Layers

Generated by Doxygen 1.9.3

Chapter 1

Changelog

All notable changes to this project will be documented in this file.

1.1 Alpha Phase

1.1.1 3.1.0a (Unreleased)

- Implemented the AttributeType class which now serves as the abstract parent class of Attribute and AttributeGroup.
- Added AttributeType::setup_widget_update_connection() which simplifies connecting attribute value change to the QWidget::update() function
- Attribute groups are now recognized by themes
- Attribute groups are now disableable
- All top-level attribute widgets now have a disable toggle switch. The disable toggle switch has been realigned to the top-left corner of attribute widgets.
- Capitalized attribute names are now derived
- Implemented the Data class which either stores a single variant or multiple state-variant pairs
- Implemented the ThemeableBox class which generalizes functionality that was shared between the Widget class and the various dialog classes.
- Implemented the Dialog class which further generalizes the dialog sub-classes
- Implemented a ScrollBar class that is themeable and customizable
- Themes are now stored in the AppData/Local/Layers directory which has been reinstated to aid multiple app support for themes.
- Implemented theme directories and app/theme UUIDs

1.1.2 3.0.0a (November 29, 2022)

- The Attribute class now inherits QObject to provide signal/slot functionality.
- Created a Variant class that wraps a QVariant and inherits QObject to provide signal/slot functionality. > Attributes store Variants which are made to be replaceable. An Attribute can replace its Variant with another Attribute's Variant. If either Attribute makes a change to the Variant, both Attributes get updated. > When Variants update, it emits Variant::changed, and the Attributes linked to them emit the Attribute::value_changed signal. This mechanism is referred to as attribute value change detection, and it replaces the Attribute↩ SharingCombo. The AttributeSharingCombo class has been deprecated and removed.
- Previously, when setting an attribute's value, the value would be set without checking if the attribute already had that value. Now it performs that check, resulting in a performance boost and better protection.
- The AttributeSet class has been removed. > Slightly different data structures are now used for storing attributes between Themes and Themeables. > Widget attributes are now initialized as public member variables, removing the need to iterate each time an Attribute needs to be referenced.
- Created Theme::consume(theme) function for applications to add their widget's theme values to the library's default themes
- Removed issue_update() since widgets can connect update() to Attribute::value_changed.
- Saving and loading now uses JSON formatting.
- Changed Variant->ints to Variant->doubles. This change was made due to JSON formatting not differentiating between int and double types
- The theme building functions have been removed. Changes to the prebuilt themes are made directly in their JSON files.
- Previously, all CustomizePanels were initialized and acquired when the application was launched. This resulted in a large amount of memory being used which was unnecessary because the user only interacts with a single CustomizePanel at a time. Now, CustomizePanels are generated during runtime as the user navigates the widget hierarchy.
- Renamed various AttributeWidget classes
- Created the AttributeGroup and AttributeLayout classes to simplify the creation of AttributeWidgets and specify their organization in CustomizePanels
- Created a FillControl class that simplifies setting color/gradient values and toggling between them.
- Created the StateAW (State Attribute Widget) class which provides an improved interface for customizing stateful attributes.
- The NumberAW (formerly known as NumberAttributeWidget) now utilizes a mini slider and takes up less space.
- More widgets are customizable: > Widgets of the Customize menu's topbar > Dialogs > ScrollAreas > Fixed App Preferences settings panel customizations
- More widget attributes are customizable: > Widget margins > Widgets can now be filled with gradients. > Fixed outline color and corner color customizations
- Collapsed text button widget is now a different color to prevent the text buttons from disappearing
- Reworked application initialization > A GitHubRepo class has been created to clarify that the string argument representing a GitHub repo provided for app initialization is a GitHub repo. > A Version class has been created to clarify that the string argument representing the app version provided for app initialization is a Version.
- Created new widget classes, MenuBar and TabBar.
- Image sequences can now be saved and loaded as a single file

- Fixed some issues where the preview window had too much functionality enabled
- Combobox items are no longer editable by default
- Button text can word wrap now

1.1.3 2.2.0a (September 11, 2021)

- Migrated to Qt 6
- Created a mechanism to update the application during start up
- Application settings are now stored in the registry system.
- The active theme selected by the user will now be applied automatically when the application is launched again.
- Prebuilt themes are now stored within the program and not in the app's theme folder.
- Fixed an issue with the theme file updating functionality that caused themes from old versions to be missing attribute sets and attributes for widgets created in later versions

1.1.4 2.1.0a (September 1, 2021)

- "Default" and "Common" states are gone. Attributes are now created as stateful or stateless. This makes customization of themeables that use both types of attributes much clearer.
- The corner radii attribute widget has been redesigned and takes up less space.
- Layers applications will no longer store theme files in a universal 'Layers'/'Layers-alpha' folder in the user's app data directory. Instead, each individual application will store its own set of theme files in a folder named after the application.
- Created a mechanism to update theme files. Themes created in 2.0.0a will get updated and continue to work in this version.
- Created a mechanism to collapse text buttons in the Customize menu's topbar

1.1.5 2.0.0a (June 23, 2021)

- Layers has been fully rewritten.
- Theme swapping and customization application are now instant (This is expected to remain the case, but there is no guarantee)
- Reworked the Customize menu. Expanded customization for all widgets and improved widget hierarchy navigation
- Rounded corner support. Many widgets now use rounded corners by default
- Most buttons and controls have increased padding around their content and increased spacing between each other to improve touch experience
- Widget attributes are now applied via paint events instead of invoking stylesheets
- Pop-up boxes are now separate windows
- Changed the alpha save directory to "C:/Users/{Your username}/AppData/Local/Layers-alpha"

Chapter 2

Layers is an application template library with a focus on themes and user customization.

2.1 Demo

The Layers Demo is a minimal Layers application that demonstrates the capabilities of the Layers software library. The source-code demonstrates how to get a basic Layers application up and running, while the binary release is provided to let developers see a Layers app in action.

2.2 Development and Testing

These are the steps to install a copy of Layers source-code on your local machine for development and testing purposes.

It is recommended to clone a copy of the project repository to be able to use project features that have been implemented. You may clone the project's repository with:

```
git clone https://github.com/huntermalm/Layers.git
```

You may also download a .zip of the project here.

2.2.0.0.1 Layers development requires Qt 6. Other Qt versions may work but have not been tested.

2.2.0.1 Documentation

Layers documentation is still in progress. Once it is completed, a link will be provided here.

2.2.0.2 Project Goals

A list of project goals is still in progress. Once it is completed, a link will be provided here.

2.3 Installation

Since Layers development is still in the alpha stage, do not expect everything this library has to offer to be stable or functional. Alpha releases are provided for developers to test library functionality. It is not recommended to start developing your application with a Layers alpha build since things can change before a production build is released.

Grab the latest version of Layers from the downloads below. Follow the standard library linking process for your IDE.

2.4 Download

- `Layers-2.2.0a.zip` (2021-09-11)

2.5 Authors

- Hunter Malm - *Project creator/Lead developer* - `huntermalm`

See also the list of `contributors` who participated in this project.

2.6 License

This project is licensed under the MIT License - see the `LICENSE` file for details.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

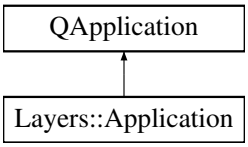
Here is a list of all documented files with brief descriptions:

Chapter 6

Class Documentation

6.1 Layers::Application Class Reference

Inheritance diagram for Layers::Application:



Public Slots

- void rename_theme (const QString &old_name, const QString &new_name)

Signals

- void **current_theme_changed** ()

Public Member Functions

- **Application** (int &argc, char **argv, const QString &name, const QUuid &uuid, QFile *icon_file=nullptr, Version *version=nullptr, GitHubRepo *github_repo=nullptr)
- QString **app_identifier** ()
- void apply_theme (Theme &theme)
- void create_theme (const QString &new_theme_name, const QString &copy_theme_name)
- Theme * current_theme () const
- QFile * icon_file ()
- Theme load_theme (const QString &file_name)
- Window * **main_window** () const
- QString & name ()
- void reapply_theme ()
- void save_theme (Theme &theme)
- QSettings & settings ()
- void store_child_themeable_pointer (Themeable &themeable)
- Theme * theme (const QString &theme_name)
- QMap< QString, Theme > & themes ()
- bool update_available ()
- bool update_on_request ()

6.1.1 Member Function Documentation

6.1.1.1 apply_theme()

```
void Application::apply_theme (
            Theme & theme )
```

Applies a theme across the entire application.

Parameters

theme	to apply

6.1.1.2 create_theme()

```
void Application::create_theme (
            const QString & new_theme_name,
            const QString & copy_theme_name )
```

Creates a new theme.

Parameters

new_theme_name	- Name to give the new theme
copy_theme_name	- Name of the app theme to copy and use as the basis for the new theme

6.1.1.3 current_theme()

```
Theme * Layers::Application::current_theme ( ) const
```

Returns a pointer to the current theme applied to the application.

Returns

pointer to current application theme

6.1.1.4 icon_file()

```
QFile * Application::icon_file ( )
```

Returns a pointer to a QFile of the application icon.

If no icon was supplied during initialization, nullptr is returned.

Returns

pointer to QFile of app icon, nullptr if none exists

6.1.1.5 load_theme()

```
Theme Application::load_theme (
            const QString & file_name )
```

Loads and returns a theme from the supplied file.

This function first attempts to load the theme as a latest version theme. If that fails, it will attempt to load the file under older version conditions until a successful load. Once the particular version is found and loaded, it is updated to the latest version.

This function is updated with each new version of Layers.

Parameters

file	to load theme from

Returns

theme loaded from file

6.1.1.6 name()

```
QString & Application::name ( )
```

Returns the name of the application.

Returns

application name

6.1.1.7 reapply_theme()

```
void Application::reapply_theme ( )
```

Reapplies the theme that is already set.

6.1.1.8 rename_theme

```
void Application::rename_theme (
            const QString & old_name,
            const QString & new_name )  [slot]
```

Renames a theme with the provided new name.

Parameters

old_name	- Name of the theme to rename
new_name	- New name to give to theme

6.1.1.9 save_theme()

```
void Application::save_theme (
            Theme & theme )
```

Saves a theme to a file.

The file is saved to 'C:/Users/{Your username}/AppData/Local/{Application name}/Themes'.

The theme name, lowercased, is used as the filename.

Parameters

theme	to save

6.1.1.10 settings()

```
QSettings & Application::settings ( )
```

Returns the application's settings.

Returns

Settings of the application

6.1.1.11 store_child_themeable_pointer()

```
void Application::store_child_themeable_pointer (
            Themeable & themeable )
```

Stores a pointer to the provided themeable.

The child themeable pointers are used to apply themes to child themeables.

Parameters

themeable	to store a pointer to

6.1.1.12 theme()

```
Theme * Application::theme (
            const QString & theme_name )
```

Returns a pointer to the application theme with the provided name.

Parameters

theme_name	- Name of the theme to be returned

Returns

pointer to theme

6.1.1.13 themes()

```
QMap< QString, Theme > & Application::themes ( )
```

Returns a reference to a QMap containing the application's themes.

The QMap pairs QStrings to Themes, where the QString is the name of the associated theme.

Returns

QMap reference to the app's themes

6.1.1.14 update_available()

```
bool Application::update_available ( )
```

Returns true if an application update is available.

This function compares the current version tag of the application (supplied during initialization) with the latest known version tag found on the application's GitHub repo (also supplied during initialization). If they do not match, true is returned.

Returns

true if update is available, false otherwise

6.1.1.15 update_on_request()

```
bool Application::update_on_request ( )
```

Prompts the user and asks if they'd like to update. Updates application if they choose to.

Returns

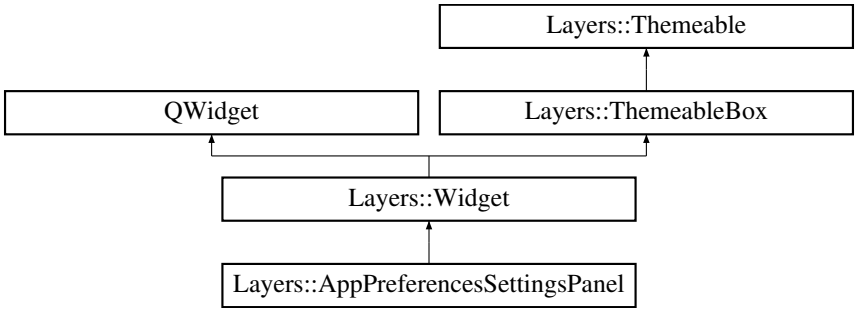
true if user chooses to update, false otherwise

The documentation for this class was generated from the following files:

- include/Application.h
- src/Application/Application.cpp

6.2 Layers::AppPreferencesSettingsPanel Class Reference

Inheritance diagram for Layers::AppPreferencesSettingsPanel:



Public Member Functions

- **AppPreferencesSettingsPanel** (QWidget ∗parent=nullptr)

Additional Inherited Members

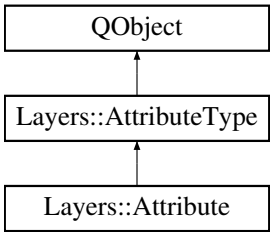
The documentation for this class was generated from the following files:

- include/SettingsPanels.h
- src/Widgets/AppPreferencesSettingsPanel.cpp

6.3 Layers::Attribute Class Reference

`#include <Attribute.h>`

Inheritance diagram for Layers::Attribute:



Signals

- void **ownership_changed** ()

Public Member Functions

- **Attribute** (const QString &name, bool disabled=false)
- **Attribute** (const QString &name, QVariant qvariant, bool disabled=false)
- **Attribute** (const QString &name, VariantMap variant_map, bool disabled=false)
- **Attribute** (const Attribute &a)
- template<typename T >
  T as () const
- template<typename T >
  T as (const QString &state) const
- void clear_data_if_owner ()
- bool contains_state (const QString &state) const
- void copy (const Attribute &attr)
- void establish_data_connection ()
- void entangle_with (Attribute &attribute)
- void init_variant_map (const VariantMap &variant_map)
- virtual bool is_stateful () const override
- bool owns_data () const
- virtual void set_state (const QString &state) override
- void set_value (QVariant qvariant, bool retain_type=true)
- void set_value (const QString &state, QVariant qvariant)
- virtual void setup_widget_update_connection (QWidget *widget) override
- QString state () const
- QList< QString > states () const
- QJsonObject to_json_object ()
- const char * typeName () const

Additional Inherited Members

6.3.1 Detailed Description

Attribute implementation used throughout Layers by AttributeGroup, Theme, and Themeable.

An Attribute stores a pointer to a Data object which is used for drawing Themeables. At first, an Attribute inializes Data and *owns* it. However, through entangle_with(), an Attribute can be made to point to Data from another Attribute.

6.3.2 Member Function Documentation

6.3.2.1 as() [1/2]

```
template<typename T >
T Layers::Attribute::as  [inline]
```

Returns Data value converted to the template type T.

If the Attribute is stateful, the value associated with the current state is returned. If the value associated with a specific state is desired, use the version of as() that requires a state argument.

Returns

Data value converted to template type T

6.3.2.2 as() [2/2]

```
template<typename T >
T Layers::Attribute::as (
            const QString & state ) const  [inline]
```

Returns Data value associated with the supplied state, converted to the template type T.

This function will work only if the Attribute is stateful and only if the state supplied exists in the state-variant map.

Returns

Data value associated with state, converted to template type T

6.3.2.3 clear_data_if_owner()

```
void Layers::Attribute::clear_data_if_owner ( )
```

Deletes the Data if the Attribute is the owner.

6.3.2.4 contains_state()

```
bool Attribute::contains_state (
            const QString & state ) const
```

Returns true if state exists in the Data, otherwise, returns false.

Parameters

state	- State to check whether it exists in the Data

Returns

True if state exists in Data, false otherwise

6.3.2.5 copy()

```
void Attribute::copy (
            const Attribute & attr )
```

Copies the supplied Attribute

Parameters

attr	- Attribute to copy

6.3.2.6 entangle_with()

```
void Attribute::entangle_with (
            Attribute & attribute )
```

Makes this Attribute to point to the Data of another Attribute.

If the caller Attribute owns its Data, the Data is deleted before the pointer is changed.

Establishes a new data connection and emits both ownership_changed() and value_changed().

Another connection is established so that if the attribute supplied changes its ownership, this function gets called again so the caller Attribute can get a pointer to the new Data.

Parameters

attribute	- Attribute to obtain the Data pointer of

6.3.2.7 establish_data_connection()

```
void Attribute::establish_data_connection ( )
```

Connects the Data changed signal to emit value_changed().

The previous connection is disconnected.

6.3.2.8 init_variant_map()

```
void Attribute::init_variant_map (
            const VariantMap & variant_map )
```

Converts to stateful Data initialized with the supplied map.

This function simply calls Data::init_variant_map() and passes the state_variant_map.

Parameters

state_variant_map	- Map to initialize the Data with

6.3.2.9 is_stateful()

```
bool Attribute::is_stateful ( ) const  [override], [virtual]
```

Returns true if stateful, otherwise, returns false.

This function simply calls Data::is_stateful().

Returns

True if stateful, false otherwise

Implements Layers::AttributeType.

6.3.2.10 owns_data()

```
bool Attribute::owns_data ( ) const
```

Returns true if Attribute owns the Data object being pointed to, otherwise, returns false.

Returns

True if Data object is owned by Attribute, false otherwise

6.3.2.11 set_state()

```
void Attribute::set_state (
            const QString & state )  [override], [virtual]
```

Sets the Attribute's active state.

Parameters

state	- QString representing new active state

Implements Layers::AttributeType.

6.3.2.12 set_value() [1/2]

```
void Attribute::set_value (
            const QString & state,
            QVariant qvariant )
```

Set the Data value associated with state.

This function simply calls Data::set_value() and passes state and qvariant. It only works with Attributes that are stateful.

Parameters

state	- State associated with value
qvariant	- QVariant containing the value being set

6.3.2.13 set_value() [2/2]

```
void Attribute::set_value (
            QVariant qvariant,
            bool retain_type = true )
```

Set the value of the Data.

This function simply calls Data::set_value() and passes qvariant and retain_type. It only works with Attributes that are not stateful.

Parameters

qvariant	- QVariant containing the value being set
retain_type	- Whether to protect the value type from change, true by default

6.3.2.14 setup_widget_update_connection()

```
void Attribute::setup_widget_update_connection (
            QWidget * widget )  [override], [virtual]
```

Implements Layers::AttributeType.

6.3.2.15 state()

```
QString Attribute::state ( ) const
```

Returns the active state of the Attribute.

Returns

Active state represented as a QString

6.3.2.16 states()

```
QList< QString > Attribute::states ( ) const
```

Returns a list of QStrings representing the available states.

If the Attribute is not stateful, then an empty list will be returned.

Returns

QString list where QStrings represent the states

6.3.2.17 to_json_object()

```
QJsonObject Attribute::to_json_object ( )
```

Returns Attribute converted to a QJsonObject.

Returns

QJsonObject pertaining to the Attribute

6.3.2.18 typeName()

```
const char * Attribute::typeName ( ) const
```

Returns the name of the value's type.

This function simply calls Data::typeName().

Returns

Name of value's type

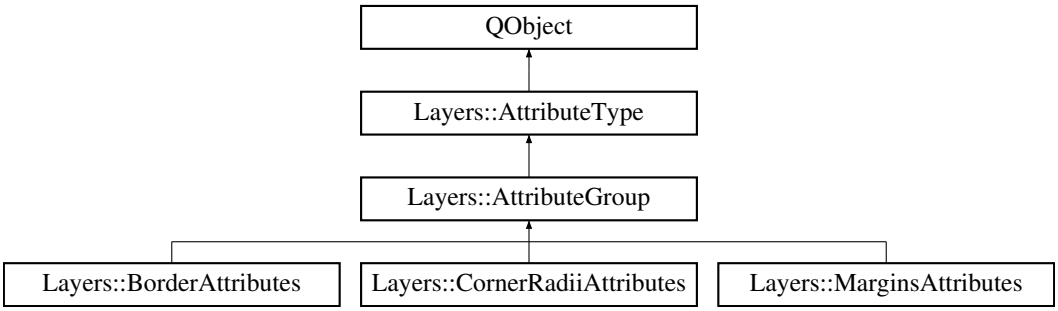
The documentation for this class was generated from the following files:

- include/Attribute.h
- src/Themes/Attributes/Attribute.cpp

6.4 Layers::AttributeGroup Class Reference

```
#include <AttributeGroup.h>
```

Inheritance diagram for Layers::AttributeGroup:



Public Member Functions

- **AttributeGroup** (const QString &name, const QMap< QString, Attribute * > &attributes, bool disabled=false)
- **AttributeGroup** (const AttributeGroup &ag)
- QMap< QString, Attribute * > & attributes ()
- QMap< QString, Attribute * >::iterator **begin** ()
- void copy (const AttributeGroup &ag)
- QMap< QString, Attribute * >::iterator **end** ()
- void entangle_with (AttributeGroup &attr_group)
- virtual bool is_stateful () const override
- virtual void set_state (const QString &state) override
- virtual void setup_widget_update_connection (QWidget *widget) override
- QJsonObject to_json_object ()

Additional Inherited Members

6.4.1 Detailed Description

Implements a container used to define a group of Attributes.

The Attributes are stored in a QMap of QString-Attribute∗ pairs where the QString matches the name of the associated Attribute.

A group of Attributes is stateful if a single Attribute in the group is stateful.

6.4.2 Member Function Documentation

6.4.2.1 attributes()

```
QMap< QString, Attribute * > & AttributeGroup::attributes ( )
```

Returns a reference to the map of Attributes contained in the group.

Returns

Reference to map of Attributes

6.4.2.2 copy()

```
void AttributeGroup::copy (
            const AttributeGroup & ag )
```

Copies the supplied AttributeGroup

Parameters

ag	- AttributeGroup to copy

6.4.2.3 entangle_with()

```
void AttributeGroup::entangle_with (
            AttributeGroup & attr_group )
```

Calls Attribute::entangle_with() on all group Attributes, passing the corresponding Attributes of attr_group matched by the Attribute names.

Parameters

attr_group	- Group of Attributes to obtain Data pointers of

6.4.2.4 is_stateful()

```
bool AttributeGroup::is_stateful ( ) const  [override], [virtual]
```

Returns true if stateful, otherwise, returns false.

This function calls Attribute::is_stateful() on all the Attributes in the group. If any of them are stateful, the group is considered stateful as well.

Returns

True if stateful, false otherwise

Implements Layers::AttributeType.

6.4.2.5 set_state()

```
void AttributeGroup::set_state (
            const QString & state )  [override], [virtual]
```

Sets the active state of all the Attributes in the group.

Parameters

state	- QString representing new active state

Implements Layers::AttributeType.

6.4.2.6 setup_widget_update_connection()

```
void AttributeGroup::setup_widget_update_connection (
            QWidget * widget )  [override], [virtual]
```

Implements Layers::AttributeType.

6.4.2.7 to_json_object()

```
QJsonObject AttributeGroup::to_json_object ( )
```

Returns AttributeGroup converted to a QJsonObject.

Returns

QJsonObject pertaining to the AttributeGroup

The documentation for this class was generated from the following files:

- include/AttributeGroup.h
- src/Themes/Attributes/AttributeGroup.cpp

6.5 Layers::AttributeSet Class Reference

Public Member Functions

- void **add_attribute** (Attribute *attribute)
- Attribute * **attribute** (const QString &attribute_name)
- QVariant * **attribute_value** (const QString &attribute_name)
- QMap< QString, Attribute * > & **attributes** ()
- bool **contains** (const QString &attribute_name)
- void **copy_values_from** (AttributeSet &other_attribute_set)
- void **remove_attribute** (const QString &attribute_name)
- bool **replace_with_proxy** (const QString &attribute_name, Attribute *proxy_attribute)
- void **replace_all_with** (AttributeSet &other_attribute_set)
- void **set_state** (const QString &state)
- QList< QString > **states** () const

Friends

- QDataStream & **operator**<< (QDataStream &stream, const AttributeSet &as)
- QDataStream & **operator**>> (QDataStream &stream, AttributeSet &as)

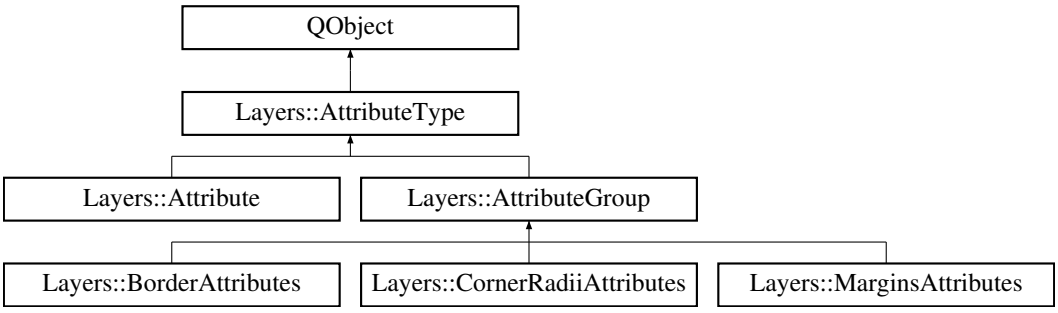
The documentation for this class was generated from the following file:

- include/AttributeSet.h

6.6 Layers::AttributeType Class Reference

```
#include <AttributeType.h>
```

Inheritance diagram for Layers::AttributeType:



Signals

- void **value_changed** ()

Public Member Functions

- **AttributeType** (const QString &name, bool disabled)
- QString capitalized_name ()
- bool disabled () const
- virtual bool is_stateful () const =0
- QString name ()
- virtual void set_disabled (bool disabled=true)
- virtual void set_state (const QString &state)=0
- virtual void **setup_widget_update_connection** (QWidget *widget)=0

Protected Attributes

- bool **m_disabled** { false }
- QString **m_name** { "" }

6.6.1 Detailed Description

AttributeType is an abstract type that should be implemented by attributes or attribute containers.

6.6.2 Member Function Documentation

6.6.2.1 capitalized_name()

```
QString AttributeType::capitalized_name ( )
```

Returns a capitalized version of the name without underscores

Returns

Capitalized name without underscores

6.6.2.2 disabled()

```
bool AttributeType::disabled ( ) const
```

Returns the disabled condition of the AttributeType

Returns

True if disabled, false otherwise

6.6.2.3 is_stateful()

```
virtual bool Layers::AttributeType::is_stateful ( ) const  [pure virtual]
```

Returns true if AttributeType is stateful

An AttributeType is stateful if its Data, or the Data of the attributes in a container, is also stateful. Because there is a difference between the implementations of attributes and their containers, this function is declared here as a pure virtual function. This means that **classes that implement AttributeType will also need to implement this function**.

