon;

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*
- *File widget.h*

Namespaces

- *Namespace nanogui*

Classes

- *Class Theme*

File toolbutton.h

Page Contents

Definition (`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

Page Contents

- *Included By*
- *Namespaces*
- *Classes*

Definition (`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

Page Contents

Definition (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 <nanogui/theme.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();

    Screen *screen();

    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; }

    float iconExtraScale() const { return mIconExtraScale; }

    void setIconExtraScale(float scale) { mIconExtraScale = scale; }

    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();

    inline float icon_scale() const { return mTheme->mIconScale * mIconExtraScale; }

protected:
    Widget *mParent;
    ref<Theme> mTheme;
    ref<Layout> mLayout;
    std::string mId;
    Vector2i mPos, mSize, mFixedSize;
    std::vector<Widget *> mChildren;

    bool mVisible;

    bool mEnabled;
    bool mFocused, mMouseFocus;
    std::string mTooltip;
    int mFontSize;

    float mIconExtraScale;
    Cursor mCursor;
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

NAMESPACE_END(nanogui)
```

Includes

- `nanogui/object.h` (*File object.h*)
- `nanogui/theme.h` (*File theme.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

Page Contents

Definition (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	• Most member methods. • All member variables.
checkbox.h	• All member methods and variables.
colorpicker.h	• Constructor and callback. • All member variables.
colorwheel.h	• Most methods and member variables.
combobox.h	• Most member methods and variables.
formhelper.h	• More detailed documentation explaining parameters for `FormHelper` methods. • Most member variables.
graph.h	• All member methods and variables.
imagepanel.h	• All member methods and variables.
imageview.h	• Most member methods.
label.h	• Some member methods and variables.
layout.h	• Nearly everything.
popup.h	• Some member methods and variables. • Explicit parameter documentation would be very useful.
popupbutton.h	• Almost everything.
progressbar.h	• Almost everything.
screen.h	• Documentation for the manual GLFW API. • All member variables.
slider.h	• Almost everything.
stackedwidget.h	• Almost everything.
tabheader.h	• Some member methods. • Some reformatting of existing documentation to use `\param` or `\return` etc.

tabwidget.h	• Some member methods. • Some reformatting of existing documentation to use `\param` or `\return` etc.

Advanced Contribution Opportunity

Currently, all partial and full template specializations are skipped. Specifically, nearly everything in `include/nanogui/serializer/*`. 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

Index

G

N

S