Returns

True if stateful, false otherwise

Implemented in Layers::Attribute, and Layers::AttributeGroup.

6.6.2.4 name()

```
QString AttributeType::name ( )
```

Returns the name of the AttributeType

Returns

Name of AttributeType

6.6.2.5 set_disabled()

```
void AttributeType::set_disabled (
            bool disabled = true )  [virtual]
```

Sets the AttributeType's disabled condition to the value supplied in the parameter

Parameters

disabled	- Whether to set the AttributeType as disabled, true by default

6.6.2.6 set_state()

```
virtual void Layers::AttributeType::set_state (
            const QString & state )  [pure virtual]
```

Sets the state of the AttributeType

If the AttributeType is a single attribute, then its state should be set. If it is an attribute container, then all of the attributes in the container should have their states set. Because there is a difference between the implementations of attributes and their containers, this function is declared here as a pure virtual function. This means that **classes that implement AttributeType will also need to implement this function**.

Parameters

state	- QString representing state being set

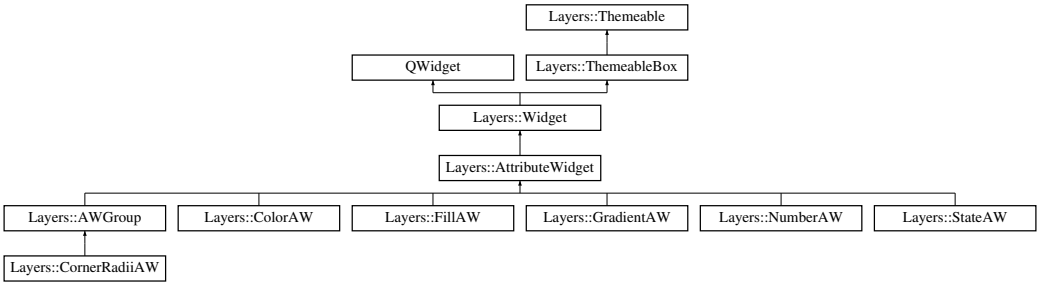
Implemented in Layers::Attribute, and Layers::AttributeGroup.

The documentation for this class was generated from the following files:

- include/AttributeType.h
- src/Themes/Attributes/AttributeType.cpp

6.7 Layers::AttributeWidget Class Reference

Inheritance diagram for Layers::AttributeWidget:



Public Slots

- virtual void **set_current_editting_state** (const QString &state)

Signals

- void **widget_disabled** ()

Public Member Functions

- **AttributeWidget** (AttributeType *attr_type=nullptr, QWidget *parent=nullptr)
- ToggleSwitch * **disable_toggle** () const
- bool **disabled** () const
- Widget * **toggle_label_separator** () const

Protected Member Functions

- void **init_attributes** ()

Protected Attributes

- AttributeType * **m_attribute_type**
- ToggleSwitch * **m_disabled_toggle** { new ToggleSwitch }
- Widget * **m_toggle_label_separator** { new Widget }

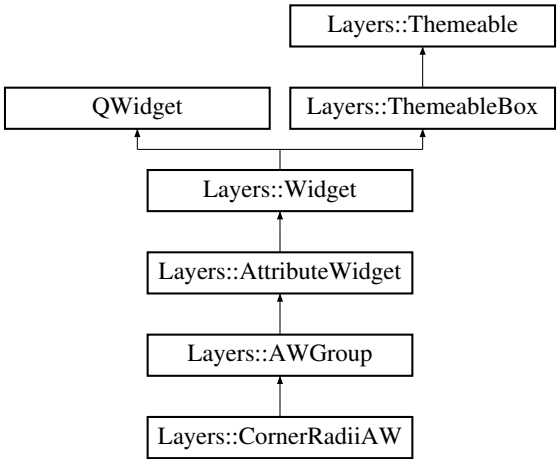
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/AttributeWidget.cpp

6.8 Layers::AWGroup Class Reference

Inheritance diagram for Layers::AWGroup:



Public Slots

- virtual void **set_current_editting_state** (const QString &state) override

Public Member Functions

- **AWGroup** (AttributeGroup *attr_group, QWidget *parent=nullptr)
- void **add_attribute_widget** (AttributeWidget *attribute_widget)
- void **set_collapsed** (bool collapsed=true)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.8.1 Member Function Documentation

6.8.1.1 init_child_themeable_reference_list()

```
void AWGroup::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

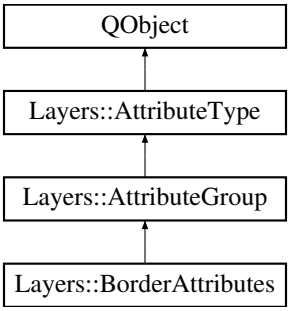
Reimplemented in Layers::CornerRadiiAW.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/AWGroup.cpp

6.9 Layers::BorderAttributes Class Reference

Inheritance diagram for Layers::BorderAttributes:



Public Member Functions

- **BorderAttributes** (const QString &name="border")

Public Attributes

- Attribute fill
- Attribute thickness

Additional Inherited Members

6.9.1 Member Data Documentation

6.9.1.1 fill

```
Attribute Layers::BorderAttributes::fill
```

Initial value:

```
{ Attribute(
            "fill",
            QColor(Qt::gray)
            ) }
```

6.9.1.2 thickness

```
Attribute Layers::BorderAttributes::thickness
```

Initial value:

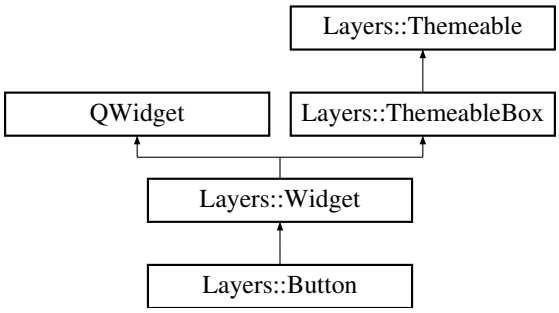
```
{ Attribute(
            "thickness",
            QVariant::fromValue(0.0)
            ) }
```

The documentation for this class was generated from the following files:

- include/AttributeGroup.h
- src/Themes/Attributes/AttributeGroup.cpp

6.10 Layers::Button Class Reference

Inheritance diagram for Layers::Button:



Signals

- void **clicked** ()

Public Member Functions

- **Button** (Graphic ∗graphic, const QString &text, bool auto_touch_target_compliance=false, QWidget ∗parent=nullptr)
- **Button** (Graphic ∗graphic, bool auto_touch_target_compliance=false, QWidget ∗parent=nullptr)
- **Button** (const QString &text, bool auto_touch_target_compliance=false, QWidget ∗parent=nullptr)
- **Button** (Graphic ∗graphic_before, Graphic ∗graphic_after, bool auto_touch_target_compliance=false, QWidget ∗parent=nullptr)
- void **disable_graphic_hover_color** (bool cond=true)
- void **disable_text_hover_color** (bool cond=true)
- bool **disabled** () const
- Graphic ∗ **graphic** () const
- void **resize** ()
- void **set_available_width** (int available_width)
- void **set_disabled** (bool cond=true)
- void **set_font_size** (int size)
- void **set_padding** (int padding)
- void **set_padding** (int left, int top, int right, int bottom)
- void **set_text_padding** (int left, int top, int right, int bottom)
- void **toggle_graphics** ()
- int **left_padding** () const
- int **top_padding** () const
- int **right_padding** () const
- int **bottom_padding** () const

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void **init** ()
- void init_child_themeable_reference_list ()
- void **setup_layout** ()

Additional Inherited Members

6.10.1 Member Function Documentation

6.10.1.1 init_child_themeable_reference_list()

```
void Button::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

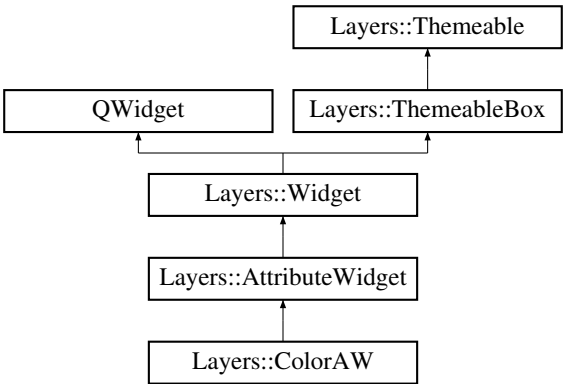
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Button.h
- src/Widgets/Controls/Button.cpp

6.11 Layers::ColorAW Class Reference

Inheritance diagram for Layers::ColorAW:



Public Slots

- void **set_current_editting_state** (const QString &state)

Public Member Functions

- **ColorAW** (Attribute *attribute, QWidget *parent=nullptr)
- ColorControl * **color_control** () const
- void **set_centered** (bool centered=true)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.11.1 Member Function Documentation

6.11.1.1 init_child_themeable_reference_list()

```
void ColorAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

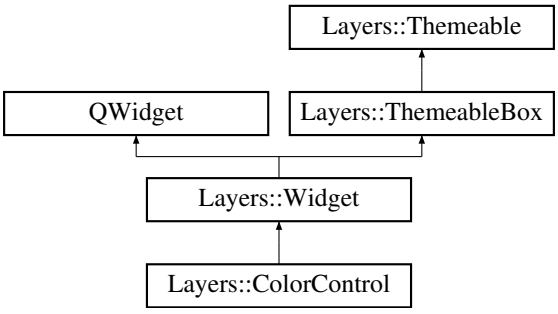
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/ColorAW.cpp

6.12 Layers::ColorControl Class Reference

Inheritance diagram for Layers::ColorControl:



Public Slots

- void **set_current_editting_state** (const QString &state)

Signals

- void **color_changed** ()

Public Member Functions

- **ColorControl** (QWidget ∗parent=nullptr)
- void **click** ()
- void **disable_clicking** (bool cond=true)

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event)
- void **init_attributes** ()

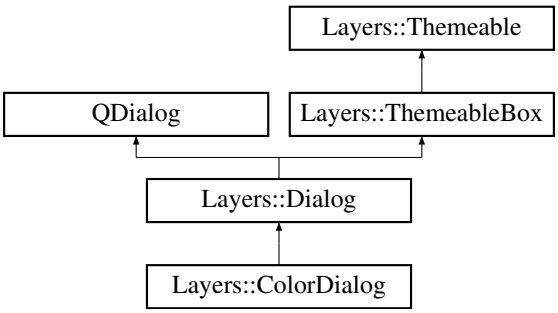
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/ColorControl.h
- src/Widgets/Controls/ColorControl.cpp

6.13 Layers::ColorDialog Class Reference

Inheritance diagram for Layers::ColorDialog:



Public Member Functions

- **ColorDialog** (QWidget ∗parent=nullptr)
- void **update_color_name_line_editor** ()

Public Attributes

- Attribute **color** { Attribute("color", QColor()) }

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.13.1 Member Function Documentation

6.13.1.1 init_child_themeable_reference_list()

```
void ColorDialog::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

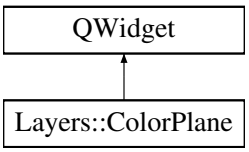
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/ColorDialog.h
- src/Widgets/Dialogs/ColorDialog.cpp

6.14 Layers::ColorPlane Class Reference

Inheritance diagram for Layers::ColorPlane:



Public Types

- enum class **Mode** { **Hue** , **Saturation** , **Value** }

Public Slots

- void **update_cursor_position** ()
- void **update_z_value** ()

Signals

- void **active_mode_changed** ()

Public Member Functions

- **ColorPlane** (QWidget ∗parent=nullptr)
- Mode **active_mode** () const
- float **pos_as_ratio** (int pos, int available_space)
- void **set_active_mode** (Mode new_active_hsv)
- void **setFixedHeight** (int h)
- void **setFixedSize** (const QSize &s)
- void **setFixedSize** (int w, int h)
- void **setFixedWidth** (int w)
- void **update_color** (float x_pos_ratio, float y_pos_ratio)
- void **update_height_dependencies** ()
- void **update_width_dependencies** ()

Public Attributes

- Attribute **color** { Attribute("color", QColor("#ff0000"))}
- Attribute **z_value** { Attribute("z_value", QVariant::fromValue(0.0)) }

Protected Member Functions

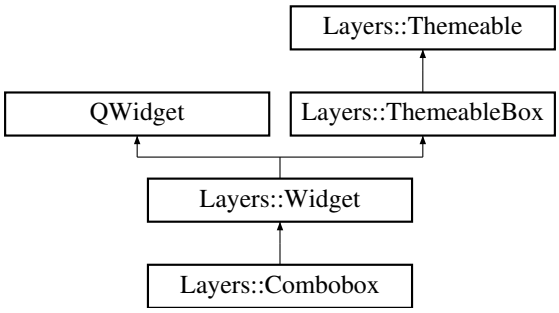
- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void **paintEvent** (QPaintEvent ∗event) override

The documentation for this class was generated from the following files:

- include/ColorPlane.h
- src/Widgets/ColorPlane.cpp

6.15 Layers::Combobox Class Reference

Inheritance diagram for Layers::Combobox:



Public Slots

- void **line_edit_return_pressed** ()

Signals

- void **current_item_changed** (const QString &current_item)
- void **item_replaced** (const QString &old_item, const QString &new_item)

Public Member Functions

- **Combobox** (QWidget ∗parent=nullptr)
- void **add_item** (const QString &item)
- void **alphabetize** ()
- void **edit_current_item** ()
- void **enable_alphabetization** (bool cond=true)
- void **set_current_item** (const QString &item)
- void **set_disabled** (bool cond=true)
- void **set_font_size** (int size)
- void **set_item_renaming_disabled** (bool disable=true)
- void **set_padding** (int left, int top, int right, int bottom)
- void **setFixedSize** (const QSize &s)
- void **setFixedSize** (int w, int h)
- QString **current_item** () const
- QList< QString > **items** ()
- void **update_theme_dependencies** ()

Public Attributes

- Attribute **a_line_edit_text_color** { Attribute("line_edit_text_color", QColor(Qt::black)) }

Protected Member Functions

- virtual bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.15.1 Member Function Documentation

6.15.1.1 init_child_themeable_reference_list()

```
void Combobox::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

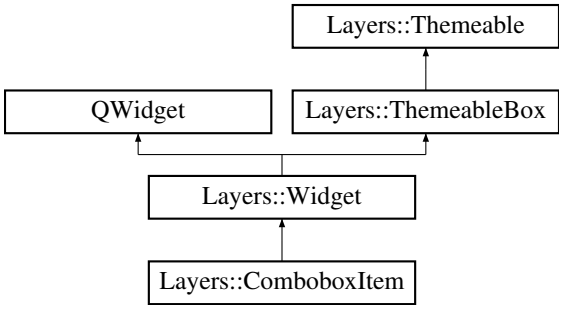
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Combobox.h
- src/Widgets/Controls/Combobox.cpp

6.16 Layers::ComboboxItem Class Reference

Inheritance diagram for Layers::ComboboxItem:



Public Member Functions

- **ComboboxItem** (const QString &item_text, QWidget *parent=nullptr)
- QString **item_text** ()
- void **replace_item_text** (const QString &new_item_text)
- void **set_font_size** (int size)
- void **setFixedSize** (const QSize &s)
- void **setFixedSize** (int w, int h)

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.16.1 Member Function Documentation

6.16.1.1 init_child_themeable_reference_list()

```
void ComboboxItem::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

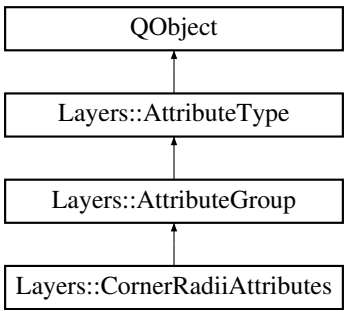
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Combobox.h
- src/Widgets/Controls/ComboboxItem.cpp

6.17 Layers::CornerRadiiAttributes Class Reference

Inheritance diagram for Layers::CornerRadiiAttributes:



Public Member Functions

- **CornerRadiiAttributes** (const QString &name="corner_radii")

Public Attributes

- Attribute bottom_left
- Attribute bottom_right
- Attribute top_left
- Attribute top_right

Additional Inherited Members

6.17.1 Member Data Documentation

6.17.1.1 bottom_left

```
Attribute Layers::CornerRadiiAttributes::bottom_left
```

Initial value:

```
{ Attribute(
            "bottom_left",
            QVariant::fromValue(0.0)
            ) }
```

6.17.1.2 bottom_right

```
Attribute Layers::CornerRadiiAttributes::bottom_right
```

Initial value:

```
{ Attribute(
            "bottom_right",
            QVariant::fromValue(0.0)
            ) }
```

6.17.1.3 top_left

```
Attribute Layers::CornerRadiiAttributes::top_left
```

Initial value:

```
{ Attribute(
            "top_left",
            QVariant::fromValue(0.0)
            ) }
```

6.17.1.4 top_right

```
Attribute Layers::CornerRadiiAttributes::top_right
```

Initial value:

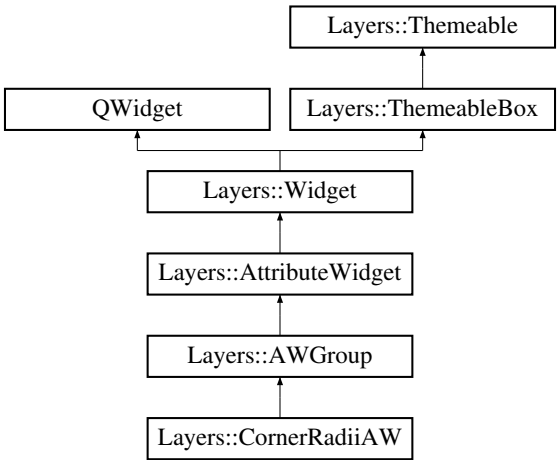
```
{ Attribute(
            "top_right",
            QVariant::fromValue(0.0)
            ) }
```

The documentation for this class was generated from the following files:

- include/AttributeGroup.h
- src/Themes/Attributes/AttributeGroup.cpp

6.18 Layers::CornerRadiiAW Class Reference

Inheritance diagram for Layers::CornerRadiiAW:



Public Slots

- void **set_current_editting_state** (const QString &state)

Public Member Functions

- **CornerRadiiAW** (CornerRadiiAttributes *linked_corner_radii, QWidget *parent=nullptr)
- MiniSlider * **tl_slider** () const
- MiniSlider * **tr_slider** () const
- MiniSlider * **bl_slider** () const
- MiniSlider * **br_slider** () const

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.18.1 Member Function Documentation

6.18.1.1 init_child_themeable_reference_list()

```
void CornerRadiiAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

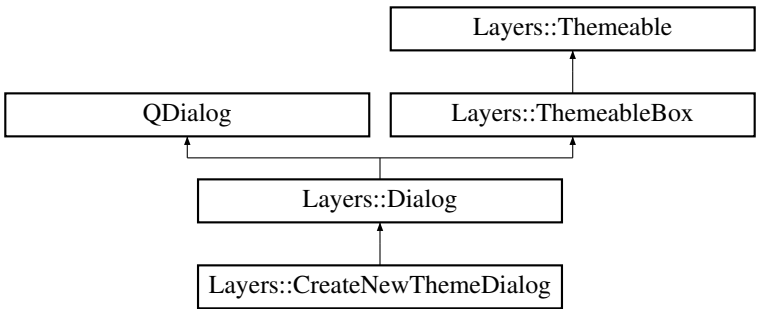
Reimplemented from Layers::AWGroup.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/CornerRadiiAW.cpp

6.19 Layers::CreateNewThemeDialog Class Reference

Inheritance diagram for Layers::CreateNewThemeDialog:



Public Member Functions

- **CreateNewThemeDialog** (QWidget *parent=nullptr)
- void **add_theme_name_to_combobox** (const QString &theme_name)
- void **clear** ()
- QString **copy_theme_name** ()
- QString **new_theme_name** ()
- void **set_current_start_theme_name** (const QString &theme_name)

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.19.1 Member Function Documentation

6.19.1.1 init_child_themeable_reference_list()

```
void CreateNewThemeDialog::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

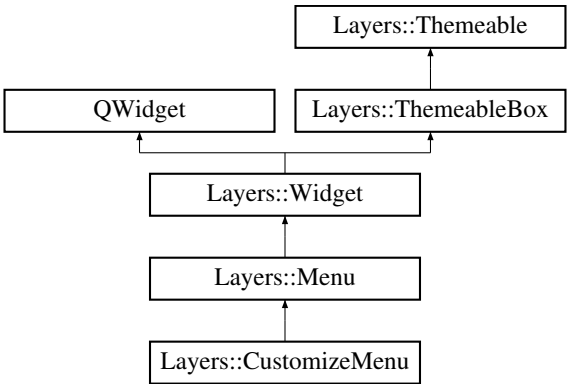
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/CreateNewThemeDialog.h
- src/Widgets/Dialogs/CreateNewThemeDialog.cpp

6.20 Layers::CustomizeMenu Class Reference

Inheritance diagram for Layers::CustomizeMenu:



Public Member Functions

- **CustomizeMenu** (QWidget ∗parent=nullptr)
- Button ∗ **apply_button** () const
- void **init_preview_window** ()
- void **open_customize_panel** (CustomizePanel ∗customize_panel)
- QList< CustomizePanel ∗ > & **panels** ()
- Widget ∗ **preview_widget** () const
- int **calculated_topbar_content_width** ()
- void **set_preview_widget** (Widget ∗widget)
- int **topbar_content_width** (bool include_collapse_button)

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.20.1 Member Function Documentation

6.20.1.1 init_child_themeable_reference_list()

```
void CustomizeMenu::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

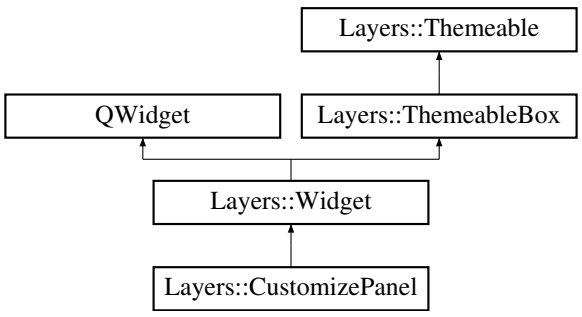
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/CustomizeMenu.h
- src/Widgets/Menus/CustomizeMenu.cpp

6.21 Layers::CustomizePanel Class Reference

Inheritance diagram for Layers::CustomizePanel:



Public Member Functions

- **CustomizePanel** (Themeable *themeable, QWidget *parent=nullptr)
- void **add_attribute_widget** (AttributeWidget *attribute_widget)
- void **add_widget_button** (Button *button, int index=-1)
- void **init_attribute_widgets** ()
- void **replace_all_aw_group_attrs_with** (AWGroup *control_aw_group)
- void **replace_all_color_awidgets_attrs_with** (ColorAW *control_color_aw)
- void **replace_all_fill_awidgets_attrs_with** (FillAW *control_fill_aw)
- void **replace_all_number_awidgets_attrs_with** (NumberAW *control_number_aw)
- void **replace_all_state_awidgets_attrs_with** (StateAW *control_state_aw)
- void **replace_all_widget_buttons_attrs_with** (Button *control_widget_button)
- void **replace_all_corner_radii_aw_attrs_with** (CornerRadiiAW *control_corner_radii_aw)

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.21.1 Member Function Documentation

6.21.1.1 init_child_themeable_reference_list()

```
void CustomizePanel::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

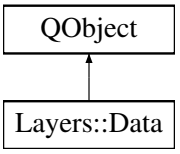
The documentation for this class was generated from the following files:

- include/CustomizePanel.h
- src/Widgets/CustomizePanel.cpp

6.22 Layers::Data Class Reference

`#include <Data.h>`

Inheritance diagram for Layers::Data:



Signals

- void **changed** ()

Public Member Functions

- **Data** (QVariant qvariant)
- **Data** (VariantMap variant_map)
- **Data** (const Data &d)
- template<typename T >
  T as (const QString &state="") const
- bool contains_state (const QString &state) const
- void copy (const Data &data)
- void init_variant_map (const VariantMap &variant_map)
- bool is_stateful () const
- void set_value (QVariant qvariant, bool retain_type=true)
- void set_value (const QString &state, QVariant qvariant)
- QList< QString > states () const
- QJsonObject to_json_object ()
- const char * typeName () const

6.22.1 Detailed Description

Data type which represents either a single Variant or multiple Variants

A Data object is a QObject that stores a pointer to either a single Variant or a QVariantMap where QString represents the state of the Variant.

Data associated with multiple Variants is stateful, whereas Data associated with a single Variant is not stateful. This can be referred to as the Data's statehood.

6.22.2 Member Function Documentation

6.22.2.1 as()

```
template<typename T >
T Layers::Data::as (
            const QString & state = "" ) const  [inline]
```

Returns value of the Variant associated with the supplied state, converted to the template type T.

Ignore the state parameter when calling this with Data that is not stateful.

Returns

Value of Variant associated with state, converted to template type T

6.22.2.2 contains_state()

```
bool Data::contains_state (
            const QString & state ) const
```

Returns true if state exists in variant map

Parameters

state	- State that might exist in variant map

Returns

True if state exists in variant map, false otherwise

6.22.2.3 copy()

```
void Data::copy (
            const Data & data )
```

Copies the supplied Data object.

If the statefulness between the data objects is the same, then the values are simply copied over. If it is not the same, the statefulness of the caller Data is converted to the statefulness of the supplied Data before the values are copied.

6.22.2.4 init_variant_map()

```
void Data::init_variant_map (
            const VariantMap & variant_map )
```

Converts to stateful Data initialized with the supplied map.

Parameters

state_variant_map	- Map to initialize the Data with

6.22.2.5 is_stateful()

```
bool Data::is_stateful ( ) const
```

Returns true if stateful, otherwise, returns false.

Returns

True if stateful, false otherwise

6.22.2.6 set_value() [1/2]

```
void Data::set_value (
            const QString & state,
            QVariant qvariant )
```

Set the value of the stored Variant associated with state.

This function will only work with Data objects that are stateful.

Parameters

state	- State associated with value
qvariant	- QVariant containing the value being set

6.22.2.7 set_value() [2/2]

```
void Data::set_value (
            QVariant qvariant,
            bool retain_type = true )
```

Set the value of the stored Variant.

This function will only work with Data objects that are not stateful.

Parameters

qvariant	- QVariant containing the value being set
retain_type	- Whether to protect the value type from change, true by default

6.22.2.8 states()

```
QList< QString > Data::states ( ) const
```

Returns a list of QStrings representing the available states.

If the Data is not stateful, then an empty list will be returned.

Returns

QString list where QStrings represent the states

6.22.2.9 to_json_object()

```
QJsonObject Data::to_json_object ( )
```

Returns Data converted to a QJsonObject.

Returns

QJsonObject pertaining to the Data

6.22.2.10 typeName()

```
const char * Data::typeName ( ) const
```

Returns the name of the type stored in the Data.

If the Data is not stateful, the type name of the single Variant is returned. If it is stateful, then the type name of the Variant associated with the first key state of the state-Variant map is returned.

Returns

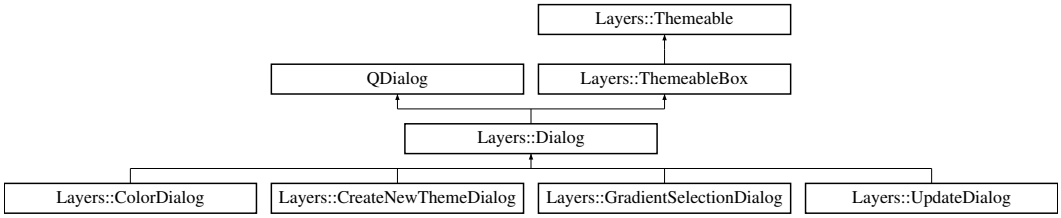
Name of type stored within the Data

The documentation for this class was generated from the following files:

- include/Data.h
- src/Themes/Attributes/Data.cpp

6.23 Layers::Dialog Class Reference

Inheritance diagram for Layers::Dialog:



Public Slots

- void **update_content_margins** ()
- void **update_titlebar** ()

Public Member Functions

- **Dialog** (const QString &title="Dialog", QWidget ∗parent=nullptr)
- void **setLayout** (QLayout ∗layout)

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()
- bool **nativeEvent** (const QByteArray &eventType, void ∗message, qintptr ∗result) override
- void **paintEvent** (QPaintEvent ∗event) override

Protected Attributes

- QVBoxLayout ∗ **m_main_layout** { new QVBoxLayout }

Additional Inherited Members

6.23.1 Member Function Documentation

6.23.1.1 init_child_themeable_reference_list()

```
void Dialog::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

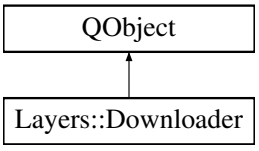
Reimplemented in Layers::GradientSelectionDialog, and Layers::UpdateDialog.

The documentation for this class was generated from the following files:

- include/Dialog.h
- src/Widgets/Dialogs/Dialog.cpp

6.24 Layers::Downloader Class Reference

Inheritance diagram for Layers::Downloader:



Public Member Functions

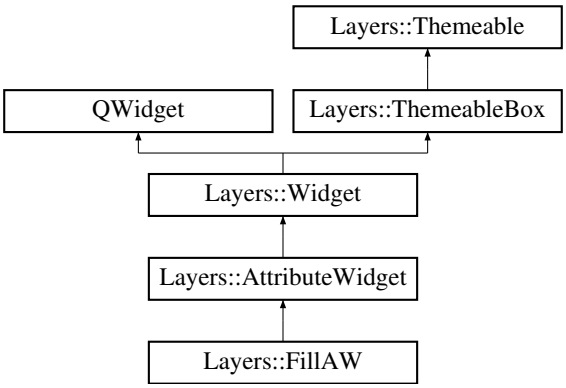
- **Downloader** (QObject *parent=0)
- QNetworkReply * **download** (const QUrl &file_url, const QDir &directory)
- QNetworkReply * **download** (const QUrl &file_url)

The documentation for this class was generated from the following files:

- include/Downloader.h
- src/Tools/Downloader.cpp

6.25 Layers::FillAW Class Reference

Inheritance diagram for Layers::FillAW:



Public Slots

- virtual void **set_current_editting_state** (const QString &state) override

Public Member Functions

- **FillAW** (Attribute *attribute, QWidget *parent=nullptr)
- FillControl * **fill_control** () const

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.25.1 Member Function Documentation

6.25.1.1 init_child_themeable_reference_list()

```
void FillAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

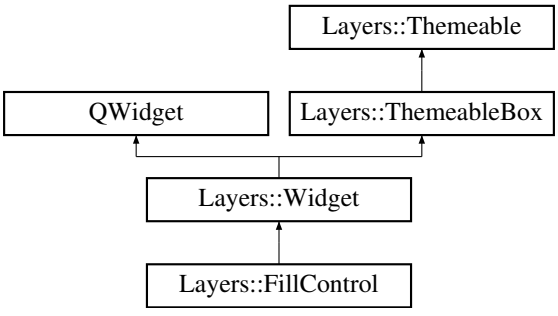
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/FillAW.cpp

6.26 Layers::FillControl Class Reference

Inheritance diagram for Layers::FillControl:



Public Slots

- void **set_current_editting_state** (const QString &state)

Public Member Functions

- **FillControl** (QWidget *parent=nullptr)
- void init_child_themeable_reference_list ()
- void **set_attribute** (Attribute *attribute)

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event)
- void **init_attributes** ()

Additional Inherited Members

6.26.1 Member Function Documentation

6.26.1.1 init_child_themeable_reference_list()

```
void FillControl::init_child_themeable_reference_list ( )  [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/FillControl.h
- src/Widgets/Controls/FillControl.cpp

6.27 Layers::GitHubRepo Class Reference

Public Member Functions

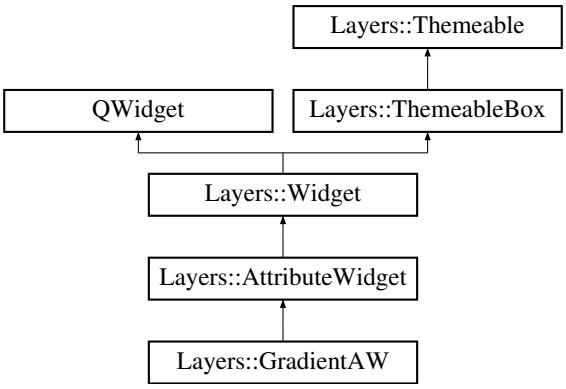
- **GitHubRepo** (const QString &repo_url)
- QString **toString** () const

The documentation for this class was generated from the following files:

- include/GitHubRepo.h
- src/Tools/GitHubRepo.cpp

6.28 Layers::GradientAW Class Reference

Inheritance diagram for Layers::GradientAW:



Public Member Functions

- **GradientAW** (const QString &attribute_label_text, Attribute *attribute, QWidget *parent=nullptr)
- void **set_centered** (bool centered=true)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.28.1 Member Function Documentation

6.28.1.1 init_child_themeable_reference_list()

```
void GradientAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

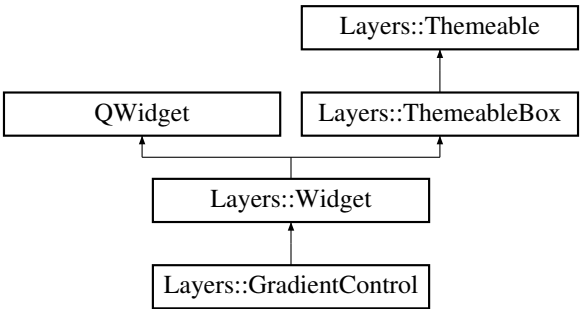
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/GradientAW.cpp

6.29 Layers::GradientControl Class Reference

Inheritance diagram for Layers::GradientControl:



Public Slots

- void **set_current_editting_state** (const QString &state)

Signals

- void **gradient_changed** ()

Public Member Functions

- **GradientControl** (QWidget ∗parent=nullptr)

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event)
- void **init_attributes** ()

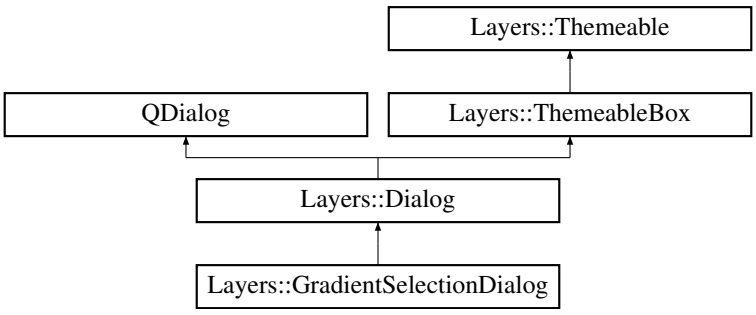
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/GradientControl.h
- src/Widgets/Controls/GradientControl.cpp

6.30 Layers::GradientSelectionDialog Class Reference

Inheritance diagram for Layers::GradientSelectionDialog:



Public Slots

- void **click_control** ()
- void **update_color_control_positions** ()

Public Member Functions

- **GradientSelectionDialog** (QGradientStops gradient_stops, QWidget *parent=nullptr)
- void **add_gradient_stop** (double stop_val, QColor color)
- QGradientStops **gradient_stops** () const
- void **update_gradient** ()

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.30.1 Member Function Documentation

6.30.1.1 init_child_themeable_reference_list()

```
void GradientSelectionDialog::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

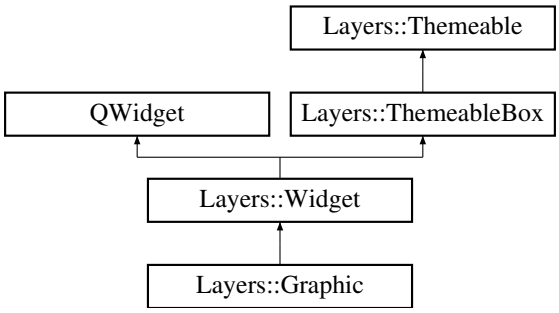
Reimplemented from Layers::Dialog.

The documentation for this class was generated from the following files:

- include/GradientSelectionDialog.h
- src/Widgets/Dialogs/GradientSelectionDialog.cpp

6.31 Layers::Graphic Class Reference

Inheritance diagram for Layers::Graphic:



Public Member Functions

- **Graphic** (const ImageSequence &image_sequence, QSize size, QWidget ∗parent=nullptr)
- **Graphic** (const QString &filepath, QSize size, QWidget ∗parent=nullptr)
- **Graphic** (const QString &filepath, QWidget ∗parent=nullptr)
- **Graphic** (const QImage &image, QWidget ∗parent=0)
- **Graphic** (const Graphic &gw)
- QSize **image_size** ()
- void **set_hovering** (bool cond=true)
- void **set_icon** (Graphic ∗icon)
- void **set_pixmap** (const QPixmap &pixmap)
- void **set_size** (QSize size)
- SVG ∗ **svg** () const

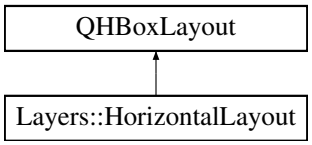
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/Graphic.h
- src/Widgets/Graphic.cpp

6.32 Layers::HorizontalLayout Class Reference

Inheritance diagram for Layers::HorizontalLayout:



Public Member Functions

- **HorizontalLayout** (QWidget ∗parent=nullptr)
- void **set_border_margin** (int border_margin)
- void **setContentsMargins** (int left, int top, int right, int bottom)
- void **update_margins** ()

Protected Attributes

- int **m_margin_left** { 0 }
- int **m_margin_top** { 0 }
- int **m_margin_right** { 0 }
- int **m_margin_bottom** { 0 }
- int **m_border_margin** { 0 }

The documentation for this class was generated from the following files:

- include/Layouts.h
- src/Layouts/HorizontalLayout.cpp

6.33 Layers::ImageSequence Class Reference

Public Member Functions

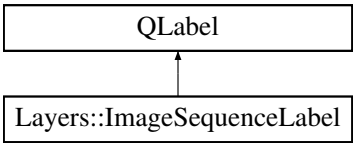
- **ImageSequence** (QDir dir)
- **ImageSequence** (QFile file)
- void **save** (QFile file)
- QList< QPixmap > **to_pixmaps** () const

The documentation for this class was generated from the following files:

- include/ImageSequence.h
- src/Tools/ImageSequence.cpp

6.34 Layers::ImageSequenceLabel Class Reference

Inheritance diagram for Layers::ImageSequenceLabel:



Public Slots

- void **time_out** ()

Public Member Functions

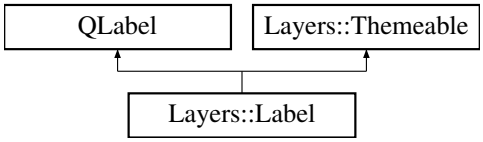
- **ImageSequenceLabel** (ImageSequence image_sequence, QSize size, QWidget *parent=nullptr)
- **ImageSequenceLabel** (const ImageSequenceLabel &isl)

The documentation for this class was generated from the following files:

- include/ImageSequenceLabel.h
- src/Widgets/ImageSequenceLabel.cpp

6.35 Layers::Label Class Reference

Inheritance diagram for Layers::Label:



Public Slots

- void **setText** (const QString &text)

Public Member Functions

- **Label** (QWidget *parent=nullptr)
- **Label** (const QString &text, QWidget *parent=0)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void **resize** ()
- void **build_wrapped_lines** ()
- void **setFont** (const QFont &f)
- void **setMaximumWidth** (int maxw)
- void **setWordWrap** (bool on)
- void **set_available_width** (int available_width)
- void **set_font_size** (int size)
- void **set_hovering** (bool cond=true)
- void **set_padding** (double left, double top, double right, double bottom)
- void **set_resize_disabled** (bool disable=true)
- int **width_unwrapped** ()

Public Attributes

- Attribute **a_fill** { Attribute("fill", QColor(Qt::white), true) }
- Attribute **a_outline_color** { Attribute("outline_color", QColor(Qt::gray), true) }
- Attribute **a_padding_top** { Attribute("top_padding", QVariant::fromValue(0.0)) }
- Attribute **a_text_color** { Attribute("text_color", QColor(Qt::black)) }
- Attribute **a_text_hover_color** { Attribute("text_hover_color", QColor(Qt::black), true) }

Protected Member Functions

- void **init_attributes** ()
- void **paintEvent** (QPaintEvent ∗event)

Protected Attributes

- QList< QString > **m_wrapped_lines**
- QPainter **painter**
- bool **m_hovering** { false }
- bool **m_resize_disabled** { false }
- bool **m_wrapping** { false }
- int **m_available_width** { 16777215 }
- int **m_padding_left** { 0 }
- int **m_padding_right** { 0 }
- int **m_padding_bottom** { 0 }

6.35.1 Member Function Documentation

6.35.1.1 apply_theme_attributes()

```
void Label::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

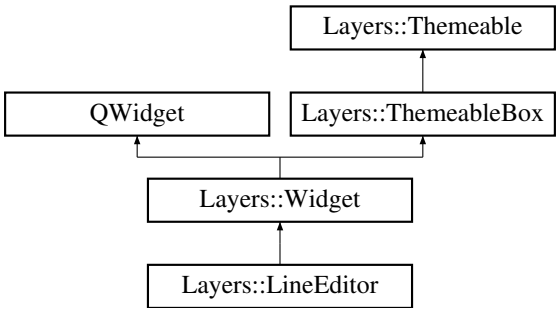
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Label.h
- src/Widgets/Label.cpp

6.36 Layers::LineEditor Class Reference

Inheritance diagram for Layers::LineEditor:



Public Slots

- void **set_current_editting_state** (const QString &state)
- void **update_theme_dependencies** ()

Signals

- void **text_edited** (const QString &text)

Public Member Functions

- **LineEditor** (QWidget *parent=nullptr)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void **reconnect_text_attribute** ()
- void **set_default_value** (const QString &default_value)
- void **set_disabled** (bool cond=true)
- void **set_font_size** (int size)
- void **set_margin** (int margin)
- void **set_margin** (int left, int top, int right, int bottom)
- void **set_text** (const QString &text)
- void **set_validator** (const QValidator *validator)
- void **setFixedSize** (int w, int h)
- void **setFixedWidth** (int w)
- QString **text** ()

Public Attributes

- Attribute **a_left_padding** { Attribute("left_padding", QVariant::fromValue(3.0)) }
- Attribute **a_text_color** { Attribute("text_color", QColor(Qt::black)) }
- Attribute **a_text** { Attribute("text", QString("")) }

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event) override
- void **init_attributes** ()

Additional Inherited Members

6.36.1 Member Function Documentation

6.36.1.1 apply_theme_attributes()

```
void LineEditor::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

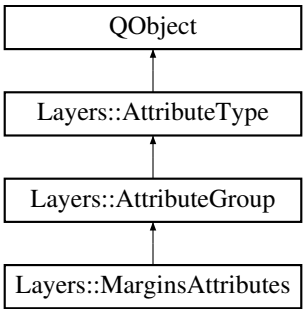
Reimplemented from Layers::ThemeableBox.

The documentation for this class was generated from the following files:

- include/LineEditor.h
- src/Widgets/Controls/LineEditor.cpp

6.37 Layers::MarginsAttributes Class Reference

Inheritance diagram for Layers::MarginsAttributes:



Public Member Functions

- **MarginsAttributes** (const QString &name="margins")

Public Attributes

- Attribute left
- Attribute top
- Attribute right
- Attribute bottom

Additional Inherited Members

6.37.1 Member Data Documentation

6.37.1.1 bottom

```
Attribute Layers::MarginsAttributes::bottom
```

Initial value:

```
{ Attribute(
            "bottom",
            QVariant::fromValue(0.0)
            ) }
```

6.37.1.2 left

```
Attribute Layers::MarginsAttributes::left
```

Initial value:

```
{ Attribute(
            "left",
            QVariant::fromValue(0.0)
            ) }
```

6.37.1.3 right

```
Attribute Layers::MarginsAttributes::right
```

Initial value:

```
{ Attribute(
            "right",
            QVariant::fromValue(0.0)
            ) }
```

6.37.1.4 top

```
Attribute Layers::MarginsAttributes::top
```

Initial value:

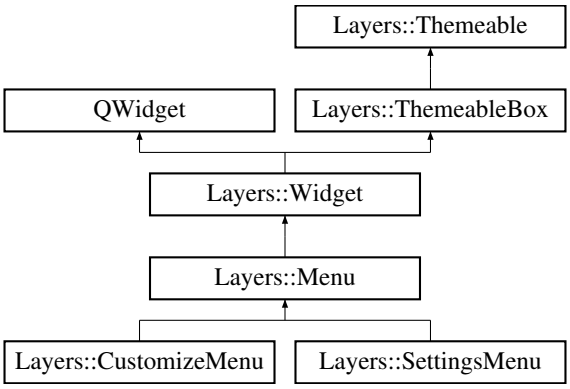
```
{ Attribute(
            "top",
            QVariant::fromValue(0.0)
            ) }
```

The documentation for this class was generated from the following files:

- include/AttributeGroup.h
- src/Themes/Attributes/AttributeGroup.cpp

6.38 Layers::Menu Class Reference

Inheritance diagram for Layers::Menu:



Public Member Functions

- **Menu** (const QString &menu_name, Graphic *menu_icon, QWidget *parent=nullptr)

Public Attributes

- Graphic * **icon** { nullptr }
- QString **name**

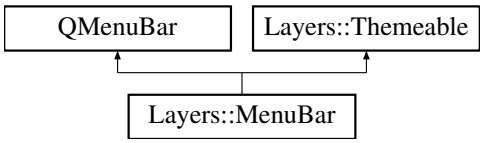
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/Menu.h
- src/Widgets/Menus/Menu.cpp

6.39 Layers::MenuBar Class Reference

Inheritance diagram for Layers::MenuBar:



Public Member Functions

- **MenuBar** (QWidget *parent=0)
- QMenu * **addMenu** (const QString &title)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void **update_theme_dependencies** ()

Public Attributes

- Attribute **a_text_color** { Attribute("text_color", QColor(Qt::gray)) }
- Attribute **a_selected_text_color** { Attribute("selected_text_color", QColor(Qt::lightGray)) }

Protected Member Functions

- QString **build_stylesheet** ()
- void **init_attributes** ()

Additional Inherited Members

6.39.1 Member Function Documentation

6.39.1.1 apply_theme_attributes()

```
void MenuBar::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

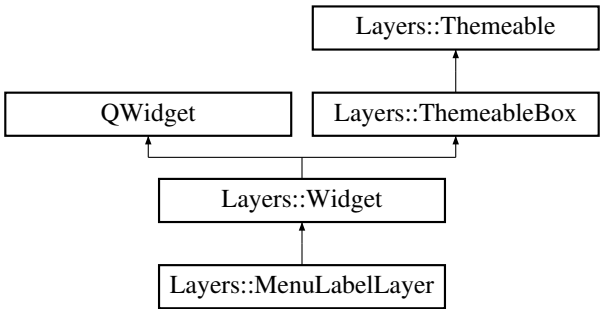
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/MenuBar.h
- src/Widgets/MenuBar.cpp

6.40 Layers::MenuLabelLayer Class Reference

Inheritance diagram for Layers::MenuLabelLayer:



Public Member Functions

- **MenuLabelLayer** (Menu *menu, QWidget *parent=nullptr)
- void **shrink** ()
- void **expand** ()
- Button * **back_button** () const
- Button * **icon_button** () const
- Label * **text_label** () const

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()
- void **setup_layout** ()

Additional Inherited Members

6.40.1 Member Function Documentation

6.40.1.1 init_child_themeable_reference_list()

```
void MenuLabelLayer::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

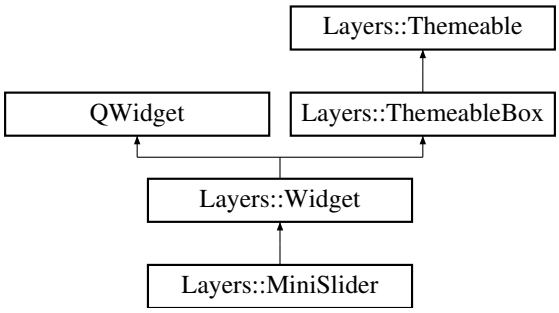
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/MenuLabelLayer.h
- src/Widgets/MenuLabelLayer.cpp

6.41 Layers::MiniSlider Class Reference

Inheritance diagram for Layers::MiniSlider:



Public Slots

- void **set_current_editting_state** (const QString &state)

Public Member Functions

- **MiniSlider** (double limit, QWidget *parent=nullptr)
- void **update_handle_pos** ()
- void **update_theme_dependencies** ()

Public Attributes

- Attribute **a_value** { Attribute("value", QVariant::fromValue(0.0)) }

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.41.1 Member Function Documentation

6.41.1.1 init_child_themeable_reference_list()

```
void MiniSlider::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

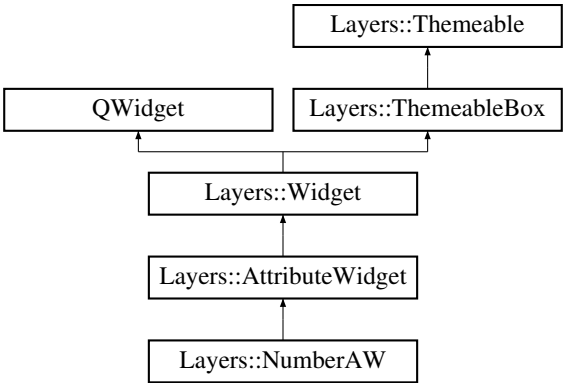
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/MiniSlider.h
- src/Widgets/Controls/MiniSlider.cpp

6.42 Layers::NumberAW Class Reference

Inheritance diagram for Layers::NumberAW:



Public Member Functions

- **NumberAW** (Attribute *attribute, QIntValidator *int_validator, QWidget *parent=nullptr)
- void **set_centered** (bool centered=true)
- void **set_unit_label_text** (const QString &unit_string)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.42.1 Member Function Documentation

6.42.1.1 init_child_themeable_reference_list()

```
void NumberAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

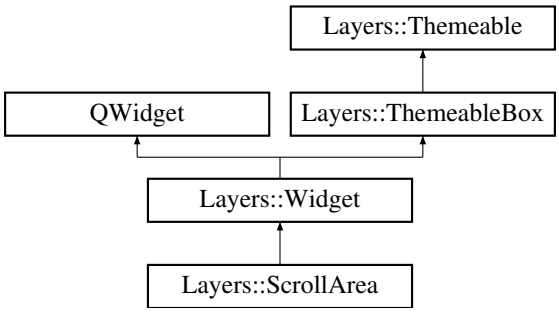
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/NumberAW.cpp

6.43 Layers::ScrollArea Class Reference

Inheritance diagram for Layers::ScrollArea:



Public Member Functions

- **ScrollArea** (QWidget ∗parent=nullptr)
- ScrollBar ∗ **horizontal_scrollbar** () const
- void **setHorizontalScrollBarPolicy** (Qt::ScrollBarPolicy policy)
- void **setVerticalScrollBarPolicy** (Qt::ScrollBarPolicy policy)
- void **setWidget** (QWidget ∗widget)
- ScrollBar ∗ **vertical_scrollbar** () const

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void init_child_themeable_reference_list ()

Protected Attributes

- QScrollArea ∗ **m_scroll_area** { new QScrollArea(this) }
- ScrollBar ∗ **m_horizontal_scrollbar** { new ScrollBar }
- ScrollBar ∗ **m_vertical_scrollbar** { new ScrollBar }

Additional Inherited Members

6.43.1 Member Function Documentation

6.43.1.1 init_child_themeable_reference_list()

```
void ScrollArea::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

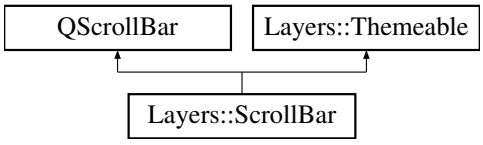
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/ScrollArea.h
- src/Widgets/ScrollArea.cpp

6.44 Layers::ScrollBar Class Reference

Inheritance diagram for Layers::ScrollBar:



Public Member Functions

- **ScrollBar** (QWidget *parent=0)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void **update_theme_dependencies** ()

Public Attributes

- Attribute **a_background_color** { Attribute("background_color", QColor(Qt::gray)) }
- Attribute **a_handle_color** { Attribute("handle_color", QColor(Qt::white)) }
- CornerRadiiAttributes **corner_radii**
- CornerRadiiAttributes **handle_corner_radii** { CornerRadiiAttributes("handle_corner_radii") }

Protected Member Functions

- QString **build_stylesheet** ()
- void **init_attributes** ()

Additional Inherited Members

6.44.1 Member Function Documentation

6.44.1.1 apply_theme_attributes()

```
void ScrollBar::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

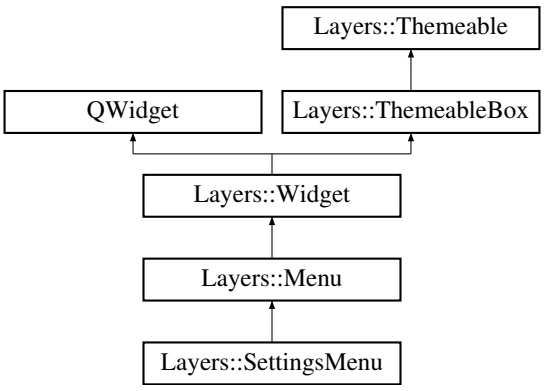
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/ScrollBar.h
- src/Widgets/Scrollbar.cpp

6.45 Layers::SettingsMenu Class Reference

Inheritance diagram for Layers::SettingsMenu:



Public Member Functions

- **SettingsMenu** (QWidget *parent=nullptr)
- void **add_settings_tab** (Graphic *icon, const QString &label_text)
- int **largest_tab_index** () const
- int **recommended_minimum_tab_width** () const
- ThemesSettingsPanel * **themes_settings_panel** () const

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event) override
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.45.1 Member Function Documentation

6.45.1.1 init_child_themeable_reference_list()

```
void SettingsMenu::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

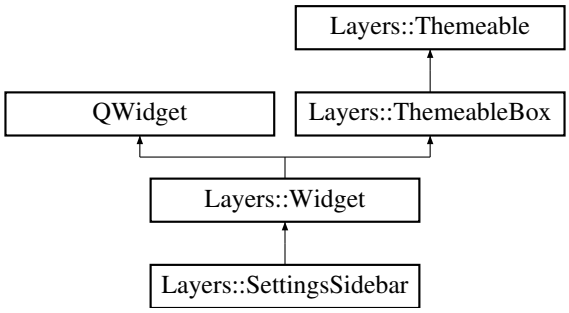
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/SettingsMenu.h
- src/Widgets/Menus/Settings/SettingsMenu.cpp

6.46 Layers::SettingsSidebar Class Reference

Inheritance diagram for Layers::SettingsSidebar:



Public Member Functions

- **SettingsSidebar** (QWidget ∗parent=nullptr)

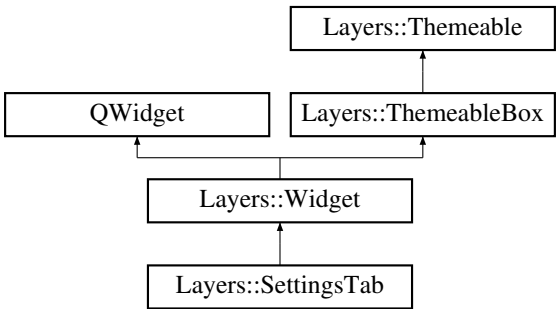
Additional Inherited Members

The documentation for this class was generated from the following files:

- include/SettingsMenu.h
- src/Widgets/Menus/Settings/SettingsSidebar.cpp

6.47 Layers::SettingsTab Class Reference

Inheritance diagram for Layers::SettingsTab:



Signals

- void **clicked** ()
- void **under_minimum_width** ()
- void **over_minimum_width** ()

Public Member Functions

- **SettingsTab** (Graphic *icon, const QString &label_text, QWidget *parent=nullptr)
- void **expand** ()
- void **shrink** ()
- int **recommended_minimum_width** ()
- void **set_disabled** (bool cond=true)

Protected Member Functions

- bool **eventFilter** (QObject *object, QEvent *event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()
- void **resizeEvent** (QResizeEvent *event)

Additional Inherited Members

6.47.1 Member Function Documentation

6.47.1.1 init_child_themeable_reference_list()

```
void SettingsTab::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

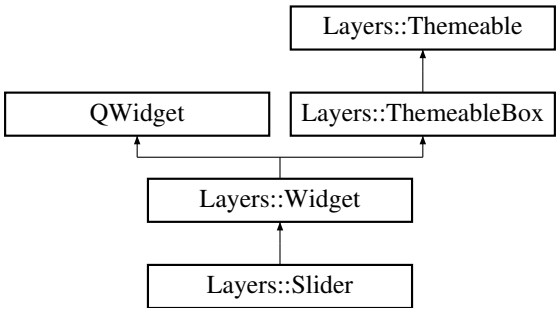
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/SettingsMenu.h
- src/Widgets/Menus/Settings/SettingsTab.cpp

6.48 Layers::Slider Class Reference

Inheritance diagram for Layers::Slider:



Public Slots

- void **update_handle_pos** ()

Signals

- void **value_changed** (int value)

Public Member Functions

- **Slider** (QWidget ∗parent=nullptr)
- **Slider** (int limit, QWidget ∗parent=nullptr)
- void **set_limit** (int limit)
- void **set_value** (double value)

Public Attributes

- Attribute **a_value** { Attribute("value", QVariant::fromValue(0.0)) }

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.48.1 Member Function Documentation

6.48.1.1 init_child_themeable_reference_list()

```
void Slider::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

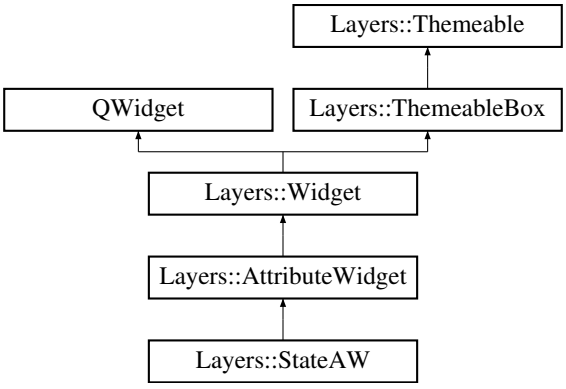
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Slider.h
- src/Widgets/Controls/Slider.cpp

6.49 Layers::StateAW Class Reference

Inheritance diagram for Layers::StateAW:



Public Member Functions

- **StateAW** (QWidget *parent=nullptr)
- void **add_attribute_widget** (AttributeWidget *attribute_widget)
- Combobox * **state_combobox** () const
- void **populate_state_combobox** (const QList< QString > &states)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.49.1 Member Function Documentation

6.49.1.1 init_child_themeable_reference_list()

```
void StateAW::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

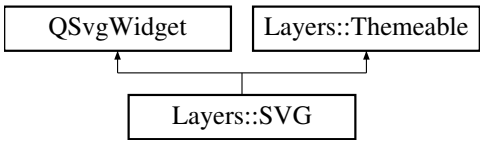
The documentation for this class was generated from the following files:

- include/AttributeWidgets.h
- src/Widgets/Attribute Widgets/StateAW.cpp

6.50 Layers::SVG Class Reference

`#include <SVG.h>`

Inheritance diagram for Layers::SVG:



Public Member Functions

- SVG (QString file_path, QWidget *parent=nullptr)
- SVG (const SVG &svg_w)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void rebuild_svg_str ()
- void **set_hovering** (bool cond=true)
- virtual void set_state (const QString &state) override
- void update ()

Public Attributes

- Attribute **a_common_color** { Attribute("common_color", QColor(Qt::black)) }
- Attribute **a_common_hover_color** { Attribute("common_hover_color", QColor(Qt::darkGray)) }
- Attribute **a_use_common_color** { Attribute("use_common_color", QVariant::fromValue(false)) }
- Attribute **a_use_common_hover_color** { Attribute("use_common_hover_color", QVariant::fromValue(false)) }

Protected Member Functions

- void init_attributes ()

Additional Inherited Members

6.50.1 Detailed Description

The SVG class provides representation for SVG files in Layers.

An SVG loads an SVG file into a string. To make the SVG appear on the screen, the string is passed to QSvg↩ Widget::load(). The load function can be called indefinitely. Therefore, changes can be made to the SVG string and loaded again, allowing for theme application.

Manipulating the SVG string itself would be punishing. To make this easier, a list of SVG elements is built from the string with build_svg_elements_list().

The SVG's constructors wait to call Themeable::init_themeable() until after the SVG elements list has been built. This is because the SVG::init_attributes() function, which is called virtually by Themeable::init_themeable(), depends on the elements list.

When a theme is applied to the SVG, the attributes are updated to values provided in the applied theme, but the SVG string needs to be updated with the new attribute values and passed along to QSvgWidget::load() before the appearance of the SVG updates. This class implements update_theme_dependencies() to ensure that this all happens after theme application.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 SVG() [1/2]

```
SVG::SVG (
            QString file_path,
            QWidget * parent = nullptr )
```

Constructs an SVG from an SVG file.

6.50.2.2 SVG() [2/2]

```
SVG::SVG (
            const SVG & svg_w )
```

Copy constructs an SVG from another SVG. Call init_attributes() AFTER m_svg_elements and attributes have been copied

6.50.3 Member Function Documentation

6.50.3.1 apply_theme_attributes()

```
void SVG::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

Reimplemented from Layers::Themeable.

6.50.3.2 init_attributes()

```
void SVG::init_attributes ( )  [protected]
```

Initializes the attributes.

This function is called by init_themeable().

6.50.3.3 rebuild_svg_str()

```
void SVG::rebuild_svg_str ( )
```

Rebuilds the SVG string from the SVG elements list.

6.50.3.4 set_state()

```
void SVG::set_state (
            const QString & state )  [override], [virtual]
```

Sets the current state of the themeable.

This function calls issue_update() to ensure the state change is visually represented.

Parameters

state	to set the themeable to

Reimplemented from Layers::Themeable.

6.50.3.5 update()

```
void SVG::update ( )
```

Updates things that depend on the theme.

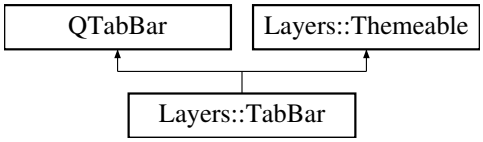
SVG elements are updated with the newly applied theme's attribute values. Then, the SVG string is rebuilt from those elements. And finally, the SVG string is passed to QSvgWidget::load() to update the SVG's appearance.

The documentation for this class was generated from the following files:

- include/SVG.h
- src/Widgets/SVG.cpp

6.51 Layers::TabBar Class Reference

Inheritance diagram for Layers::TabBar:



Public Member Functions

- **TabBar** (QWidget *parent=0)
- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void **SetCurrentTab** (const QString &text)
- bool **ContainsTab** (const QString &text)
- void **removeTab** (const QString &text)
- void **update_theme_dependencies** ()

Public Attributes

- Attribute **a_selected_fill_color** { Attribute("selected_fill_color", QColor(Qt::gray)) }
- Attribute **a_text_color** { Attribute("text_color", QColor(Qt::white)) }

Protected Member Functions

- QString **build_stylesheet** ()
- void **init_attributes** ()

Additional Inherited Members

6.51.1 Member Function Documentation

6.51.1.1 apply_theme_attributes()

```
void TabBar::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/TabBar.h
- src/Widgets/TabBar.cpp

6.52 Layers::Theme Class Reference

```
#include <Theme.h>
```

Public Member Functions

- **Theme** (const QString &name, bool editable=true)
- **Theme** (const QJsonDocument &json_document, QUuid *uuid=nullptr)
- void clear ()
- void **consume** (Theme &&theme)
- bool contains_attributes_for_tag (const QString &themeable_tag)
- void copy (Theme &theme)
- void **copy_attribute_values_of** (Themeable *themeable)
- bool editable ()
- QString **identifier** ()
- Attribute * **init_attribute** (const QString &name, bool disabled, const QJsonValue &attr_value)
- QString & name ()
- void set_name (const QString &new_name)
- QList< QString > themeable_tags ()
- QJsonDocument **to_json_document** (ThemeDataType data_type=ThemeDataType::All)
- QMap< QString, AttributeType * > & operator[ ] (const QString &themeable_tag)

6.52.1 Detailed Description

Provides structure for Layers themes.

Layers themes store sets of attributes associated with their themeable tags. When a theme is applied, themeables retrieve their attribute sets from the theme by passing along their tags.

6.52.2 Member Function Documentation

6.52.2.1 clear()

```
void Theme::clear ( )
```

Adds a themeable tag paired with a set of attributes

Does nothing if themeable tag already exists in the theme.

Parameters

themeable_tag	of the themeable that the supplied attributes belong to
attributes	that belong to a themeable for this theme to store

6.52.2.2 contains_attributes_for_tag()

```
bool Theme::contains_attributes_for_tag (
            const QString & themeable_tag )
```

Returns true if the theme contains any attributes for the given themeable tag; otherwise returns false.

Parameters

themeable_tag	used to check whether the theme contains attributes for it

Returns

True if theme contains attributes for tag, false otherwise

6.52.2.3 copy()

```
void Theme::copy (
            Theme & theme )
```

Copies the attribute sets of another theme.

Parameters

theme	to copy attribute sets from

6.52.2.4 editable()

```
bool Theme::editable ( )
```

Returns true if the theme is a custom, user-created theme

Returns

true if theme is custom, false otherwise

6.52.2.5 name()

```
QString & Theme::name ( )
```

Returns a reference to the theme's name

Returns

Reference to theme's name

6.52.2.6 operator[]()

```
QMap< QString, AttributeType * > & Theme::operator[] (
            const QString & themeable_tag )
```

Returns a reference to the attribute set of the themeable tag given in the subscript.

This function does NOT check first whether the supplied themeable tag exists in the theme. For this reason, it is recommended to call contains_attributes_for_tag() first.

Returns

Reference to attribute set of themeable_tag

6.52.2.7 set_name()

```
void Theme::set_name (
            const QString & new_name )
```

Sets the theme's name

Parameters

new_name	to set as the theme's name

6.52.2.8 themeable_tags()

```
QList< QString > Theme::themeable_tags ( )
```

Returns a list of all of the themeable tags contained in the theme

Returns

list of themeable tags that exist in the theme

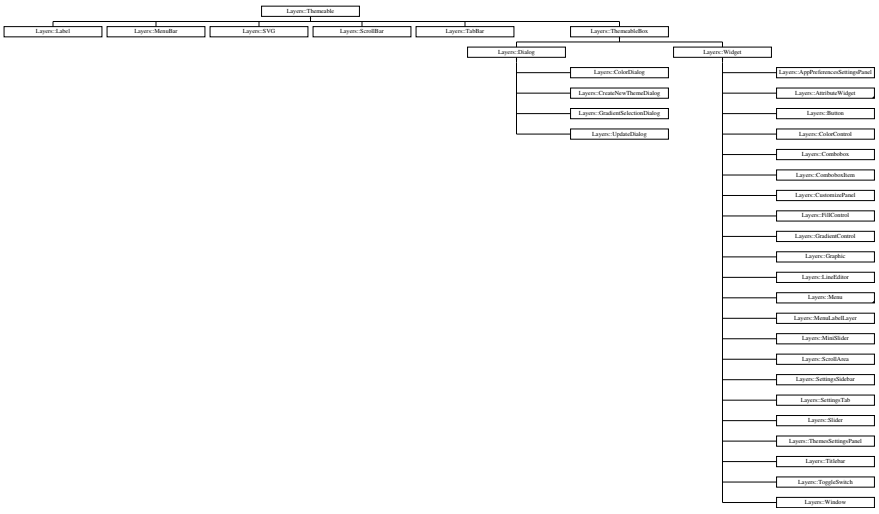
The documentation for this class was generated from the following files:

- include/Theme.h
- src/Themes/Theme.cpp

6.53 Layers::Themeable Class Reference

```
#include <Themeable.h>
```

Inheritance diagram for Layers::Themeable:



Public Member Functions

- void store_child_themeable_pointer (Themeable *child_themeable)
- virtual void apply_theme (Theme &theme)
- virtual void **apply_theme_attributes** (QMap< QString, AttributeType * > &theme_attrs)
- void assign_tag_prefixes (QList< QString > parent_prefixes=QList< QString >(), const QString &parent_name="")
- QMap< QString, AttributeType * > & attributes ()
- QList< AttributeType * > & **attribute_layout** ()
- QList< Themeable * > & **child_themeable_references** ()
- void **copy_attribute_values_to** (Theme *theme)
- Theme * current_theme ()
- CustomizePanel * customize_panel ()
- Graphic * icon () const
- bool is_stateful () const
- QString * name () const
- QString * proper_name () const
- void reapply_theme ()
- void remove_child_themeable_reference (Themeable *child_themeable)
- template<typename T >
  void **replace_all_attributes_with** (T *themeable)
- void **set_is_app_themeable** (bool is_app_themeable)
- void **set_functionality_disabled** (bool disabled=true)
- void set_icon (Graphic *icon)
- void set_name (const QString &name)
- void set_proper_name (const QString &proper_name)
- virtual void set_state (const QString &state)
- QString **state** () const
- QList< QString > states () const
- QString & tag ()
- void unassign_prefixes ()

Protected Member Functions

- virtual void init_child_themeable_reference_list ()

Protected Attributes

- bool **m_functionality_disabled** { false }
- bool **m_tag_prefixes_assigned** { false }
- bool **m_shared_attributes** { false }
- bool **m_is_app_themeable** { false }
- bool **m_is_stateful** { false }
- CustomizePanel * **m_customize_panel** { nullptr }
- Graphic * **m_icon** { nullptr }
- QString * **m_name** { nullptr }
- QString * **m_proper_name** { nullptr }
- QString **m_state** { "" }
- QString **m_tag** { "" }
- QList< AttributeType * > **m_attribute_layout** { QList<AttributeType*>() }
- QMap< QString, AttributeType * > **m_attributes** { QMap<QString, AttributeType*>() }
- QList< Themeable * > **m_child_themeables**
- QList< QString > **m_filtered_attributes**
- QList< QString > **m_tag_prefixes**
- Theme * **m_current_theme** { nullptr }

6.53.1 Detailed Description

Provides compatibility and structure for the Layers theme system.

The Themeable class is designed to be wrapped with QWidget classes to allow them to be themed in a Layers application. A themeable builds a theme tag from details provided during initialization. The tag is used to retrieve attributes from themes.

Before a theme can be applied to a themeable, the themeable must have a name. This is because a name is the minimum requirement to construct the theme tag. To set a themeable's name, use set_name().

Themes are applied to themeables recursively through apply_theme(). To do this, references to child themeables need to be stored. These references are stored through store_child_themeable_pointer().

Before a themeable can be customized in a Layers application, two requirements must be fullfilled:

1. A proper name must be defined with set_proper_name()
2. Classes that implement Themeable need to define setup_customize_panel()

6.53.2 Member Function Documentation

6.53.2.1 apply_theme()

```
void Themeable::apply_theme (
            Theme & theme )  [virtual]
```

Applies the given theme to the caller and its children.

This function works recursively to apply the given theme to the caller and children in the caller's hierarchy.

A name must be set with set_name() before themes can be applied. This is because a name is the minimum requirement to construct the theme tag.

Parameters

theme	to be applied

Reimplemented in Layers::ThemesSettingsPanel, and Layers::Window.

6.53.2.2 assign_tag_prefixes()

```
void Themeable::assign_tag_prefixes (
            QList< QString > parent_prefixes = QList<QString>(),
            const QString & parent_name = "" )
```

Assigns tag prefixes from the parent and the parent's name.

This function works recursively to assign tag prefixes to the caller and children in the caller's hierarchy. Each themeable's name gets added to the tag prefix list as this function gets called down the hierarchy.

If no arguments are given, then assignment to the caller is skipped and the children have their prefixes assigned. However, the caller will still be marked as having its prefixes assigned.

6.53.2.3 attributes()

```
QMap< QString, AttributeType * > & Themeable::attributes ( )
```

Get a reference to the attribute set of the given state.

Parameters

state	of the attribute set to be returned, 'default' by default

Returns

Reference to attribute set of given state

6.53.2.4 current_theme()

```
Theme * Themeable::current_theme ( )
```

Gets the address of the currently applied theme. Returns nullptr if no theme has been applied.

Returns

Address of currently applied theme, or nullptr

6.53.2.5 customize_panel()

```
CustomizePanel * Themeable::customize_panel ( )
```

Get the address of the themeable's customize panel. Returns nullptr if the panel does not exist.

Returns

Address of customize panel, or nullptr

6.53.2.6 icon()

```
Graphic * Themeable::icon ( ) const
```

Get the address of the themeable's icon. Returns nullptr if no icon exists.

Returns

Address of icon, or nullptr

6.53.2.7 init_child_themeable_reference_list()

```
void Themeable::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented in Layers::StateAW, Layers::AWGroup, Layers::CornerRadiiAW, Layers::ColorAW, Layers::GradientAW, Layers::FillAW, Layers::NumberAW, Layers::Button, Layers::ColorDialog, Layers::ComboboxItem, Layers::Combobox, Layers::CreateNewThemeDialog, Layers::CustomizeMenu, Layers::CustomizePanel, Layers::Dialog, Layers::FillControl, Layers::GradientSelectionDialog, Layers::MenuLabelLayer, Layers::MiniSlider, Layers::ScrollArea, Layers::SettingsTab, Layers::SettingsMenu, Layers::ThemesSettingsPanel, Layers::Slider, Layers::Titlebar, Layers::ToggleSwitch, Layers::UpdateDialog, and Layers::Window.

6.53.2.8 is_stateful()

```
bool Themeable::is_stateful ( ) const
```

Initializes customize panels and adds them to the provided list.

This function works recursively to initialize all of the customize panels in the caller's hierarchy. As the panels are created, they are added to the list parameter.

Parameters

list	to store initialized customize panels

6.53.2.9 name()

```
QString * Themeable::name ( ) const
```

Calls the inheriting QWidget's update() function

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to call the inheriting QWidget's update().

6.53.2.10 proper_name()

```
QString * Layers::Themeable::proper_name ( ) const
```

Get the address of the proper name. Returns nullptr if no proper name has been set.

Returns

Address of proper name, or nullptr

6.53.2.11 reapply_theme()

```
void Themeable::reapply_theme ( )
```

Reapplies the theme that is already applied to the caller.

Does nothing if no theme has been applied yet.

6.53.2.12 remove_child_themeable_reference()

```
void Themeable::remove_child_themeable_reference (
            Themeable * child_themeable )
```

Removes the given child themeable from the references list and unassigns tag prefixes.

Parameters

child_themeable	to be removed from the reference list

6.53.2.13 set_icon()

```
void Themeable::set_icon (
            Graphic * icon )
```

Sets an icon for the themeable; replaces it if one already exists.

Parameters

icon	for the themeable

6.53.2.14 set_name()

```
void Themeable::set_name (
            const QString & name )
```

Sets the name of the themeable; replaces it if one already exists.

A themeable name is the minimum requirement to construct the themeable's theme tag needed to apply a theme.

Parameters

name	of the themeable

6.53.2.15 set_proper_name()

```
void Themeable::set_proper_name (
            const QString & proper_name )
```

Sets a proper name for the themeable; replaces it if one already exists.

A themeable's proper name is used to represent the themeable to the user. It is recommended to use capitalization when setting a proper name.

Parameters

proper_name	for the themeable

6.53.2.16 set_state()

```
void Themeable::set_state (
            const QString & state )  [virtual]
```

Sets the current state of the themeable.

This function calls issue_update() to ensure the state change is visually represented.

Parameters

state	to set the themeable to

Reimplemented in Layers::SVG.

6.53.2.17 states()

```
QList< QString > Themeable::states ( ) const
```

Get a reference to the attribute set of the given state.

Parameters

state	of the attribute set to be returned, 'default' by default

Returns

Reference to attribute set of given state

Gets a list of the states that are used to identify the caller's attribute sets.

Returns

List of states that identify attribute sets

6.53.2.18 store_child_themeable_pointer()

```
void Themeable::store_child_themeable_pointer (
            Themeable * child_themeable )
```

Adds a themeable to the child themeable references list.

If the caller has already assigned tag prefixes, then the newly added themeable reference will have its prefixes assigned during this function. Otherwise, child themeables will have their prefixes assigned when the parent calls assign_tag_prefixes().

Parameters

child_themeable	to be added to the reference list

6.53.2.19 tag()

```
QString & Themeable::tag ( )
```

Get the theme tag.

The theme tag is created structurally upon calling this function. The first part of the tag is the primary prefix which determines the owning application of the themeable. It is constructed as 'app/*application name/*'. If the themeable is built-in with Layers, and not application specific, then the primary prefix is constructed as 'layers/'.

The tag is then followed by the tag prefixes. The prefixes are made up of parent themeable names going all the way up the hierarchy from the calling themeable. They are separated by slashes in the tag.

The last part of the tag is the themeable's name.

Returns

Theme tag

6.53.2.20 unassign_prefixes()

```
void Themeable::unassign_prefixes ( )
```

Clears the tag prefix list

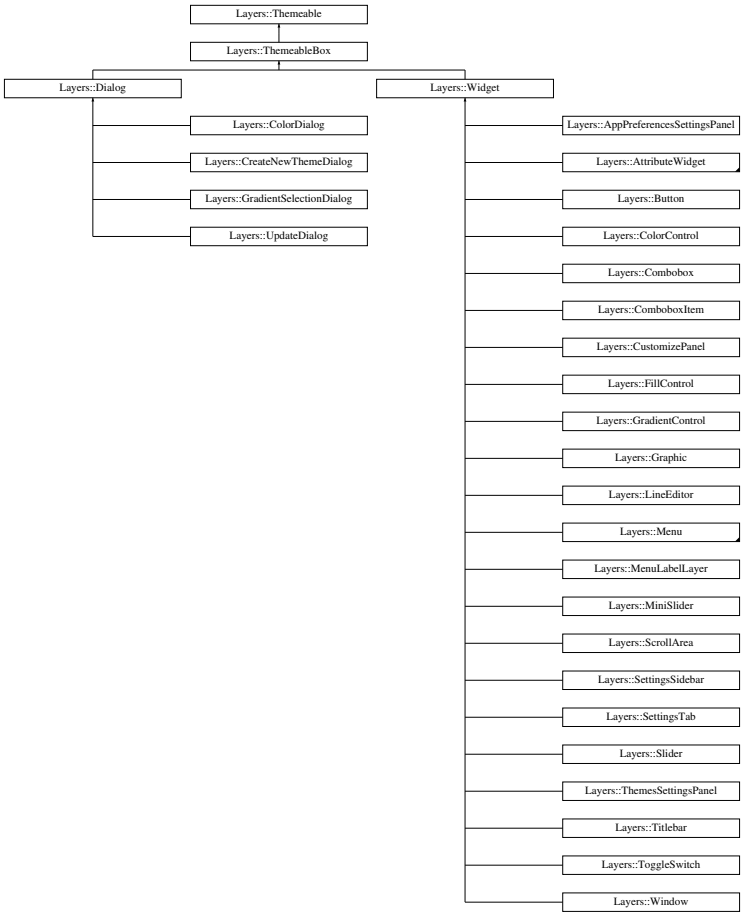
This function works recursively to clear the tag prefixes of the caller and children in the caller's hierarchy.

The documentation for this class was generated from the following files:

- include/Themeable.h
- src/Themes/Themeable.cpp

6.54 Layers::ThemeableBox Class Reference

Inheritance diagram for Layers::ThemeableBox:



Public Member Functions

- virtual void apply_theme_attributes (QMap< QString, AttributeType * > &theme_attrs) override
- void set_margin (double margin)
- void set_margin (double left, double top, double right, double bottom)

Public Attributes

- BorderAttributes **border**
- CornerRadiiAttributes **corner_radii**
- MarginsAttributes **margins**
- Attribute a_corner_color
- Attribute a_fill
- Attribute a_hover_fill
- Attribute a_outline_color

Protected Member Functions

- void init_attributes ()
- void paint (QWidget *widget)

Protected Attributes

- bool **m_hovering** { false }

6.54.1 Member Function Documentation

6.54.1.1 apply_theme_attributes()

```
void ThemeableBox::apply_theme_attributes (
            QMap< QString, AttributeType * > & theme_attrs )  [override], [virtual]
```

Reimplemented from Layers::Themeable.

6.54.1.2 init_attributes()

```
void ThemeableBox::init_attributes ( )  [protected]
```

Overrides the QWidget::eventFilter() to handle widget hover coloring

Initializes the widget's attributes.

This function uses calls to set_attribute_value() to define attributes.

Widget attributes include background color/gradient, corner radii, margins, outline color, and other varius numerical values, colors, and booleans.

6.54.1.3 paint()

```
void ThemeableBox::paint (
            QWidget * widget )  [protected]
```

Paints the widget with values obtained from the widget's attributes.

6.54.1.4 set_margin() [1/2]

```
void ThemeableBox::set_margin (
            double left,
            double top,
            double right,
            double bottom )
```

Sets the margin attributes individually.

Parameters

left	margin
top	margin
right	margin
bottom	margin

6.54.1.5 set_margin() [2/2]

```
void ThemeableBox::set_margin (
            double margin )
```

Sets all margin attributes with one value.

Parameters

margin	

6.54.2 Member Data Documentation

6.54.2.1 a_corner_color

```
Attribute Layers::ThemeableBox::a_corner_color
```

Initial value:

```
{ Attribute(
            "corner_color",
            QColor(Qt::gray),
            true
            ) }
```

6.54.2.2 a_fill

```
Attribute Layers::ThemeableBox::a_fill
```

Initial value:

```
{ Attribute(
            "fill",
            QColor(Qt::white)
            ) }
```

6.54.2.3 a_hover_fill

```
Attribute Layers::ThemeableBox::a_hover_fill
```

Initial value:

```
{ Attribute(
            "hover_fill",
            QColor(Qt::lightGray),
            true
            ) }
```

6.54.2.4 a_outline_color

```
Attribute Layers::ThemeableBox::a_outline_color
```

Initial value:

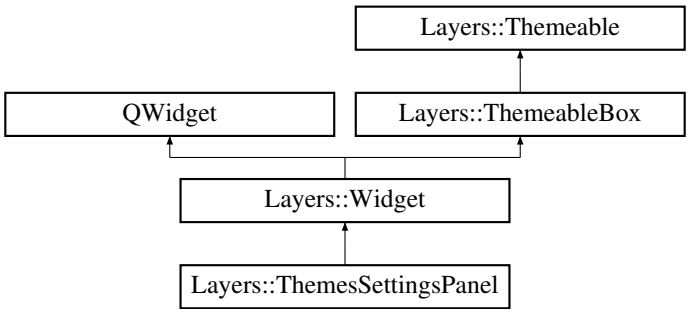
```
{ Attribute(
            "outline_color",
            QColor(Qt::gray),
            true
            ) }
```

The documentation for this class was generated from the following files:

- include/ThemeableBox.h
- src/Themes/ThemeableBox.cpp

6.55 Layers::ThemesSettingsPanel Class Reference

Inheritance diagram for Layers::ThemesSettingsPanel:



Public Member Functions

- **ThemesSettingsPanel** (QWidget *parent=nullptr)
- void apply_theme (Theme &theme)
- Button * **customize_theme_button** () const
- Button * **new_theme_button** () const
- Combobox * **theme_combobox** () const
- void **show_custom_theme_buttons** (bool cond=true)

Protected Member Functions

- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.55.1 Member Function Documentation

6.55.1.1 apply_theme()

```
void ThemesSettingsPanel::apply_theme (
            Theme & theme )  [virtual]
```

Applies the given theme to the caller and its children.

This function works recursively to apply the given theme to the caller and children in the caller's hierarchy.

A name must be set with set_name() before themes can be applied. This is because a name is the minimum requirement to construct the theme tag.

Parameters

theme	to be applied

Reimplemented from Layers::Themeable.

6.55.1.2 init_child_themeable_reference_list()

```
void ThemesSettingsPanel::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

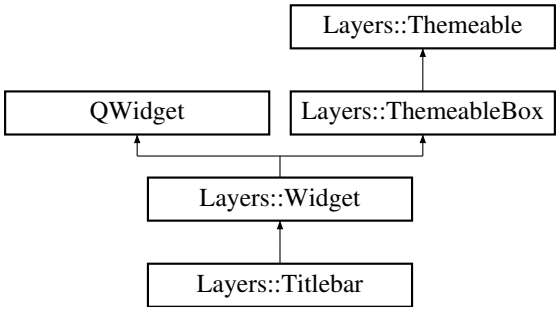
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/SettingsPanels.h
- src/Widgets/ThemesSettingsPanel.cpp

6.56 Layers::Titlebar Class Reference

Inheritance diagram for Layers::Titlebar:



Signals

- void **window_icon_updated** ()

Public Member Functions

- **Titlebar** (QWidget *parent=nullptr)
- void **add_mll** (MenuLabelLayer *mll)
- void **remove_mlls_past** (int index)
- bool **is** (QWidget *widget)
- void **set_window_icon** (const Graphic &icon_graphic)
- void **set_window_title** (const QString &title)
- Button * **window_icon** () const
- Button * **settings_button** () const
- Button * **minimize_button** () const
- Button * **maximize_button** () const
- Button * **exit_button** () const

Protected Member Functions

- void init_child_themeable_reference_list ()
- void **resizeEvent** (QResizeEvent ∗event)
- void **setup_layout** ()

Additional Inherited Members

6.56.1 Member Function Documentation

6.56.1.1 init_child_themeable_reference_list()

```
void Titlebar::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

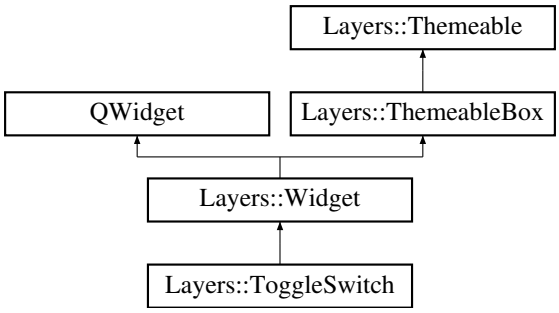
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Titlebar.h
- src/Widgets/Titlebar.cpp

6.57 Layers::ToggleSwitch Class Reference

Inheritance diagram for Layers::ToggleSwitch:



Signals

- void **toggled_event** ()

Public Member Functions

- **ToggleSwitch** (bool vertical=false, QWidget ∗parent=nullptr)
- void **setFixedHeight** (int h)
- void **set_toggled** (bool toggled)
- void **toggle** (bool emit_toggled_event=true)
- bool **toggled** () const
- void **update_layout_margins** ()
- void **update_spacer_size** ()

Public Attributes

- Attribute **a_padding_left** { Attribute("Left Padding", QVariant::fromValue(2.0)) }
- Attribute **a_padding_top** { Attribute("Top Padding", QVariant::fromValue(2.0)) }
- Attribute **a_padding_right** { Attribute("Right Padding", QVariant::fromValue(2.0)) }
- Attribute **a_padding_bottom** { Attribute("Bottom Padding", QVariant::fromValue(2.0)) }

Protected Member Functions

- bool **eventFilter** (QObject ∗object, QEvent ∗event) override
- void **init_attributes** ()
- void init_child_themeable_reference_list ()

Additional Inherited Members

6.57.1 Member Function Documentation

6.57.1.1 init_child_themeable_reference_list()

```
void ToggleSwitch::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

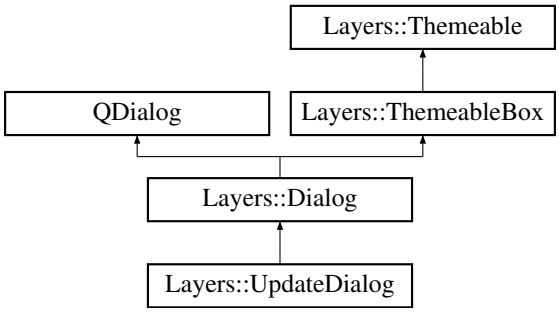
Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/ToggleSwitch.h
- src/Widgets/Controls/ToggleSwitch.cpp

6.58 Layers::UpdateDialog Class Reference

Inheritance diagram for Layers::UpdateDialog:



Public Member Functions

- **UpdateDialog** (const QString &current_version_tag, const QString &latest_version_tag, QWidget *parent=nullptr)

Protected Member Functions

- void init_child_themeable_reference_list ()

Additional Inherited Members

6.58.1 Member Function Documentation

6.58.1.1 init_child_themeable_reference_list()

```
void UpdateDialog::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Dialog.

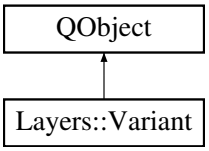
The documentation for this class was generated from the following files:

- include/UpdateDialog.h
- src/Widgets/Dialogs/UpdateDialog.cpp

6.59 Layers::Variant Class Reference

`#include <Variant.h>`

Inheritance diagram for Layers::Variant:



Signals

- void **changed** ()

Public Member Functions

- **Variant** (double d)
- **Variant** (QColor color)
- **Variant** (QVariant qvariant)
- **Variant** (const Variant &variant)
- void **operator=** (const Variant &variant)
- void **operator=** (const QVariant &qvariant)
- bool **operator!=** (const QVariant &qvariant)
- const char * typeName () const
- template<typename T >
  T value () const

6.59.1 Detailed Description

Variant type with signal/slot support

The Variant class is a QObject that wraps a QVariant to enable Qt signal/slot functionality.

The purpose of having a variant type with signal/slot support is to enable value change detection. Any time the QVariant is modified, the changed() signal gets emitted.

6.59.2 Member Function Documentation

6.59.2.1 typeName()

```
const char * Variant::typeName ( ) const
```

Returns the name of the type stored in the variant.

Returns

Type name of value stored in the variant

6.59.2.2 value()

```
template<typename T >
T Layers::Variant::value  [inline]
```

Returns the stored value converted to the template type T.

Returns

Value converted to the template type T

The documentation for this class was generated from the following files:

- include/Variant.h
- src/Themes/Attributes/Variant.cpp

6.60 Layers::Version Class Reference

Public Member Functions

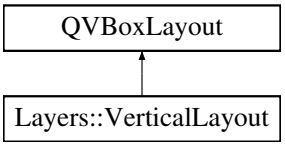
- **Version** (int major, int minor=0, int patch=0, QString phase="")
- **Version** (QString version_string)
- QString **toString** ()

The documentation for this class was generated from the following files:

- include/Version.h
- src/Tools/Version.cpp

6.61 Layers::VerticalLayout Class Reference

Inheritance diagram for Layers::VerticalLayout:



Public Member Functions

- **VerticalLayout** (QWidget *parent=nullptr)
- void **set_border_margin** (int border_margin)
- void **setContentsMargins** (int left, int top, int right, int bottom)
- void **update_margins** ()

Protected Attributes

- int **m_margin_left** { 0 }
- int **m_margin_top** { 0 }
- int **m_margin_right** { 0 }
- int **m_margin_bottom** { 0 }
- int **m_border_margin** { 0 }

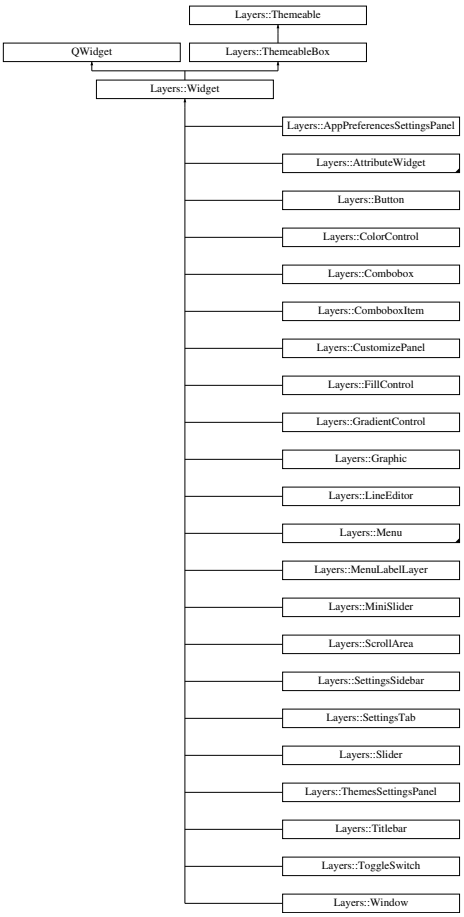
The documentation for this class was generated from the following files:

- include/Layouts.h
- src/Layouts/VerticalLayout.cpp

6.62 Layers::Widget Class Reference

`#include <Widget.h>`

Inheritance diagram for Layers::Widget:



Public Member Functions

- **Widget** (QWidget ∗parent=nullptr)

Protected Member Functions

- bool eventFilter (QObject ∗object, QEvent ∗event) override
- void init_attributes ()
- void paintEvent (QPaintEvent ∗event) override

Additional Inherited Members

6.62.1 Detailed Description

The Widget class wraps a QWidget with a Themeable to give QWidgets compatibility with the Layers theme system. The Layers Widget class overrides the QWidget's paintEvent() and uses the attributes provided by the Themeable class to handle the widget's appearance.

6.62.2 Member Function Documentation

6.62.2.1 eventFilter()

```
bool Widget::eventFilter (
            QObject * object,
            QEvent * event )  [override], [protected]
```

Overrides the QWidget::eventFilter() to handle widget hover coloring

6.62.2.2 init_attributes()

```
void Widget::init_attributes ( )  [protected]
```

Initializes the widget's attributes.

This function uses calls to set_attribute_value() to define attributes.

Widget attributes include background color/gradient, corner radii, margins, outline color, and other varius numerical values, colors, and booleans.

6.62.2.3 paintEvent()

```
void Widget::paintEvent (
            QPaintEvent * event )  [override], [protected]
```

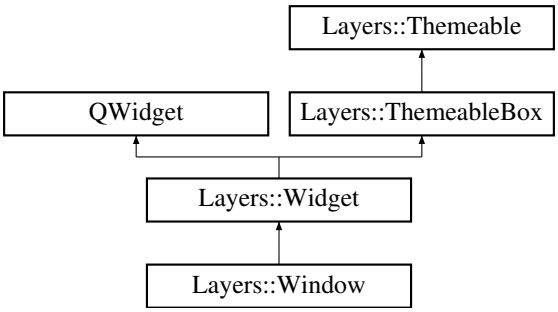
Paints the widget with values obtained from the widget's attributes.

The documentation for this class was generated from the following files:

- include/Widget.h
- src/Widgets/Widget.cpp

6.63 Layers::Window Class Reference

Inheritance diagram for Layers::Window:



Public Slots

- void **customize_clicked** ()
- void **exit_clicked** ()
- void **maximize_clicked** ()
- void **minimize_clicked** ()
- void **new_theme_clicked** ()
- void **open_menu** (Menu *menu)
- void **settings_clicked** ()

Public Member Functions

- **Window** (bool preview=false, QWidget *parent=nullptr)
- void **add_menu** (Menu *menu)
- Menu * **app_menu** () const
- void apply_theme (Theme &theme)
- void **assign_tag_prefixes** ()
- template<typename T >
  void **build_main_widget** ()
- void **center_dialog** (QDialog *dialog)
- ColorDialog * **control_color_dialog** () const
- GradientSelectionDialog * **control_gradient_selection_dialog** () const
- CustomizeMenu * **customize_menu** () const
- void **finalize** ()
- void **link_theme_name** (const QString &name)
- void **set_main_widget** (Widget *main_widget)
- void **set_window_icon** (const Graphic &icon_graphic)
- void **set_window_title** (const QString &title)
- SettingsMenu * **settings_menu** () const
- Titlebar * **titlebar** () const
- void **update_theme_dependencies** ()

Protected Member Functions

- void init_child_themeable_reference_list ()
- bool **nativeEvent** (const QByteArray &eventType, void *message, qintptr *result) override
- void **paintEvent** (QPaintEvent *event) override

Additional Inherited Members

6.63.1 Member Function Documentation

6.63.1.1 apply_theme()

```
void Window::apply_theme (
            Theme & theme )  [virtual]
```

Applies the given theme to the caller and its children.

This function works recursively to apply the given theme to the caller and children in the caller's hierarchy.

A name must be set with set_name() before themes can be applied. This is because a name is the minimum requirement to construct the theme tag.

Parameters

theme	to be applied

Reimplemented from Layers::Themeable.

6.63.1.2 init_child_themeable_reference_list()

```
void Window::init_child_themeable_reference_list ( )  [protected], [virtual]
```

Updates things that depend on the theme. Called by apply_theme().

This function can be defined by implementers of the Themeable class to handle anything they want to change when a theme gets applied that can't be changed through attributes. However, it is not required to implement this function.

Initializes the attributes.

This is a pure virtual function that must be defined by other classes that inherit the Themeable class. This function should be used to define a themeable's attributes with calls to set_attribute_value().

This function is called by init_themeable().

Initializes the customize panel.

A themeable MUST have a proper name to initialize a customize panel. Set one using set_proper_name().

This function is responsible for calling setup_customize_panel(), which is a pure virtual function that must be defined by other classes that inherit the Themeable class.

Returns

The initialized customize panel.

Initializes the reference list to child themeables.

A list of child themeable references is necessary to allow the apply_theme() function to work recursively.

Classes that inherit the Themeable class should define this function and use store_child_themeable_pointer() to populate the reference list.

This function is called by init_themeable().

Reimplemented from Layers::Themeable.

The documentation for this class was generated from the following files:

- include/Window.h
- src/Widgets/Window.cpp

Chapter 7

File Documentation

7.1 Application.h

```
1 #ifndef APPLICATION_H
2 #define APPLICATION_H
3
4 #include <QApplication>
5 #include <QDir>
6 #include <QSettings>
7 #include <QUuid>
8 #include "directories.h"
9 #include "Theme.h"
10 #include "Window.h"
11
12 namespace Layers
13 {
14     class Downloader;
15     class GitHubRepo;
16     class Version;
17
18     class Application :  public QApplication
19     {
20         Q_OBJECT
21
22     signals:
23         void current_theme_changed();
24
25     public:
26         Application(
27             int& argc, char** argv,
28             const QString& name,
29             const QUuid& uuid,
30             QFile* icon_file = nullptr,
31             Version* version = nullptr,
32             GitHubRepo* github_repo = nullptr);
33
34         QString app_identifier();
35
41         void apply_theme(Theme& theme);
42
49         void create_theme(const QString& new_theme_name, const QString& copy_theme_name);
50
56         Theme* current_theme() const;
57
65         QFile* icon_file();
66
79         Theme load_theme(const QString& file_name);
80
81         Window* main_window() const;
82
88         QString& name();
89
93         void reapply_theme();
94
104         void save_theme(Theme& theme);
105
111         QSettings& settings();
112
120         void store_child_themeable_pointer(Themeable& themeable);
121
128         Theme* theme(const QString& theme_name);
```

```
129
137         QMap<QString, Theme>& themes();
138
148         bool update_available();
149
155         bool update_on_request();
156
157     public slots:
164         void rename_theme(const QString& old_name, const QString& new_name);
165
166     private:
167         void init_directories();
168         void init_fonts();
169         void init_themes();
170         void init_latest_version_tag();
171
172         QDir m_app_dir;
173
174         QDir m_app_themes_dir;
175
176         QDir m_layers_dir{ QDir(layers_path()) };
177
178         QDir m_layers_themes_dir{ QDir(layers_themes_path()) };
179
180         QList<Themeable*> m_child_themeables;
181
182         Theme* m_current_theme{ nullptr };
183
184         Downloader* m_downloader{ nullptr };
185
186         QString m_github_api_repos_url_base{ "https://api.github.com/repos" };
187
188         GitHubRepo* m_github_repo{ nullptr };
189
190         QFile* m_icon_file{ nullptr };
191
192         QString* m_latest_version{ nullptr };
193
194         QString m_name;
195
196         QString m_name_underscored;
197
198         QSettings m_settings;
199
200         QMap<QString, Theme> m_themes;
201
202         QUuid m_uuid;
203
204         Version* m_version{ nullptr };
205     };
206 }
207
208 #define layersApp (static_cast<Layers::Application*>(qApp))
209
210 #endif // APPLICATION_H
```

7.2 Attribute.h

```
1 #ifndef ATTRIBUTE_H
2 #define ATTRIBUTE_H
3
4 #include "AttributeType.h"
5 #include "Data.h"
6
7 namespace Layers
8 {
16     class Attribute :  public AttributeType
17     {
18         Q_OBJECT
19
20     signals:
21         void ownership_changed();
22
23     public:
24         Attribute(const QString& name, bool disabled = false);
25         Attribute(const QString& name, QVariant qvariant, bool disabled = false);
26         Attribute(const QString& name, VariantMap variant_map, bool disabled = false);
27         Attribute(const Attribute& a);
28         ~Attribute();
29
39         template<typename T>
40         T as() const;
41
```

```
50          template<typename T>
51          T as(const QString& state) const;
52
56          void clear_data_if_owner();
57
64          bool contains_state(const QString& state) const;
65
71          void copy(const Attribute& attr);
72
78          void establish_data_connection();
79
92          void entangle_with(Attribute& attribute);
93
101          void init_variant_map(const VariantMap& variant_map);
102
110          virtual bool is_stateful() const override;
111
117          bool owns_data() const;
118
124          virtual void set_state(const QString& state) override;
125
135          void set_value(QVariant qvariant, bool retain_type = true);
136
146          void set_value(const QString& state, QVariant qvariant);
147
148          virtual void setup_widget_update_connection(QWidget* widget) override;
149
155          QString state() const;
156
164          QList<QString> states() const;
165
171          QJsonObject to_json_object();
172
180          const char* typeName() const;
181
182      private:
183          Data* m_data{ nullptr };
184
185          QMetaObject::Connection m_data_connection;
186
187          bool m_owns_data{ true };
188
189          QString m_state{ "" };
190      };
191
192      template<typename T>
193      inline T Attribute::as()const
194  {
195          if (m_data->is_stateful())
196              return m_data->as<T>(m_state);
197
198          return m_data->as<T>();
199      }
200
201      template<typename T>
202      inline T Attribute::as(const QString& state)const
203  {
204          return m_data->as<T>(state);
205      }
206  }
207
208  #endif // ATTRIBUTE_H
```

7.3 AttributeGroup.h

```
1  #ifndef ATTRIBUTEGROUP_H
2  #define ATTRIBUTEGROUP_H
3
4  #include "Attribute.h"
5
6  namespace Layers
7  {
16      class AttributeGroup :  public AttributeType
17      {
18          Q_OBJECT
19
20      public:
21          AttributeGroup();
22          AttributeGroup(const QString& name, const QMap<QString, Attribute*>& attributes, bool disabled =
        false);
23          AttributeGroup(const AttributeGroup& ag);
24
30          QMap<QString, Attribute*>& attributes();
```

```
31
32          QMap<QString, Attribute*>::iterator begin();
33
39          void copy(const AttributeGroup& ag);
40
41          QMap<QString, Attribute*>::iterator end();
42
49          void entangle_with(AttributeGroup& attr_group);
50
59          virtual bool is_stateful() const override;
60
66          virtual void set_state(const QString& state) override;
67
68          virtual void setup_widget_update_connection(QWidget* widget) override;
69
75          QJsonObject to_json_object();
76
77      private:
78          QMap<QString, Attribute*> m_attributes;
79      };
80
81      class BorderAttributes :  public AttributeGroup
82      {
83          Q_OBJECT
84
85      public:
86          BorderAttributes(const QString& name = "border");
87
88          Attribute fill{ Attribute(
89              "fill",
90              QColor(Qt::gray)
91              ) };
92
93          Attribute thickness{ Attribute(
94              "thickness",
95              QVariant::fromValue(0.0)
96              ) };
97      };
98
99      class CornerRadiiAttributes :  public AttributeGroup
100      {
101          Q_OBJECT
102
103      public:
104          CornerRadiiAttributes(const QString& name = "corner_radii");
105
106          Attribute bottom_left{ Attribute(
107              "bottom_left",
108              QVariant::fromValue(0.0)
109              ) };
110
111          Attribute bottom_right{ Attribute(
112              "bottom_right",
113              QVariant::fromValue(0.0)
114              ) };
115
116          Attribute top_left{ Attribute(
117              "top_left",
118              QVariant::fromValue(0.0)
119              ) };
120
121          Attribute top_right{ Attribute(
122              "top_right",
123              QVariant::fromValue(0.0)
124              ) };
125      };
126
127      class MarginsAttributes :  public AttributeGroup
128      {
129          Q_OBJECT
130
131      public:
132          MarginsAttributes(const QString& name = "margins");
133
134          Attribute left{ Attribute(
135              "left",
136              QVariant::fromValue(0.0)
137              ) };
138
139          Attribute top{ Attribute(
140              "top",
141              QVariant::fromValue(0.0)
142              ) };
143
144          Attribute right{ Attribute(
145              "right",
146              QVariant::fromValue(0.0)
```

```
147             ) };
148
149         Attribute bottom{ Attribute(
150             "bottom",
151             QVariant::fromValue(0.0)
152             ) };
153     };
154 }
155
156 #endif // ATTRIBUTEGROUP_H
```

7.4 AttributeLayout.h

```
1 //#ifndef ATTRIBUTELAYOUT_H
2 //#define ATTRIBUTELAYOUT_H
3 //
4 //#include "Attribute.h"
5 //
6 //namespace Layers
7 //{
8 //  class AttributeType
9 //  {
10 //   public:
11 //
12 //
13 //  private:
14 //      QList<>
15 //
16 //  };
17 //}
18 //
19 //#endif // ATTRIBUTELAYOUT_H
```

7.5 AttributeSet.h

```
1 #ifndef ATTRIBUTESET_H
2 #define ATTRIBUTESET_H
3
4 #include <QDataStream>
5
6 #include "Attribute.h"
7
8 namespace Layers
9 {
10     class AttributeSet
11     {
12     public:
13         void add_attribute(Attribute* attribute);
14
15         Attribute* attribute(const QString& attribute_name);
16
17         QVariant* attribute_value(const QString& attribute_name);
18
19         QMap<QString, Attribute*>& attributes();
20
21         bool contains(const QString& attribute_name);
22
23         void copy_values_from(AttributeSet& other_attribute_set);
24
25         void remove_attribute(const QString& attribute_name);
26
27         bool replace_with_proxy(const QString& attribute_name, Attribute* proxy_attribute);
28
29         void replace_all_with(AttributeSet& other_attribute_set);
30
31         void set_state(const QString& state);
32
33         QList<QString> states() const;
34
35         friend QDataStream& operator <<(QDataStream& stream, const AttributeSet& as)
36         {
37             stream << as.m_attributes.count();
38
39             for (Attribute* attr :  as.m_attributes)
40                 stream << *attr;
41
42             return stream;
43         }
44
```

```
45          friend QDataStream& operator »(QDataStream& stream, AttributeSet& as)
46          {
47              qsizetype attr_count;
48
49              stream » attr_count;
50
51              for (int i = 0; i < attr_count; i++)
52              {
53                  Attribute* attr = new Attribute("");
54
55                  stream » *attr;
56
57                  as.m_attributes.insert(attr->name(), attr);
58              }
59
60              return stream;
61          }
62
63      private:
64          QMap<QString, Attribute*> m_attributes{ QMap<QString, Attribute*>() };
65      };
66  }
67
68  #endif // ATTRIBUTESET_H
```

7.6 AttributeType.h

```
1  #ifndef ATTRIBUTETYPE_H
2  #define ATTRIBUTETYPE_H
3
4  #include <QObject>
5  #include <QWidget>
6
7  namespace Layers
8  {
13      class AttributeType :  public QObject
14      {
15          Q_OBJECT
16
17      signals:
18          void value_changed();
19
20      public:
21          AttributeType(const QString& name, bool disabled);
22
28          QString capitalized_name();
29
35          bool disabled() const;
36
47          virtual bool is_stateful() const = 0;
48
54          QString name();
55
61          virtual void set_disabled(bool disabled = true);
62
74          virtual void set_state(const QString& state) = 0;
75
76          virtual void setup_widget_update_connection(QWidget* widget) = 0;
77
78      protected:
79          bool m_disabled{ false };
80
81          QString m_name{ "" };
82      };
83  }
84
85  #endif // ATTRIBUTETYPE_H
```

7.7 AttributeWidgets.h

```
1  #ifndef ATTRIBUTEWIDGET_H
2  #define ATTRIBUTEWIDGET_H
3
4  #include "Button.h"
5  #include "ColorControl.h"
6  #include "Combobox.h"
7  #include "GradientControl.h"
8  #include "FillControl.h"
9  #include "Graphic.h"
```

```
10 #include "Label.h"
11 #include "LineEditor.h"
12 #include "MiniSlider.h"
13 #include "ToggleSwitch.h"
14
15 namespace Layers
16 {
17     class AttributeWidget :  public Widget
18     {
19         Q_OBJECT
20
21     signals:
22         void widget_disabled();
23
24     public:
25         AttributeWidget(AttributeType* attr_type = nullptr, QWidget* parent = nullptr);
26
27         ToggleSwitch* disable_toggle() const;
28
29         bool disabled() const;
30
31         Widget* toggle_label_separator() const;
32
33     public slots:
34         virtual void set_current_editting_state(const QString& state);
35
36     protected:
37         void init_attributes();
38
39         AttributeType* m_attribute_type;
40
41         ToggleSwitch* m_disabled_toggle{ new ToggleSwitch };
42
43         Widget* m_toggle_label_separator{ new Widget };
44     };
45
46     class StateAW :  public AttributeWidget
47     {
48         Q_OBJECT
49
50     public:
51         StateAW(QWidget* parent = nullptr);
52
53         void add_attribute_widget(AttributeWidget* attribute_widget);
54
55         Combobox* state_combobox() const;
56
57         void populate_state_combobox(const QList<QString>& states);
58
59     protected:
60         void init_child_themeable_reference_list();
61
62     private:
63         void setup_layout();
64
65         Combobox* m_state_combobox{ new Combobox };
66
67         Label* m_label{ new Label("State")};
68
69         QList<AttributeWidget*> m_child_attribute_widgets{ QList<AttributeWidget*>() };
70
71         QVBoxLayout* m_main_layout{ new QVBoxLayout };
72
73         QVBoxLayout* m_widgets_vbox{ new QVBoxLayout };
74     };
75
76     class AWGroup :  public AttributeWidget
77     {
78         Q_OBJECT
79
80     public:
81         AWGroup(AttributeGroup* attr_group, QWidget* parent = nullptr);
82
83         void add_attribute_widget(AttributeWidget* attribute_widget);
84
85         void set_collapsed(bool collapsed = true);
86
87         //virtual bool disabled() const;
88
89     public slots:
90         virtual void set_current_editting_state(const QString& state) override;
91
92     protected:
93         void init_child_themeable_reference_list();
94
95     private:
96         void setup_layout();
```

```
97
98          Button* m_collapse_button{ new Button(new Graphic(":/svgs/collapse_arrow_right.svg", QSize(8,
    12)), new Graphic(":/svgs/collapse_arrow_down.svg", QSize(12, 8)), true) };
99
100         bool m_collapsed{ true };
101
102         Label* m_label{ nullptr };
103
104         QList<AttributeWidget*> m_child_attribute_widgets{ QList<AttributeWidget*>() };
105
106         QVBoxLayout* m_widgets_vbox{ new QVBoxLayout };
107     };
108
109     class CornerRadiiAW :  public AWGroup
110     {
111         Q_OBJECT
112
113     public:
114         CornerRadiiAW(CornerRadiiAttributes* linked_corner_radii, QWidget* parent = nullptr);
115
116         MiniSlider* tl_slider() const;
117         MiniSlider* tr_slider() const;
118         MiniSlider* bl_slider() const;
119         MiniSlider* br_slider() const;
120
121     public slots:
122         void set_current_editting_state(const QString& state);
123
124     protected:
125         void init_child_themeable_reference_list();
126
127     private:
128         void setup_layout();
129
130         AttributeWidget* m_attribute_widget;
131
132         MiniSlider* m_tl_slider{ new MiniSlider(30.0) };
133         MiniSlider* m_tr_slider{ new MiniSlider(30.0) };
134         MiniSlider* m_bl_slider{ new MiniSlider(30.0) };
135         MiniSlider* m_br_slider{ new MiniSlider(30.0) };
136
137         LineEditor* m_tl_line_editor{ new LineEditor };
138         LineEditor* m_tr_line_editor{ new LineEditor };
139         LineEditor* m_bl_line_editor{ new LineEditor };
140         LineEditor* m_br_line_editor{ new LineEditor };
141
142         QVBoxLayout* m_main_layout{ new QVBoxLayout };
143
144         Widget* m_example_widget{ new Widget };
145     };
146
147     class ColorAW :  public AttributeWidget
148     {
149         Q_OBJECT
150
151     public:
152         ColorAW(Attribute* attribute, QWidget* parent = nullptr);
153
154         ColorControl* color_control() const;
155
156         void set_centered(bool centered = true);
157
158     public slots:
159         void set_current_editting_state(const QString& state);
160
161     protected:
162         void init_child_themeable_reference_list();
163
164     private:
165         bool m_centered{ false };
166
167         ColorControl* m_color_control{ new ColorControl };
168
169         Label* m_attribute_label;
170
171         Widget* m_left_stretch{ new Widget };
172         Widget* m_right_stretch{ new Widget };
173     };
174
175     class GradientAW :  public AttributeWidget
176     {
177         Q_OBJECT
178
179     public:
180         GradientAW(const QString& attribute_label_text, Attribute* attribute, QWidget* parent =
    nullptr);
181
```

```
182         void set_centered(bool centered = true);
183
184     protected:
185         void init_child_themeable_reference_list();
186
187     private:
188         bool m_centered{ false };
189
190         GradientControl* m_gradient_control{ new GradientControl };
191
192         Label* m_attribute_label;
193
194         Widget* m_left_stretch{ new Widget };
195         Widget* m_right_stretch{ new Widget };
196     };
197
198     class FillAW :  public AttributeWidget
199     {
200         Q_OBJECT
201
202     public:
203         FillAW(Attribute* attribute, QWidget* parent = nullptr);
204
205         FillControl* fill_control() const;
206
207         //void set_centered(bool centered = true);
208
209     public slots:
210         virtual void set_current_editting_state(const QString& state) override;
211
212     protected:
213         void init_child_themeable_reference_list();
214
215     private:
216         bool m_centered{ false };
217
218         FillControl* m_fill_control{ new FillControl };
219
220         Label* m_attribute_label;
221
222         Widget* m_left_stretch{ new Widget };
223         Widget* m_right_stretch{ new Widget };
224     };
225
226     class NumberAW :  public AttributeWidget
227     {
228         Q_OBJECT
229
230     public:
231         NumberAW(Attribute* attribute, QIntValidator* int_validator, QWidget* parent = nullptr);
232
233         void set_centered(bool centered = true);
234
235         void set_unit_label_text(const QString& unit_string);
236
237     protected:
238         void init_child_themeable_reference_list();
239
240     private:
241         void setup_layout();
242
243         bool m_centered{ false };
244
245         Label* m_attribute_label{ nullptr };
246         Label* m_unit_label{ new Label };
247
248         LineEditor* m_line_editor{ new LineEditor };
249
250         QVBoxLayout* m_main_layout{ new QVBoxLayout };
251
252         MiniSlider* m_slider{ nullptr };
253
254         Widget* m_left_stretch{ new Widget };
255         Widget* m_right_stretch{ new Widget };
256     };
257 }
258
259 #endif // ATTRIBUTEWIDGET_H
```

7.8 build_themes.h

```
1 #ifndef BUILD_THEMES_H
2 #define BUILD_THEMES_H
```

```
3
4 namespace Layers
5 {
6     class Theme;
7
8     Theme build_layers_blue_theme();
9     Theme build_layers_dark_theme();
10     Theme build_layers_light_theme();
11 }
12
13 #endif // BUILD_THEMES_H
```

7.9 Button.h

```
1 #ifndef BUTTON_H
2 #define BUTTON_H
3
4 #include <QGraphicsOpacityEffect>
5 #include <QHBoxLayout>
6
7 #include "Label.h"
8 #include "Widget.h"
9
10 namespace Layers
11 {
12     class Button :  public Widget
13     {
14         Q_OBJECT
15
16     signals:
17         void clicked();
18
19     public:
20         Button(Graphic* graphic, const QString& text, bool auto_touch_target_compliance = false, QWidget*
    parent = nullptr);
21         Button(Graphic* graphic, bool auto_touch_target_compliance = false, QWidget* parent = nullptr);
22         Button(const QString& text, bool auto_touch_target_compliance = false, QWidget* parent =
    nullptr);
23         Button(Graphic* graphic_before, Graphic* graphic_after, bool auto_touch_target_compliance =
    false, QWidget* parent = nullptr);
24         ~Button();
25
26         void disable_graphic_hover_color(bool cond = true);
27         void disable_text_hover_color(bool cond = true);
28
29         bool disabled() const;
30
31         Graphic* graphic() const;
32
33         void resize();
34
35         void set_available_width(int available_width);
36         void set_disabled(bool cond = true);
37         void set_font_size(int size);
38         void set_padding(int padding);
39         void set_padding(int left, int top, int right, int bottom);
40         void set_text_padding(int left, int top, int right, int bottom);
41
42         void toggle_graphics();
43
44         int left_padding() const;
45         int top_padding() const;
46         int right_padding() const;
47         int bottom_padding() const;
48
49     protected:
50         bool eventFilter(QObject* object, QEvent* event) override;
51
52         void init();
53         void init_child_themeable_reference_list();
54
55         void setup_layout();
56
57     private:
58         QHBoxLayout* main_layout{ new QHBoxLayout };
59
60         QGraphicsOpacityEffect* m_button_opacity{ new QGraphicsOpacityEffect };
61
62         bool m_auto_touch_target_compliance{ false };
63
64         bool m_disabled{ false };
65
66         bool m_use_graphic_hover_color{ true };
```

```
67          bool m_use_text_hover_color{ true };
68
69          int m_available_width{ 16777215 };
70
71          Graphic* m_graphic{ nullptr };
72          Graphic* m_graphic_after{ nullptr };
73
74          Label* m_text_label{ nullptr };
75      };
76  }
77
78  #endif // BUTTON_H
```

7.10 calculate.h

```
1  #ifndef MATH_H
2  #define MATH_H
3
4  #include <cmath>
5
6  namespace Layers
7  {
8      inline double round(double d)
9      {
10         return floor(d + 0.5);
11     }
12
13     inline bool is_even(int i)
14     {
15         if (i % 2) return false;
16
17         return true;
18     }
19
20     inline double inner_radius(int outer_radius, int thickness)
21     {
22         double border_lower_bound = 3;
23         double border_range = 27;
24         double border_percent = (thickness - border_lower_bound) / border_range;
25
26         double y_int_lower_bound = -2.2164;
27         double y_int_range = 2.5446;
28
29         double slope_lower_bound = 0.3435;
30         double slope_range = 0.6205;
31
32         double y_int = (border_percent * y_int_range) + y_int_lower_bound;
33
34         double slope = ((1 - border_percent) * slope_range) + slope_lower_bound;
35
36         double value = round(slope * double(outer_radius) + y_int);
37
38         if (value < 0) value = 0;
39
40         return value;
41     }
42 }
43
44 #endif // MATH_H
```

7.11 ColorControl.h

```
1  #ifndef COLORCONTROL_H
2  #define COLORCONTROL_H
3
4  #include "Widget.h"
5
6  namespace Layers
7  {
8      class ColorControl :  public Widget
9      {
10         Q_OBJECT
11
12     signals:
13         void color_changed();
14
15     public:
16         ColorControl(QWidget* parent = nullptr);
17         ~ColorControl();
```

```
18
19          void click();
20
21          void disable_clicking(bool cond = true);
22
23          //void set_attribute(Attribute* attribute);
24
25          //Attribute a_corner_radii{ Attribute("Corner Radii", QVariant::fromValue(5.0)) };
26          //Attribute a_inner_border_color{ Attribute("Inner Border Color", QColor("#2c2c2c")) };
27          //Attribute a_outer_border_color{ Attribute("Outer Border Color", QColor("#d6d6d6")) };
28
29      public slots:
30          void set_current_editting_state(const QString& state);
31
32      protected:
33          bool eventFilter(QObject* object, QEvent* event);
34
35          void init_attributes();
36
37      private:
38          bool clicking_disabled{ false };
39          bool open_on_release{ false };
40
41          QMetaObject::Connection attribute_connection;
42
43          QList<QString> m_attribute_states{ QList<QString>() };
44      };
45  }
46
47  #endif // COLORCONTROL_H
```

7.12 ColorDialog.h

```
1  #ifndef COLORDIALOG_H
2  #define COLORDIALOG_H
3
4  #include "ColorPlane.h"
5  #include "Dialog.h"
6  #include "LineEditor.h"
7  #include "Slider.h"
8
9  namespace Layers
10 {
11      class ColorDialog :  public Dialog
12      {
13          Q_OBJECT
14
15      public:
16          ColorDialog(QWidget* parent = nullptr);
17
18          Attribute color{ Attribute("color", QColor()) };
19
20          void update_color_name_line_editor();
21
22      protected:
23          void init_attributes();
24          void init_child_themeable_reference_list();
25
26      private:
27          void setup_layout();
28
29          Button* m_apply_button{ new Button("Apply") };
30
31          LineEditor* m_color_name_line_editor{ new LineEditor };
32
33          ColorPlane* m_color_plane{ new ColorPlane };
34
35          Slider* m_z_slider{ new Slider(359) };
36      };
37  }
38
39  #endif // COLORDIALOG_H
```

7.13 ColorPlane.h

```
1  #ifndef COLORPLANE_H
2  #define COLORPLANE_H
3
4  #include <QWidget>
```

```
5
6 #include "Attribute.h"
7 #include "calculate.h"
8 #include "Widget.h"
9
10 namespace Layers
11 {
12     class ColorPlane :  public QWidget
13     {
14         Q_OBJECT
15
16     signals:
17         void active_mode_changed();
18
19     public:
20         enum class Mode { Hue, Saturation, Value };
21
22         ColorPlane(QWidget* parent = nullptr);
23
24         Mode active_mode() const;
25
26         float pos_as_ratio(int pos, int available_space);
27
28         void set_active_mode(Mode new_active_hsv);
29
30         void setFixedHeight(int h);
31         void setFixedSize(const QSize& s);
32         void setFixedSize(int w, int h);
33         void setFixedWidth(int w);
34
35         void update_color(float x_pos_ratio, float y_pos_ratio);
36
37         void update_height_dependencies();
38         void update_width_dependencies();
39
40         Attribute color{ Attribute("color", QColor("#ff0000"))};
41
42         Attribute z_value{ Attribute("z_value", QVariant::fromValue(0.0)) };
43
44     public slots:
45         void update_cursor_position();
46         void update_z_value();
47
48     protected:
49         bool eventFilter(QObject* object, QEvent* event) override;
50
51         void paintEvent(QPaintEvent* event) override;
52
53     private:
54         void handle_mouse_event(QPoint& mouse_pos);
55
56         void init_attributes();
57
58         Mode m_active_mode{ Mode::Hue };
59
60         Widget* m_cursor{ new Widget(this) };
61
62         int m_draw_height{ 245 };
63         int m_draw_width{ 245 };
64
65         const int margin{ 5 };
66         const int max_H{ 359 };
67         const int max_SV{ 255 };
68         const int max_RGB{ 255 };
69
70         bool m_dragging{ false };
71     };
72 }
73
74 #endif // COLORPLANE_H
```

7.14 Combobox.h

```
1 #ifndef COMBOBOX_H
2 #define COMBOBOX_H
3
4 #include <QLineEdit>
5 #include <QVBoxLayout>
6
7 #include "Label.h"
8 #include "Widget.h"
9
10 namespace Layers
```

```
11 {
12     class ComboboxItem :  public Widget
13     {
14         Q_OBJECT
15 
16     public:
17         ComboboxItem(const QString& item_text, QWidget* parent = nullptr);
18 
19         QString item_text();
20 
21         void replace_item_text(const QString& new_item_text);
22 
23         void set_font_size(int size);
24         void setFixedSize(const QSize& s);
25         void setFixedSize(int w, int h);
26 
27     protected:
28         void init_attributes();
29         void init_child_themeable_reference_list();
30 
31     private:
32         Label* m_item_label;
33 
34         QString m_item_text;
35     };
36 
37     class Combobox :  public Widget
38     {
39         Q_OBJECT
40 
41     signals:
42         void current_item_changed(const QString& current_item);
43         void item_replaced(const QString& old_item, const QString& new_item);
44 
45     public:
46         Combobox(QWidget* parent = nullptr);
47 
48         void add_item(const QString& item);
49 
50         void alphabetize();
51 
52         void edit_current_item();
53 
54         void enable_alphabetization(bool cond = true);
55 
56         void set_current_item(const QString& item);
57         void set_disabled(bool cond = true);
58         void set_font_size(int size);
59         void set_item_renaming_disabled(bool disable = true);
60         void set_padding(int left, int top, int right, int bottom);
61         void setFixedSize(const QSize& s);
62         void setFixedSize(int w, int h);
63 
64         QString current_item() const;
65 
66         QList<QString> items();
67 
68         void update_theme_dependencies();
69 
70         Attribute a_line_edit_text_color{ Attribute("line_edit_text_color", QColor(Qt::black)) };
71 
72     public slots:
73         void line_edit_return_pressed();
74 
75     protected:
76         virtual bool eventFilter(QObject* object, QEvent* event) override;
77 
78         void init_attributes();
79         void init_child_themeable_reference_list();
80 
81     private:
82         void setup_layout();
83 
84         bool m_alphabetize{ false };
85         bool m_disabled{ false };
86         bool m_item_renaming_disabled{ true };
87 
88         ComboboxItem* m_control_combobox_item{ new ComboboxItem("") };
89         ComboboxItem* m_current_combobox_item{ nullptr };
90 
91         Label* m_current_item_label{ new Label(this) };
92 
93         QLineEdit* m_line_edit{ new QLineEdit(this) };
94 
95         QList<ComboboxItem*> m_combobox_items;
96 
97         QVBoxLayout* m_drop_down_layout{ new QVBoxLayout };
```

```
98
99          Widget* m_drop_down{ new Widget };
100     };
101 }
102
103 #endif // COMBOBOX_H
```

7.15 CreateNewThemeDialog.h

```
1 #ifndef CREATENEWTHEMEDIALOG_H
2 #define CREATENEWTHEMEDIALOG_H
3
4 #include "Combobox.h"
5 #include "Dialog.h"
6 #include "LineEditor.h"
7
8 namespace Layers
9 {
10      class CreateNewThemeDialog :  public Dialog
11      {
12          Q_OBJECT
13
14      public:
15          CreateNewThemeDialog(QWidget* parent = nullptr);
16
17          void add_theme_name_to_combobox(const QString& theme_name);
18
19          void clear();
20
21          QString copy_theme_name();
22
23          QString new_theme_name();
24
25          void set_current_start_theme_name(const QString& theme_name);
26
27      protected:
28          void init_attributes();
29          void init_child_themeable_reference_list();
30
31      private:
32          void setup_layout();
33
34          Button* m_create_button{ new Button("Create") };
35
36          Combobox* m_start_theme_combobox{ new Combobox };
37
38          Label* m_name_label{ new Label("Name") };
39          Label* m_start_as_label{ new Label("Start as copy of") };
40
41          LineEditor* m_theme_name_line_edit{ new LineEditor };
42      };
43 }
44
45 #endif // CREATENEWTHEMEDIALOG_H
```

7.16 CustomizeMenu.h

```
1 #ifndef CUSTOMIZEMENU_H
2 #define CUSTOMIZEMENU_H
3
4 #include <QIntValidator>
5
6 #include "AttributeWidgets.h"
7 #include "Button.h"
8 #include "Graphic.h"
9 #include "Layouts.h"
10 #include "Menu.h"
11 #include "ScrollArea.h"
12
13 namespace Layers
14 {
15      class CustomizeMenu :  public Menu
16      {
17          Q_OBJECT
18
19      public:
20          CustomizeMenu(QWidget* parent = nullptr);
21
22          Button* apply_button() const;
```

```
23
24          void init_preview_window();
25
26          void open_customize_panel(CustomizePanel* customize_panel);
27
28          QList<CustomizePanel*>& panels();
29
30          Widget* preview_widget() const;
31
32          int calculated_topbar_content_width();
33
34          void set_preview_widget(Widget* widget);
35
36          int topbar_content_width(bool include_collapse_button);
37
38      protected:
39          bool eventFilter(QObject* object, QEvent* event) override;
40
41          void init_child_themeable_reference_list();
42
43      private:
44          void adjust_collapsed_widget();
45          void collapse_text_buttons();
46          void expand_text_buttons();
47          void setup_layout();
48
49          AWGroup* m_control_aw_group{ new AWGroup(new AttributeGroup) };
50          ColorAW* m_control_color_aw{ new ColorAW(new Attribute("", QColor())) };
51          CornerRadiiAW* m_control_corner_radii_aw{ new CornerRadiiAW(new CornerRadiiAttributes) };
52          FillAW* m_control_fill_aw{ new FillAW(new Attribute("", QColor())) };
53          NumberAW* m_control_number_aw{ new NumberAW(new Attribute("", QVariant::fromValue(0.0)), new
    QIntValidator) };
54          StateAW* m_control_state_aw{ new StateAW };
55          Button* m_control_widget_button{ new Button(new Graphic(":/svgs/settings_animated.svg", QSize(24,
    24)), QString("")) };
56
57          HorizontalLayout* m_main_layout{ new HorizontalLayout };
58          QVBoxLayout* m_collapsed_text_buttons_layout{ new QVBoxLayout };
59          QVBoxLayout* m_sidebar_layout{ new QVBoxLayout };
60          QHBoxLayout* m_topbar_layout{ new QHBoxLayout };
61          QGridLayout* m_preview_layout{ new QGridLayout };
62
63          Widget* m_sidebar_widget{ new Widget };
64          Widget* m_topbar{ new Widget };
65          Widget* m_preview_frame = new Widget;
66
67          ScrollArea* m_sidebar{ new ScrollArea };
68          ScrollArea* m_preview_scroll_area{ new ScrollArea };
69
70          Widget* m_preview_widget{ nullptr };
71
72          QList<CustomizePanel*> m_panel_stack;
73          QList<Button*> m_text_button_stack;
74          QList<Button*> m_topbar_text_buttons;
75          QList<Button*> m_collapsed_text_buttons;
76          QList<Graphic*> m_arrow_graphics;
77
78          QSize* m_previous_size{ nullptr };
79
80          Button* m_apply_button{ new Button("Apply", true) };
81          Button* m_collapse_menu_button{ new Button(new Graphic(":/svgs/ellipsis.svg", QSize(32, 8)),
    true) };
82
83          Graphic* m_control_arrow_graphic{ new Graphic(":/svgs/collapse_arrow_right.svg", QSize(8, 12)) };
84
85          CustomizePanel* m_control_customize_panel{ nullptr };
86
87          Button* m_control_text_button{ new Button("") };
88
89          Widget* m_collapse_menu{ new Widget };
90      };
91  }
92
93  #endif // CUSTOMIZEMENU_H
```

7.17 CustomizePanel.h

```
1  #ifndef CUSTOMIZEPANEL_H
2  #define CUSTOMIZEPANEL_H
3
4  #include "AttributeWidgets.h"
5  #include "Button.h"
6  #include "Label.h"
```

```
7
8 namespace Layers
9 {
10     class CustomizePanel :  public Widget
11     {
12         Q_OBJECT
13
14     public:
15         CustomizePanel(Themeable* themeable, QWidget* parent = nullptr);
16         ~CustomizePanel();
17
18         void add_attribute_widget(AttributeWidget* attribute_widget);
19
20         void add_widget_button(Button* button, int index = -1);
21
22         void init_attribute_widgets();
23
24         void replace_all_aw_group_attrs_with(AWGroup* control_aw_group);
25         void replace_all_color_awidgets_attrs_with(ColorAW* control_color_aw);
26         void replace_all_fill_awidgets_attrs_with(FillAW* control_fill_aw);
27         void replace_all_number_awidgets_attrs_with(NumberAW* control_number_aw);
28         void replace_all_state_awidgets_attrs_with(StateAW* control_state_aw);
29         void replace_all_widget_buttons_attrs_with(Button* control_widget_button);
30         void replace_all_corner_radii_aw_attrs_with(CornerRadiiAW* control_corner_radii_aw);
31
32     protected:
33         void init_attributes();
34         void init_child_themeable_reference_list();
35
36     private:
37         void setup_layout();
38
39         bool m_layout_setup{ false };
40         bool m_showing_primary{ true };
41
42         StateAW* m_state_aw{ nullptr };
43
44         QVBoxLayout* m_attributes_layout{ new QVBoxLayout };
45         QVBoxLayout* m_widgets_layout{ new QVBoxLayout };
46         QVBoxLayout* m_widget_buttons_layout{ new QVBoxLayout };
47
48         Button* m_show_all_button{ new Button("Show All", true) };
49         Button* m_show_primary_button{ new Button("Show Primary", true) };
50
51         Label* m_attributes_label{ new Label("Attributes:") };
52         Label* m_widgets_label{ new Label("Widgets:") };
53
54         QList<AttributeWidget*> m_attribute_widgets{ QList<AttributeWidget*>() };
55
56         QList<AWGroup*> m_aw_groups{ QList<AWGroup*>() };
57         QList<ColorAW*> m_color_awidgets{ QList<ColorAW*>() };
58         QList<CornerRadiiAW*> m_corner_radii_awidgets{ QList<CornerRadiiAW*>() };
59         QList<FillAW*> m_fill_awidgets{ QList<FillAW*>() };
60         QList<NumberAW*> m_number_awidgets{ QList<NumberAW*>() };
61         QList<StateAW*> m_state_awidgets{ QList<StateAW*>() };
62         QList<Button*> m_widget_buttons{ QList<Button*>() };
63
64         Themeable* m_themeable;
65     };
66 }
67
68 #endif // CUSTOMIZEPANEL_H
```

7.18 Data.h

```
1 #ifndef DATA_H
2 #define DATA_H
3
4 #include <QJsonObject>
5
6 #include "Variant.h"
7
8 namespace Layers
9 {
21     class Data :  public QObject
22     {
23         Q_OBJECT
24
25     signals:
26         void changed();
27
28     public:
29         Data(QVariant qvariant);
```

```
30          Data(VariantMap variant_map);
31          Data(const Data& d);
32          ~Data();
33
44          template<typename T>
45          T as(const QString& state = "") const;
46
53          bool contains_state(const QString& state) const;
54
62          void copy(const Data& data);
63
69          void init_variant_map(const VariantMap& variant_map);
70
76          bool is_stateful() const;
77
86          void set_value(QVariant qvariant, bool retain_type = true);
87
96          void set_value(const QString& state, QVariant qvariant);
97
105          QList<QString> states() const;
106
112          QJsonObject to_json_object();
113
123          const char* typeName() const;
124
125      private:
126          Variant* m_variant{ nullptr };
127
128          VariantMap* m_variant_map{ nullptr };
129      };
130
131      template<typename T>
132      inline T Data::as(const QString& state)const
133  {
134          if (state == "")
135              return m_variant->value<T>();
136          else
137              return (*m_variant_map)[state].value<T>();
138      }
139  }
140
141  #endif // DATA_H
```

7.19 Dialog.h

```
1  #ifndef DIALOG_H
2  #define DIALOG_H
3
4  #include <QDialog>
5
6  #include "Button.h"
7  #include "Graphic.h"
8
9  namespace Layers
10 {
11     class Dialog :  public QDialog, public ThemeableBox
12     {
13         Q_OBJECT
14
15     public:
16         Dialog(const QString& title = "Dialog", QWidget* parent = nullptr);
17
18         void setLayout(QLayout* layout);
19
20     public slots:
21         void update_content_margins();
22         void update_titlebar();
23
24     protected:
25         void init_attributes();
26         void init_child_themeable_reference_list();
27
28         bool nativeEvent(const QByteArray& eventType, void* message, qintptr* result) override;
29
30         void paintEvent(QPaintEvent* event) override;
31
32         QVBoxLayout* m_main_layout{ new QVBoxLayout };
33
34     private:
35         void init_titlebar();
36
37         void setup_layout();
38
```

```
39          bool m_hovering{ false };
40
41          Widget* m_titlebar{ new Widget };
42
43          Label* m_window_title_label;
44
45          Button* m_exit_button = new Button(new Graphic(":/svgs/exit.svg", QSize(20, 20)), true);
46      };
47  }
48
49  #endif // DIALOG_H
```

7.20 directories.h

```
1  #ifndef DIRECTORIES_H
2  #define DIRECTORIES_H
3
4  #include <QString>
5
6  namespace Layers
7  {
8      QString app_path(const QString& app_name);
9      QString app_themes_path(const QString& app_name);
10      QString layers_path();
11      QString layers_themes_path();
12      QString local_app_data_path();
13  }
14
15  #endif // DIRECTORIES_H
```

7.21 Downloader.h

```
1  #ifndef DOWNLOADER_H
2  #define DOWNLOADER_H
3
4  #include <QFile>
5  #include <QNetworkAccessManager>
6  #include <QNetworkReply>
7
8  namespace Layers
9  {
10      class Downloader :  public QObject
11      {
12          Q_OBJECT
13
14      public:
15          Downloader(QObject* parent = 0);
16
17          QNetworkReply* download(const QUrl& file_url, const QDir& directory);
18
19          QNetworkReply* download(const QUrl& file_url);
20
21      private:
22          QNetworkAccessManager m_network_manager;
23      };
24  }
25
26  #endif // DOWNLOADER_H
```

7.22 FillControl.h

```
1  #ifndef FILLCONTROL_H
2  #define FILLCONTROL_H
3
4  #include <QGraphicsOpacityEffect>
5
6  #include "ColorControl.h"
7  #include "GradientControl.h"
8  #include "Label.h"
9  #include "ToggleSwitch.h"
10
11 namespace Layers
12 {
13     class FillControl :  public Widget
14     {
```

```
15          Q_OBJECT
16
17      public:
18          FillControl(QWidget* parent = nullptr);
19          ~FillControl();
20
21          void init_child_themeable_reference_list();
22
23          void set_attribute(Attribute* attribute);
24
25      public slots:
26          void set_current_editting_state(const QString& state);
27
28      protected:
29          bool eventFilter(QObject* object, QEvent* event);
30
31          void init_attributes();
32
33      private:
34          void setup_layout();
35
36          ColorControl* m_color_control{ new ColorControl };
37
38          Label* m_color_label{ new Label("Color") };
39
40          QGraphicsOpacityEffect* m_color_label_opacity{ new QGraphicsOpacityEffect };
41
42          Widget* m_dialog{ new Widget };
43
44          ToggleSwitch* m_fill_type_toggle{ new ToggleSwitch(true)};
45
46          GradientControl* m_gradient_control{ new GradientControl };
47
48          Label* m_gradient_label{ new Label("Gradient") };
49
50          QGraphicsOpacityEffect* m_gradient_label_opacity{ new QGraphicsOpacityEffect };
51      };
52  }
53
54  #endif // FILLCONTROL_H
```

7.23 GitHubRepo.h

```
1  #ifndef GITHUBREPO_H
2  #define GITHUBREPO_H
3
4  #include <QString>
5
6  namespace Layers
7  {
8      class GitHubRepo
9      {
10      public:
11          GitHubRepo(const QString& repo_url);
12
13          QString toString() const;
14
15      private:
16          QString m_repo_name{ "" };
17          QString m_user_name{ "" };
18      };
19  }
20
21  #endif // GITHUBREPO_H
```

7.24 GradientControl.h

```
1  #ifndef GRADIENTCONTROL_H
2  #define GRADIENTCONTROL_H
3
4  #include "Widget.h"
5
6  namespace Layers
7  {
8      class GradientControl :  public Widget
9      {
10         Q_OBJECT
11
12      signals:
```

```
13          void gradient_changed();
14
15      public:
16          GradientControl(QWidget* parent = nullptr);
17
18          //void set_attribute(Attribute* attribute);
19
20      public slots:
21          void set_current_editting_state(const QString& state);
22
23      protected:
24          bool eventFilter(QObject* object, QEvent* event);
25
26          void init_attributes();
27
28      //private:
29          //Attribute* m_attribute{ nullptr };
30      };
31  }
32
33  #endif // GRADIENTCONTROL_H
```

7.25 GradientSelectionDialog.h

```
1  #ifndef GRADIENTSELECTIONDIALOG_H
2  #define GRADIENTSELECTIONDIALOG_H
3
4  #include <QTimer>
5
6  #include "Button.h"
7  #include "ColorControl.h"
8  #include "Dialog.h"
9  #include "Graphic.h"
10
11 namespace Layers
12 {
13      class GradientSelectionDialog :  public Dialog
14      {
15          Q_OBJECT
16
17      public:
18          GradientSelectionDialog(QGradientStops gradient_stops, QWidget* parent = nullptr);
19
20          void add_gradient_stop(double stop_val, QColor color);
21
22          QGradientStops gradient_stops() const;
23
24          void update_gradient();
25
26      public slots:
27          void click_control();
28          void update_color_control_positions();
29
30      protected:
31          bool eventFilter(QObject* object, QEvent* event) override;
32
33          void init_attributes();
34          void init_child_themeable_reference_list();
35
36      private:
37          void init_color_controls();
38          void init_gradient_widget();
39
40          void setup_layout();
41
42          Widget* m_gradient_widget{ new Widget };
43
44          QGradientStops m_gradient_stops{ { 0.0, Qt::white },{ 1.0, Qt::black } };
45
46          QList<ColorControl*> color_controls;
47
48          Button* m_apply_button{ new Button("Apply") };
49
50          ColorControl* m_selected_color_control{ nullptr };
51          ColorControl* m_clicking_color_control{ nullptr };
52
53          QPoint m_selection_start_point;
54
55          int m_selected_control_start_x{ 0 };
56
57          QTimer m_single_click_timer;
58
59          bool m_moved{ false };
```

```
60      };
61  }
62
63  #endif // GRADIENTSELECTIONDIALOG_H
```

7.26 Graphic.h

```
1  #ifndef GRAPHIC_H
2  #define GRAPHIC_H
3
4  #include <QLabel>
5
6  #include "ImageSequenceLabel.h"
7  #include "SVG.h"
8  #include "Widget.h"
9
10 namespace Layers
11 {
12      class Graphic :  public Widget
13      {
14          Q_OBJECT
15
16      public:
17          Graphic(const ImageSequence& image_sequence, QSize size, QWidget* parent = nullptr);
18          Graphic(const QString& filepath, QSize size, QWidget* parent = nullptr);
19          Graphic(const QString& filepath, QWidget* parent = nullptr);
20          Graphic(const QImage& image, QWidget* parent = 0);
21          Graphic(const Graphic& gw);
22          ~Graphic();
23
24          QSize image_size();
25
26          void set_hovering(bool cond = true);
27          void set_icon(Graphic* icon);
28          void set_pixmap(const QPixmap& pixmap);
29          void set_size(QSize size);
30
31          SVG* svg() const;
32
33      private:
34          QSize m_image_size{ QSize() };
35          QLabel* m_bitmap_label{ nullptr };
36          SVG* m_svg_widget{ nullptr };
37          ImageSequenceLabel* m_image_sequence_label{ nullptr };
38      };
39 }
40
41 #endif // GRAPHIC_H
```

7.27 ImageSequence.h

```
1  #ifndef IMAGESEQUENCE_H
2  #define IMAGESEQUENCE_H
3
4  #include <QDir>
5
6  namespace Layers
7  {
8      class ImageSequence
9      {
10      public:
11          ImageSequence(QDir dir);
12
13          ImageSequence(QFile file);
14
15          void save(QFile file);
16
17          QList<QPixmap> to_pixmaps() const;
18
19      private:
20          QList<QImage> m_images{ QList<QImage>() };
21      };
22 }
23
24 #endif // IMAGESEQUENCE_H
```

7.28 ImageSequenceLabel.h

```
1 #ifndef IMAGESEQUENCELABEL_H
2 #define IMAGESEQUENCELABEL_H
3
4 #include <QLabel>
5 #include <QTimer>
6
7 #include "ImageSequence.h"
8
9 namespace Layers
10 {
11     class ImageSequenceLabel :  public QLabel
12     {
13         Q_OBJECT
14
15     public:
16         ImageSequenceLabel(ImageSequence image_sequence, QSize size, QWidget* parent = nullptr);
17         ImageSequenceLabel(const ImageSequenceLabel& isl);
18
19     public slots:
20         void time_out();
21
22     private:
23         int m_frame_number{ 0 };
24
25         QTimer m_timer;
26
27         QList<QPixmap> m_pixmaps{ QList<QPixmap>() };
28     };
29 }
30
31 #endif // IMAGESEQUENCELABEL_H
```

7.29 Label.h

```
1 #ifndef LABEL_H
2 #define LABEL_H
3
4 #include <QLabel>
5 #include <QPainter>
6
7 #include "Attribute.h"
8 #include "Themeable.h"
9
10 namespace Layers
11 {
12     class Label :  public QLabel, public Themeable
13     {
14         Q_OBJECT
15
16     public:
17         Label(QWidget* parent = nullptr);
18         Label(const QString& text, QWidget* parent = 0);
19
20         virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
21
22         void resize();
23
24         void build_wrapped_lines();
25
26         void setFont(const QFont& f);
27
28         void setMaximumWidth(int maxw);
29
30         void setWordWrap(bool on);
31
32         void set_available_width(int available_width);
33         void set_font_size(int size);
34         void set_hovering(bool cond = true);
35         void set_padding(double left, double top, double right, double bottom);
36         void set_resize_disabled(bool disable = true);
37
38         int width_unwrapped();
39
40         Attribute a_fill{ Attribute("fill", QColor(Qt::white), true) };
41         Attribute a_outline_color{ Attribute("outline_color", QColor(Qt::gray), true) };
42         Attribute a_padding_top{ Attribute("top_padding", QVariant::fromValue(0.0)) };
43         Attribute a_text_color{ Attribute("text_color", QColor(Qt::black)) };
44         Attribute a_text_hover_color{ Attribute("text_hover_color", QColor(Qt::black), true) };
45
46     public slots:
47         void setText(const QString& text);
```

```
48
49      protected:
50          void init_attributes();
51          void paintEvent(QPaintEvent* event);
52
53          QList<QString> m_wrapped_lines;
54
55          QPainter painter;
56
57          bool m_hovering{ false };
58          bool m_resize_disabled{ false };
59          bool m_wrapping{ false };
60
61          int m_available_width{ 16777215 };
62
63          int m_padding_left{ 0 };
64          //int m_padding_top{ 0 };
65          int m_padding_right{ 0 };
66          int m_padding_bottom{ 0 };
67      };
68  }
69
70  #endif // LABEL_H
```

7.30 Layouts.h

```
1  #ifndef LAYOUTS_H
2  #define LAYOUTS_H
3
4  #include <QHBoxLayout>
5  #include <QVBoxLayout>
6
7  namespace Layers
8  {
9      class HorizontalLayout :  public QHBoxLayout
10     {
11     public:
12         HorizontalLayout(QWidget* parent = nullptr);
13
14         void set_border_margin(int border_margin);
15         void setContentsMargins(int left, int top, int right, int bottom);
16
17         void update_margins();
18
19     protected:
20         int m_margin_left{ 0 };
21         int m_margin_top{ 0 };
22         int m_margin_right{ 0 };
23         int m_margin_bottom{ 0 };
24         int m_border_margin{ 0 };
25     };
26
27     class VerticalLayout :  public QVBoxLayout
28     {
29     public:
30         VerticalLayout(QWidget* parent = nullptr);
31
32         void set_border_margin(int border_margin);
33         void setContentsMargins(int left, int top, int right, int bottom);
34
35         void update_margins();
36
37     protected:
38         int m_margin_left{ 0 };
39         int m_margin_top{ 0 };
40         int m_margin_right{ 0 };
41         int m_margin_bottom{ 0 };
42         int m_border_margin{ 0 };
43     };
44 }
45
46 #endif // LAYOUTS_H
```

7.31 LineEditor.h

```
1  #ifndef LINEEDITOR_H
2  #define LINEEDITOR_H
3
4  #include <QLineEdit>
```

```
5
6 #include "Widget.h"
7
8 namespace Layers
9 {
10     class LineEditor :  public Widget
11     {
12         Q_OBJECT
13
14     signals:
15         void text_edited(const QString& text);
16
17     public:
18         LineEditor(QWidget* parent = nullptr);
19
20         virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
21
22         void reconnect_text_attribute();
23
24         void set_default_value(const QString& default_value);
25         void set_disabled(bool cond = true);
26         void set_font_size(int size);
27         void set_margin(int margin);
28         void set_margin(int left, int top, int right, int bottom);
29
30         void set_text(const QString& text);
31         void set_validator(const QValidator* validator);
32
33         void setFixedSize(int w, int h);
34
35         void setFixedWidth(int w);
36
37         QString text();
38
39         Attribute a_left_padding{ Attribute("left_padding", QVariant::fromValue(3.0)) };
40         Attribute a_text_color{ Attribute("text_color", QColor(Qt::black)) };
41         Attribute a_text{ Attribute("text", QString("")) };
42
43     public slots:
44         void set_current_editting_state(const QString& state);
45         void update_theme_dependencies();
46
47     protected:
48         bool eventFilter(QObject* object, QEvent* event) override;
49
50         void init_attributes();
51
52     private:
53         QString* m_default_value{ nullptr };
54
55         bool m_disabled{ false };
56
57         QLineEdit* m_line_edit{ new QLineEdit(this) };
58     };
59 }
60
61 #endif // LINEEDITOR_H
```

7.32 Menu.h

```
1 #ifndef MENU_H
2 #define MENU_H
3
4 #include "Widget.h"
5
6 namespace Layers
7 {
8     class Menu :  public Widget
9     {
10         Q_OBJECT
11
12     public:
13         Menu(const QString& menu_name, Graphic* menu_icon, QWidget* parent = nullptr);
14
15         Graphic* icon{ nullptr };
16
17         QString name;
18     };
19 }
20
21 #endif // MENU_H
```

7.33 MenuBar.h

```
1 #ifndef MENUBAR_H
2 #define MENUBAR_H
3
4 #include <QMenuBar>
5
6 #include "Attribute.h"
7 #include "Themeable.h"
8
9 namespace Layers
10 {
11     class MenuBar :  public QMenuBar, public Themeable
12     {
13         Q_OBJECT
14
15     public:
16         MenuBar(QWidget* parent = 0);
17
18         // TODO: Make override other overloaded versions from the parent class
19         QMenu* addMenu(const QString& title);
20
21         virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
22
23         //void issue_update();
24
25         void update_theme_dependencies();
26
27         Attribute a_text_color{ Attribute("text_color", QColor(Qt::gray)) };
28         Attribute a_selected_text_color{ Attribute("selected_text_color", QColor(Qt::lightGray)) };
29
30         //Attribute a_text_color{ Attribute("Text Color", QColor(Qt::red)) };
31         //Attribute a_selected_text_color{ Attribute("Selected Text Color", QColor(Qt::blue)) };
32
33     protected:
34         QString build_stylesheet();
35
36         void init_attributes();
37
38         //void paintEvent(QPaintEvent* event);
39
40     private:
41         QList<QMenu*> m_menus{ QList<QMenu*>() };
42     };
43 }
44
45 #endif // MENUBAR_H
```

7.34 MenuLabelLayer.h

```
1 #ifndef MENULABELLAYER_H
2 #define MENULABELLAYER_H
3
4 #include "Button.h"
5 #include "Graphic.h"
6 #include "Label.h"
7 #include "Menu.h"
8
9 namespace Layers
10 {
11     class MenuLabelLayer :  public Widget
12     {
13         Q_OBJECT
14
15     public:
16         MenuLabelLayer(Menu* menu, QWidget* parent = nullptr);
17
18         void shrink();
19         void expand();
20
21         Button* back_button() const;
22         Button* icon_button() const;
23
24         Label* text_label() const;
25
26     protected:
27         void init_attributes();
28         void init_child_themeable_reference_list();
29
30         void setup_layout();
31
32     private:
33         QHBoxLayout* main_layout = new QHBoxLayout;
```

```
34
35          Button* m_back_button{ new Button(new Graphic(":/svgs/back.svg", QSize(21, 18)), false) };
36          Button* m_icon_button{ nullptr };
37
38          Label* m_text_label{ nullptr };
39
40          Widget* m_stretch_widget{ new Widget };
41      };
42  }
43
44  #endif // MENULABELLAYER_H
```

7.35 MiniSlider.h

```
1  #ifndef MINISLIDER_H
2  #define MINISLIDER_H
3
4  #include "Widget.h"
5
6  namespace Layers
7  {
8      class MiniSlider :  public Widget
9      {
10         Q_OBJECT
11
12     public:
13         MiniSlider(double limit, QWidget* parent = nullptr);
14
15         void update_handle_pos();
16         void update_theme_dependencies();
17
18         Attribute a_value{ Attribute("value", QVariant::fromValue(0.0)) };
19
20     public slots:
21         void set_current_editting_state(const QString& state);
22
23     protected:
24         bool eventFilter(QObject* object, QEvent* event) override;
25
26         void init_attributes();
27         void init_child_themeable_reference_list();
28
29     private:
30         void setup_layout();
31
32         Widget* m_bar{ new Widget };
33         Widget* m_handle{ new Widget(this) };
34
35         double m_limit{ 99.0 };
36
37         int m_mouse_move_scale{ 5 };
38         int m_value_on_click{ 0 };
39
40         bool m_dragging_handle{ false };
41
42         QPoint m_mouse_click_position{ QPoint() };
43     };
44 }
45
46 #endif // MINISLIDER_H
```

7.36 ScrollArea.h

```
1  #ifndef SCROLLAREA_H
2  #define SCROLLAREA_H
3
4  #include <QScrollArea>
5  //#include <QScrollBar>
6
7  #include "ScrollBar.h"
8  #include "Widget.h"
9
10 namespace Layers
11 {
12     class ScrollArea :  public Widget
13     {
14         Q_OBJECT
15
16     public:
```

```
17          ScrollArea(QWidget* parent = nullptr);
18
19          ScrollBar* horizontal_scrollbar() const;
20
21          void setHorizontalScrollBarPolicy(Qt::ScrollBarPolicy policy);
22
23          void setVerticalScrollBarPolicy(Qt::ScrollBarPolicy policy);
24
25          void setWidget(QWidget* widget);
26
27          ScrollBar* vertical_scrollbar() const;
28
29      protected:
30          bool eventFilter(QObject* object, QEvent* event) override;
31
32          void init_child_themeable_reference_list();
33
34          QScrollArea* m_scroll_area{ new QScrollArea(this) };
35
36          ScrollBar* m_horizontal_scrollbar{ new ScrollBar };
37          ScrollBar* m_vertical_scrollbar{ new ScrollBar };
38      };
39  }
40
41  #endif // SCROLLAREA_H
```

7.37 ScrollBar.h

```
1  #ifndef SCROLLBAR_H
2  #define SCROLLBAR_H
3
4  #include <QScrollBar>
5
6  #include "Attribute.h"
7  #include "Themeable.h"
8
9  namespace Layers
10 {
11      class ScrollBar :  public QScrollBar, public Themeable
12      {
13          Q_OBJECT
14
15      public:
16          ScrollBar(QWidget* parent = 0);
17
18          virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
19
20          void update_theme_dependencies();
21
22          Attribute a_background_color{ Attribute("background_color", QColor(Qt::gray)) };
23          Attribute a_handle_color{ Attribute("handle_color", QColor(Qt::white)) };
24
25          CornerRadiiAttributes corner_radii;
26
27          CornerRadiiAttributes handle_corner_radii{ CornerRadiiAttributes("handle_corner_radii") };
28
29      protected:
30          QString build_stylesheet();
31
32          void init_attributes();
33      };
34  }
35
36  #endif // SCROLLBAR_H
```

7.38 SettingsMenu.h

```
1  #ifndef SETTINGSMENU_H
2  #define SETTINGSMENU_H
3
4  #include <QHBoxLayout>
5
6  #include "Layouts.h"
7  #include "Menu.h"
8  #include "SettingsPanels.h"
9
10 namespace Layers
11 {
12      class Label;
```

```
13
14      class SettingsTab :  public Widget
15      {
16          Q_OBJECT
17
18      signals:
19          void clicked();
20          void under_minimum_width();
21          void over_minimum_width();
22
23      public:
24          SettingsTab(Graphic* icon, const QString& label_text, QWidget* parent = nullptr);
25
26          void expand();
27          void shrink();
28
29          int recommended_minimum_width();
30
31          void set_disabled(bool cond = true);
32
33      protected:
34          bool eventFilter(QObject* object, QEvent* event) override;
35
36          void init_attributes();
37          void init_child_themeable_reference_list();
38
39          void resizeEvent(QResizeEvent* event);
40
41      private:
42          void setup_layout();
43
44          QHBoxLayout* main_layout{ new QHBoxLayout };
45
46          bool m_disabled{ false };
47
48          Label* m_text_label;
49
50          Graphic* m_tab_icon;
51
52          Widget* m_spacer{ new Widget };
53          Widget* m_stretch_widget{ new Widget };
54          Widget* m_stretch_widget2{ new Widget };
55      };
56
57      class SettingsSidebar :  public Widget
58      {
59          Q_OBJECT
60
61      public:
62          SettingsSidebar(QWidget* parent = nullptr);
63      };
64
65      class SettingsMenu :  public Menu
66      {
67          Q_OBJECT
68
69      public:
70          SettingsMenu(QWidget* parent = nullptr);
71
72          void add_settings_tab(Graphic* icon, const QString& label_text);
73
74          int largest_tab_index() const;
75
76          int recommended_minimum_tab_width() const;
77
78          ThemesSettingsPanel* themes_settings_panel() const;
79
80      protected:
81          bool eventFilter(QObject* object, QEvent* event) override;
82
83          void init_child_themeable_reference_list();
84
85      private slots:
86          void shrink_tabs();
87          void expand_tabs();
88
89      private:
90          void setup_layout();
91
92          bool m_dragging_sidebar{ false };
93          bool m_frozen{ false };
94          bool m_hovering_over_divider{ false };
95          bool m_shrunk{ false };
96
97          VerticalLayout* m_sidebar_layout = new VerticalLayout;
98
99          QList<SettingsTab*> m_settings_tabs;
```

```
100
101         QPoint last_pos;
102
103         SettingsSidebar* m_sidebar{ new SettingsSidebar };
104
105         AppPreferencesSettingsPanel* m_app_preferences_settings_panel{ new AppPreferencesSettingsPanel
    };
106
107         ThemesSettingsPanel* m_themes_settings_panel{ new ThemesSettingsPanel };
108     };
109 }
110
111 #endif // SETTINGSMENU_H
```

7.39 SettingsPanels.h

```
1 #ifndef SETTINGSPANELS_H
2 #define SETTINGSPANELS_H
3
4 #include "Button.h"
5 #include "Combobox.h"
6 #include "Graphic.h"
7
8 namespace Layers
9 {
10     class AppPreferencesSettingsPanel :  public Widget
11     {
12         Q_OBJECT
13
14     public:
15         AppPreferencesSettingsPanel(QWidget* parent = nullptr);
16     };
17
18     class ThemesSettingsPanel :  public Widget
19     {
20         Q_OBJECT
21
22     public:
23         ThemesSettingsPanel(QWidget* parent = nullptr);
24
25         void apply_theme(Theme& theme);
26
27         Button* customize_theme_button() const;
28
29         Button* new_theme_button() const;
30
31         Combobox* theme_combobox() const;
32
33         void show_custom_theme_buttons(bool cond = true);
34
35     protected:
36         void init_attributes();
37         void init_child_themeable_reference_list();
38
39     private:
40         void setup_layout();
41
42         Label* m_theme_label{ new Label("Theme") };
43
44         Combobox* m_theme_combobox{ new Combobox };
45
46         Button* m_new_theme_button{ new Button(new Graphic(":/svgs/new_theme.svg", QSize(20, 20)), true)
    };
47         Button* m_customize_theme_button{ new Button(new Graphic(":/svgs/customize_theme.svg", QSize(20,
    20)), true) };
48         Button* m_delete_theme_button{ new Button(new Graphic(":/svgs/delete_theme.svg", QSize(17, 20)),
    true) };
49         Button* m_theme_info_button{ new Button(new Graphic(":/svgs/info_theme.svg", QSize(20, 20)),
    true) };
50
51         Widget* m_control_separator{ new Widget };
52         Widget* m_separator_1{ new Widget };
53         Widget* m_separator_2{ new Widget };
54         Widget* m_spacer_1{ new Widget };
55         Widget* m_spacer_2{ new Widget };
56     };
57 }
58
59 #endif // SETTINGSPANELS_H
```

7.40 Slider.h

```
1 #ifndef SLIDER_H
2 #define SLIDER_H
3
4 #include "Widget.h"
5
6 namespace Layers
7 {
8     class Slider :  public Widget
9     {
10         Q_OBJECT
11
12     signals:
13         void value_changed(int value);
14
15     public:
16         Slider(QWidget* parent = nullptr);
17         Slider(int limit, QWidget* parent = nullptr);
18
19         void set_limit(int limit);
20
21         void set_value(double value);
22
23         Attribute a_value{ Attribute("value", QVariant::fromValue(0.0)) };
24
25     public slots:
26         void update_handle_pos();
27
28     protected:
29         bool eventFilter(QObject* object, QEvent* event) override;
30
31         void init_attributes();
32         void init_child_themeable_reference_list();
33
34     private:
35         void init();
36
37         void setup_layout();
38
39         Widget* m_bar{ new Widget };
40         Widget* m_handle{ new Widget(this) };
41
42         int m_limit{ 99 };
43         double m_value_on_click{ 0.0 };
44
45         bool m_dragging_handle{ false };
46
47         bool m_is_ratio_slider;
48
49         QPoint m_mouse_click_position{ QPoint() };
50     };
51 }
52
53 #endif // SLIDER_H
```

7.41 SVG.h

```
1 #ifndef SVG_H
2 #define SVG_H
3
4 #include <QSvgWidget>
5
6 #include "Themeable.h"
7
8 namespace Layers
9 {
29     class SVG :  public QSvgWidget, public Themeable
30     {
31         Q_OBJECT
32
33     public:
37         SVG(QString file_path, QWidget* parent = nullptr);
38
42         SVG(const SVG& svg_w);
43
44         virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
45
49         void rebuild_svg_str();
50
51         void set_hovering(bool cond = true);
52
53         virtual void set_state(const QString& state) override;
```

```
54
62          void update();
63
64          Attribute a_common_color{ Attribute("common_color", QColor(Qt::black)) };
65          Attribute a_common_hover_color{ Attribute("common_hover_color", QColor(Qt::darkGray)) };
66          Attribute a_use_common_color{ Attribute("use_common_color", QVariant::fromValue(false)) };
67          Attribute a_use_common_hover_color{ Attribute("use_common_hover_color",
      QVariant::fromValue(false)) };
68
69      protected:
75          void init_attributes();
76
77      private:
78          void init_size();
79
80          void build_svg_elements_list();
81
82          QString element_id(const QString& element);
83
84          bool m_theming_blocked{ false };
85
86          bool m_hovering{ false };
87
88          QString m_svg_str{  };
89
90          QStringList m_svg_elements{  };
91      };
92  }
93
94  #endif // SVG_H
```

7.42 TabBar.h

```
1  #ifndef TABBAR_H
2  #define TABBAR_H
3
4  #include <QTabBar>
5
6  #include "Attribute.h"
7  #include "Themeable.h"
8
9  namespace Layers
10 {
11      class TabBar :  public QTabBar, public Themeable
12      {
13          Q_OBJECT
14
15      public:
16          TabBar(QWidget* parent = 0);
17
18          virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs) override;
19
20          void SetCurrentTab(const QString& text);
21
22          bool ContainsTab(const QString& text);
23
24          //void removeTab(int index);
25          void removeTab(const QString& text);
26
27          void update_theme_dependencies();
28
29          Attribute a_selected_fill_color{ Attribute("selected_fill_color", QColor(Qt::gray)) };
30          Attribute a_text_color{ Attribute("text_color", QColor(Qt::white)) };
31
32      protected:
33          QString build_stylesheet();
34
35          void init_attributes();
36      };
37  }
38
39  #endif // TABBAR_H
```

7.43 Theme.h

```
1  #ifndef THEME_H
2  #define THEME_H
3
4  #include <QDataStream>
```

```
5 #include <QHash>
6 #include <QJsonDocument>
7 #include <QString>
8 #include <QUuid>
9
10 #include "Attribute.h"
11
12 namespace Layers
13 {
14     class Attribute;
15     class Themeable;
16
17     // NOTE: Below has not been updated to support AttributeType
18     //inline QDataStream& operator <<(QDataStream& stream, const QMap<QString, Attribute*>& attr_map)
19     //{
20     //  stream << attr_map.count();
21
22     //  for (const QString& attr_tag :  attr_map.keys())
23     //  {
24     //      stream << attr_tag;
25     //      stream << *attr_map[attr_tag];
26     //  }
27     //
28     //  return stream;
29     //}
30
31     // NOTE: Below has not been updated to support AttributeType
32     //inline QDataStream& operator >>(QDataStream& stream, QMap<QString, Attribute*>& attr_map)
33     //{
34     //  qsizetype attr_count;
35     //
36     //  stream >> attr_count;
37     //
38     //  for (int i = 0; i < attr_count; i++)
39     //  {
40     //      QString attr_tag = "";
41     //      Attribute* attr = new Attribute("");
42     //
43     //      stream >> attr_tag;
44     //      stream >> *attr;
45     //
46     //      attr_map[attr_tag] = attr;
47     //  }
48     //
49     //  return stream;
50     //}
51
52     enum class ThemeDataType
53     {
54         All, Application, Layers
55     };
56
63     class Theme
64     {
65     public:
66         Theme();
67         Theme(const QString& name, bool editable = true);
68         Theme(const QJsonDocument& json_document, QUuid* uuid = nullptr);
69
78         //void add_attributes(
79         //  const QString& themeable_tag,
80         //  QMap<QString, Attribute*> attributes);
81
82         void clear();
83
84         void consume(Theme&& theme);
85
92         bool contains_attributes_for_tag(const QString& themeable_tag);
93
99         void copy(Theme& theme);
100
101          void copy_attribute_values_of(Themeable* themeable);
102
108          bool editable();
109
110          QString identifier();
111
112          Attribute* init_attribute(const QString& name, bool disabled, const QJsonValue& attr_value);
113
119          QString& name();
120
126          void set_name(const QString& new_name);
127
133          QList<QString> themeable_tags();
134
135          QJsonDocument to_json_document(ThemeDataType data_type = ThemeDataType::All);
136
```

```
145         QMap<QString, AttributeType*>& operator[](const QString& themeable_tag);
146
147         //friend QDataStream& operator <<(QDataStream& stream, const Theme& t)
148         //{
149         //  stream << t.m_data;
150         //  stream << t.m_editable;
151         //  stream << t.m_name;
152         //  return stream;
153         //}
154
155         //friend QDataStream& operator >>(QDataStream& stream, Theme& t)
156         //{
157         //  stream >> t.m_data;
158         //  stream >> t.m_editable;
159         //  stream >> t.m_name;
160         //  return stream;
161         //}
162
163     private:
164         QHash<QString, QMap<QString, AttributeType*>> m_data{ QHash<QString, QMap<QString,
      AttributeType*>>() };
165
166         bool m_editable{ true };
167
168         QString m_name{ "" };
169
170         QUuid* m_uuid{ nullptr };
171     };
172 }
173
174 #endif // THEME_H
```

7.44 theme_loading.h

```
1 #ifndef THEME_LOADING_H
2 #define THEME_LOADING_H
3
4 #include <QDataStream>
5 #include <QDir>
6 #include <QFile>
7
8 #include "Theme.h"
9
10 namespace Layers
11 {
12     Theme load_theme_1(const QString& file_name, const QString& app_identifier);
13 }
14
15 #endif // THEME_LOADING_H
```

7.45 Themeable.h

```
1 #ifndef THEMEABLE_H
2 #define THEMEABLE_H
3
4 #include "AttributeGroup.h"
5
6 namespace Layers
7 {
8     class AttributeWidget;
9     class CustomizePanel;
10     class Graphic;
11     class Theme;
12
33     class Themeable
34     {
35     public:
36         ~Themeable();
37
47         void store_child_themeable_pointer(Themeable* child_themeable);
48
60         virtual void apply_theme(Theme& theme);
61
62         virtual void apply_theme_attributes(QMap<QString, AttributeType*>& theme_attrs);
63
74         void assign_tag_prefixes(QList<QString> parent_prefixes = QList<QString>(), const QString&
      parent_name = "");
75
82         QMap<QString, AttributeType*>& attributes();
```

```
83
84          QList<AttributeType*>& attribute_layout();
85
86          //QMap<QString, AttributeWidget*>& attribute_widgets();
87
88          QList<Themeable*>& child_themeable_references();
89
90          void copy_attribute_values_to(Theme* theme);
91
97          Theme* current_theme();
98
104          CustomizePanel* customize_panel();
105
111          Graphic* icon() const;
112
121          //void initialize_and_acquire_panels(QList<CustomizePanel*>& list);
122
123          bool is_stateful() const;
124
131          //virtual void issue_update() = 0;
132
133          QString* name() const;
134
140          QString* proper_name() const;
141
147          void reapply_theme();
148
154          void remove_child_themeable_reference(Themeable* child_themeable);
155
156          template<typename T>
157          void replace_all_attributes_with(T* themeable);
158
159          void set_is_app_themeable(bool is_app_themeable);
160
161          void set_functionality_disabled(bool disabled = true);
162
163          //void set_attribute_value(
164          //  const QString& state,
165          //  const QString& attribute_name,
166          //  QVariant value);
167
168          //void set_attribute_value(
169          //  const QString& attribute_name,
170          //  QVariant value);
171
177          void set_icon(Graphic* icon);
178
187          void set_name(const QString& name);
188
197          void set_proper_name(const QString& proper_name);
198
206          virtual void set_state(const QString& state);
207
208          QString state() const;
209
216          //QMap<QString, StatefulAttribute>& stateful_attributes();
217
223          QList<QString> states() const;
224
241          QString& tag();
242
249          void unassign_prefixes();
250
258          //virtual void update_theme_dependencies();
259
260      protected:
270          //void init_attributes();
271
272          //virtual void init_attribute_widgets();
273
284          //CustomizePanel* init_customize_panel();
285
297          virtual void init_child_themeable_reference_list();
298
299          bool m_functionality_disabled{ false };
300          bool m_tag_prefixes_assigned{ false };
301          bool m_shared_attributes{ false };
302          bool m_is_app_themeable{ false };
303          bool m_is_stateful{ false };
304
305          CustomizePanel* m_customize_panel{ nullptr };
306
307          Graphic* m_icon{ nullptr };
308
309          QString* m_name{ nullptr };
310          QString* m_proper_name{ nullptr };
311          QString m_state{ "" };
```

```
312         QString m_tag{ "" };
313
314         //QMap<QString, bool> m_ACW_pre_init_primary_values{ QMap<QString, bool>() };
315         //QMap<QString, AttributeWidget*> m_attribute_widgets{ QMap<QString, AttributeWidget*>() };
316
317         QList<AttributeType*> m_attribute_layout{ QList<AttributeType*>() };
318
319         QMap<QString, AttributeType*> m_attributes{ QMap<QString, AttributeType*>() };
320
321         QList<Themeable*> m_child_themeables;
322
323         QList<QString> m_filtered_attributes;
324         QList<QString> m_tag_prefixes;
325
326         Theme* m_current_theme{ nullptr };
327     };
328
329     template<typename T>
330     inline void Themeable::replace_all_attributes_with(T* themeable)
331     {
332         if (typeid(*this) == typeid(*themeable))
333         {
334             for (const QString& attr_type_key :  m_attributes.keys())
335             {
336                 if (Attribute* attr = dynamic_cast<Attribute*>(m_attributes[attr_type_key]))
337
      attr->entangle_with(*dynamic_cast<Attribute*>(themeable->m_attributes[attr_type_key]));
338                 else if (AttributeGroup* attr_group =
      dynamic_cast<AttributeGroup*>(m_attributes[attr_type_key]))
339
      attr_group->entangle_with(*dynamic_cast<AttributeGroup*>(themeable->m_attributes[attr_type_key]));
340             }
341
342             for (Themeable* this_child_themeable :  m_child_themeables)
343                 if (this_child_themeable->m_name)
344                     for (Themeable* child_themeable :  themeable->m_child_themeables)
345                         if (*child_themeable->m_name == *this_child_themeable->m_name)
346                             this_child_themeable->replace_all_attributes_with(child_themeable);
347         }
348     }
349 }
350
351 #endif // THEMEABLE_H
```

7.46 ThemeableBox.h

```
1 #ifndef THEMEABLEBOX_H
2 #define THEMEABLEBOX_H
3
4 #include <QPainter>
5
6 #include "Attribute.h"
7 #include "Themeable.h"
8
9 namespace Layers
10 {
14     class ThemeableBox :  public Themeable
15     {
16     public:
17         virtual void apply_theme_attributes(
18             QMap<QString, AttributeType*>& theme_attrs) override;
19
25         void set_margin(double margin);
26
35         void set_margin(double left, double top, double right, double bottom);
36
37         BorderAttributes border;
38
39         CornerRadiiAttributes corner_radii;
40
41         MarginsAttributes margins;
42
43         Attribute a_corner_color{ Attribute(
44             "corner_color",
45             QColor(Qt::gray),
46             true
47             ) };
48
49         Attribute a_fill{ Attribute(
50             "fill",
51             QColor(Qt::white)
52             ) };
53
```

```
54          Attribute a_hover_fill{ Attribute(
55              "hover_fill",
56              QColor(Qt::lightGray),
57              true
58              ) };
59
60          Attribute a_outline_color{ Attribute(
61              "outline_color",
62              QColor(Qt::gray),
63              true
64              ) };
65
66      protected:
70          //bool eventFilter(QObject* object, QEvent* event);
71
80          void init_attributes();
81
85          void paint(QWidget* widget);
86
87          bool m_hovering{ false };
88      };
89  }
90
91  #endif // THEMEABLEBOX_H
```

7.47 Titlebar.h

```
1  #ifndef TITLEBAR_H
2  #define TITLEBAR_H
3
4  #include "MenuLabelLayer.h"
5
6  namespace Layers
7  {
8      class Titlebar :  public Widget
9      {
10         Q_OBJECT
11
12     signals:
13         void window_icon_updated();
14
15     public:
16         Titlebar(QWidget* parent = nullptr);
17
18         void add_mll(MenuLabelLayer* mll);
19         void remove_mlls_past(int index);
20
21         bool is(QWidget* widget);
22
23         void set_window_icon(const Graphic& icon_graphic);
24
25         void set_window_title(const QString& title);
26
27         Button* window_icon() const;
28         Button* settings_button() const;
29         Button* minimize_button() const;
30         Button* maximize_button() const;
31         Button* exit_button() const;
32
33     protected:
34         void init_child_themeable_reference_list();
35
36         void resizeEvent(QResizeEvent* event);
37
38         void setup_layout();
39
40     private:
41         QHBoxLayout* main_layout = new QHBoxLayout;
42
43         Button* m_window_icon{ new Button(new Graphic(QFile(":/image_sequences/layers_logo.imgseq"),
    QSize(35, 35)), true) };
44
45         Label* m_window_title_label{ new Label("Window") };
46
47         Button* m_settings_button{ new Button(new Graphic(":/svgs/settings.svg", QSize(20, 20)), true) };
48         Button* m_minimize_button{ new Button(new Graphic(":/svgs/minimize.svg", QSize(20, 20)), true) };
49         Button* m_maximize_button{ new Button(new Graphic(":/svgs/maximize.svg", QSize(20, 20)), true) };
50         Button* m_exit_button{ new Button(new Graphic(":/svgs/exit.svg", QSize(20, 20)), true) };
51
52         Widget* m_buttons_container{ new Widget(this) };
53
54         QList<MenuLabelLayer*> mll_stack;
55
```

```
56          MenuLabelLayer* m_base_mll{ nullptr };
57
58          MenuLabelLayer* m_control_mll{ new MenuLabelLayer(new Menu("Control Menu", new
        Graphic(":/svgs/icon_icon.svg", QSize(20, 20)))) };
59
60          Widget* m_stretch_widget{ new Widget };
61      };
62  }
63
64  #endif // TITLEBAR_H
```

7.48 ToggleSwitch.h

```
1  #ifndef TOGGLESWITCH_H
2  #define TOGGLESWITCH_H
3
4  #include <QHBoxLayout>
5  #include <QVBoxLayout>
6
7  #include "Widget.h"
8
9  namespace Layers
10 {
11      class Label;
12
13      class ToggleSwitch :  public Widget
14      {
15          Q_OBJECT
16
17      signals:
18          void toggled_event();
19
20      public:
21          ToggleSwitch(bool vertical = false, QWidget* parent = nullptr);
22
23          void setFixedHeight(int h);
24
25          void set_toggled(bool toggled);
26
27          void toggle(bool emit_toggled_event = true);
28
29          bool toggled() const;
30
31          void update_layout_margins();
32
33          void update_spacer_size();
34
35          Attribute a_padding_left{ Attribute("Left Padding", QVariant::fromValue(2.0)) };
36          Attribute a_padding_top{ Attribute("Top Padding", QVariant::fromValue(2.0)) };
37          Attribute a_padding_right{ Attribute("Right Padding", QVariant::fromValue(2.0)) };
38          Attribute a_padding_bottom{ Attribute("Bottom Padding", QVariant::fromValue(2.0)) };
39
40      protected:
41          bool eventFilter(QObject* object, QEvent* event) override;
42
43          void init_attributes();
44
45          void init_child_themeable_reference_list();
46
47      private:
48          void setup_layout();
49
50          QHBoxLayout* m_layout_h{ nullptr };
51
52          QVBoxLayout* m_layout_v{ nullptr };
53
54          Widget* m_spacer{ new Widget };
55          Widget* m_square{ new Widget };
56
57          bool m_vertical{ false };
58      };
59  }
60
61  #endif // TOGGLESWITCH_H
```

7.49 UpdateDialog.h

```
1  #ifndef UPDATEDIALOG_H
2  #define UPDATEDIALOG_H
```

```
3
4 #include "Dialog.h"
5
6 namespace Layers
7 {
8     class UpdateDialog :  public Dialog
9     {
10         Q_OBJECT
11
12     public:
13         UpdateDialog(const QString& current_version_tag, const QString& latest_version_tag, QWidget*
      parent = nullptr);
14
15     protected:
16         void init_child_themeable_reference_list();
17
18     private:
19         void setup_layout();
20
21         Button* m_remind_me_later_button{ new Button("Remind Me Later") };
22         Button* m_update_button{ new Button("Update") };
23
24         Label* m_message_label;
25     };
26 }
27
28 #endif // UPDATEDIALOG_H
```

7.50 Variant.h

```
1 #ifndef VARIANT_H
2 #define VARIANT_H
3
4 #include <QColor>
5 #include <QGradient>
6 #include <QVariant>
7
8 namespace Layers
9 {
20     class Variant :  public QObject
21     {
22         Q_OBJECT
23
24     signals:
25         void changed();
26
27     public:
28         Variant();
29         Variant(double d);
30         Variant(QColor color);
31         Variant(QVariant qvariant);
32         Variant(const Variant& variant);
33
34         void operator=(const Variant& variant);
35         void operator=(const QVariant& qvariant);
36         bool operator!=(const QVariant& qvariant);
37
43         const char* typeName() const;
44
50         template<typename T>
51         T value() const;
52
53     private:
54         QVariant m_qvariant;
55     };
56
57     template<typename T>
58     inline T Variant::value()const
59 {
60         return m_qvariant.value<T>();
61     }
62
63     using VariantMap = QMap<QString, Variant>;
64
65 }
66
67 #endif // VARIANT_H
```

7.51 Version.h

```
1 #ifndef VERSION_H
```

```
2 #define VERSION_H
3
4 #include <QList>
5 #include <QString>
6
7 namespace Layers
8 {
9     class Version
10     {
11     public:
12         Version(int major, int minor = 0, int patch = 0, QString phase = "");
13
14         Version(QString version_string);
15
16         Version();
17
18         QString toString();
19
20     private:
21         const QList<QString> m_acceptable_phases{ QList<QString>({
22             "alpha", "a", "beta", "b", "release-candidate", "rc" }) };
23
24         int m_major{ 0 };
25         int m_minor{ 0 };
26         int m_patch{ 0 };
27
28         QString m_phase{ "" };
29
30         QString m_separator_charactor{ "" };
31     };
32 }
33
34 #endif // VERSION_H
```

7.52 Widget.h

```
1 #ifndef WIDGET_H
2 #define WIDGET_H
3
4 #include <QWidget>
5
6 #include "ThemeableBox.h"
7
8 namespace Layers
9 {
15     class Widget :  public QWidget, public ThemeableBox
16     {
17         Q_OBJECT
18
19     public:
20         Widget(QWidget* parent = nullptr);
21
22     protected:
26         bool eventFilter(QObject* object, QEvent* event) override;
27
36         void init_attributes();
37
41         void paintEvent(QPaintEvent* event) override;
42     };
43 }
44
45 #endif // WIDGET_H
```

7.53 Window.h

```
1 #ifndef WINDOW_H
2 #define WINDOW_H
3
4 #include "ColorDialog.h"
5 #include "CreateNewThemeDialog.h"
6 #include "CustomizeMenu.h"
7 #include "GradientSelectionDialog.h"
8 #include "SettingsMenu.h"
9 #include "Titlebar.h"
10 #include "UpdateDialog.h"
11
12 namespace Layers
13 {
14     class Window :  public Widget
```

```
15      {
16          Q_OBJECT
17
18      public:
19          Window(bool preview = false, QWidget* parent = nullptr);
20
21          void add_menu(Menu* menu);
22
23          Menu* app_menu() const;
24
25          void apply_theme(Theme& theme);
26
27          void assign_tag_prefixes();
28
29          template<typename T>
30          void build_main_widget();
31
32          void center_dialog(QDialog* dialog);
33
34          ColorDialog* control_color_dialog() const;
35
36          GradientSelectionDialog* control_gradient_selection_dialog() const;
37
38          CustomizeMenu* customize_menu() const;
39
40          void finalize();
41
42          void link_theme_name(const QString& name);
43
44          void set_main_widget(Widget* main_widget);
45
46          void set_window_icon(const Graphic& icon_graphic);
47
48          void set_window_title(const QString& title);
49
50          SettingsMenu* settings_menu() const;
51
52          Titlebar* titlebar() const;
53
54          void update_theme_dependencies();
55
56      public slots:
57          void customize_clicked();
58          void exit_clicked();
59          void maximize_clicked();
60          void minimize_clicked();
61          void new_theme_clicked();
62          void open_menu(Menu* menu);
63          void settings_clicked();
64
65      protected:
66          void init_child_themeable_reference_list();
67
68          bool nativeEvent(const QByteArray& eventType, void* message, qintptr* result) override;
69
70          void paintEvent(QPaintEvent* event) override;
71
72      private:
73          void setup_layout();
74
75          bool m_maximized{ false };
76          bool m_preview{ false };
77
78          CreateNewThemeDialog* m_create_new_theme_dialog{ new CreateNewThemeDialog };
79
80          ColorDialog* m_control_color_dialog{ new ColorDialog };
81
82          GradientSelectionDialog* m_control_gradient_selection_dialog{ new
    GradientSelectionDialog(QGradientStops()) };
83
84          UpdateDialog* m_control_update_dialog{ new UpdateDialog("", "")};
85
86          QVBoxLayout* m_app_menu_layout{ new QVBoxLayout };
87          QVBoxLayout* m_main_layout{ new QVBoxLayout };
88
89          QList<Menu*> m_menus;
90          QList<Menu*> m_menu_stack;
91
92          Titlebar* m_titlebar{ new Titlebar };
93
94          // TODO: Make Menu constructor that does not require an icon
95          Menu* m_app_menu{ new Menu("App", new Graphic(":/svgs/settings_animated.svg", QSize(24, 24))) };
96
97          SettingsMenu* m_settings_menu{ new SettingsMenu };
98
99          CustomizeMenu* m_customize_menu{ new CustomizeMenu };
100
```

```
101         Widget* m_main_widget{ nullptr };
102     };
103 
104     template<typename T>
105     inline void Window::build_main_widget()
106     {
107         m_main_widget = new T;
108 
109         //m_main_widget->set_icon(new Graphic(layersApp->icon_file()->fileName()));
110         m_main_widget->set_is_app_themeable(true);
111         //m_main_widget->apply_theme(*layersApp->current_theme());
112         m_main_widget->apply_theme(*m_current_theme);
113 
114         store_child_themeable_pointer(m_main_widget);
115 
116         m_app_menu_layout->addWidget(m_main_widget);
117 
118         if (m_customize_menu->preview_widget())
119         {
120             Window* preview_window = dynamic_cast<Window*>(m_customize_menu->preview_widget());
121 
122             if (preview_window)
123                 preview_window->build_main_widget<T>();
124         }
125     }
126 }
127 
128 #endif // WINDOW_H
```

Index