Absolute Documentation

Release 0.2.0

Colin Duquesnoy

Mar 16, 2019

Contents:

CHAPTER 1

About

Absolute is a QtQuickControls Style and Extension Plugin that provides an easily customizable theme and a set of extra controls for building **Desktop Applications**.

We made a style/theme that is specifically designed for desktop applications (less padding, mouse interaction by default,...)

The theme is based on the concept of contrast and transparency: you defined a base color and a contrast color that is the opposite of the base color. The base color is used as the background color of Root container (ApplicationWindow, Dialog, Popup) and the contrast color is used as the background of any other control with a very low opacity. This means you can stack containers on top of each others and always achieve a good automatic contrast.

The theme is not (and will never be) optimized for mobile devices!!!

1.1 Resources

- Documentation
- Screenshots
- Wiki
- Examples

1.2 QtQuick Controls 2 Style Status

Here is the current status of the QQC2 Style.

Note: Not all controls have been styled. Some controls were left out because we think they are not that useful for desktop applications. If you're missing one of the unstyled controls, feel free to submit a Merge Request!

1.2.1 List of styled controls

- ApplicationWindow
- BusyIndicator
- Button
- CheckBox
- CheckDelegate
- ComboBox
- Dialog
- DialogButtonBox
- Drawer
- Frame
- GroupBox
- ItemDelegate
- Label
- Menu
- MenuBar
- MenuBarItem
- MenuItem
- MenuSeparator
- Pane
- Page
- Popup
- ProgressBar
- RadioButton
- RadioDelegate
- RangeSlider
- ScrollBar
- ScrollIndicator
- ScrollView
- Slider
- SpinBox
- Switch
- SwitchDelegate
- TabBar
- TabButton
- TextArea

- TextField
- ToolBar
- ToolButton
- ToolSeparator
- ToolTip

1.2.2 List of un-styled controls

- DelayButton
- Dial
- PageIndicator
- RoundButton
- SwipeDelegate
- Tumbler

CHAPTER 2

Changelog

2.1 0.2.0

- Add TabView and TabItem to facilitate the creation of Tab based interfaces
- TabButton has a new `closable` property and a new `closeRequest` signal
- Add navigation buttons to TabView to be able to reach invisible tabs
- Fix compatibility issues with Qt 5.12

2.2 0.1.0

Initial release with an almost complete QQC2 Theme and one extra control

CHAPTER 3

Getting Started

This page will guide you through installing Absolute and using it in your application

3.1 Installation

There are 3 different ways to install and use Absolute:

1. Just drop the content of the **src** to a path on the **QML2_IMPORT_PATH** (e.g. in your qml resources)
2. Install the project with CMake, qml files will be installed in the Qt5 qml install directory

```
mkdir build && cd build
cmake ..
make
sudo make install
```

3. Use Absolute as a CMake sub-project

```
set(ABSOLUTE_QML_INSTALL_DIR "${CMAKE_BINARY_DIR}/lib/MyAwesomeProject/qml") #␣
↪optional, set this if you already have a path where you put your project's qml file.
set(ABSOLUTE_BUILD_EXAMPLES OFF) # skip building examples
set(ABSOLUTE_BUILD_TESTS OFF) # skip building tests
add_subdirectory(third-party/absolute)
```

3.2 Using Absolute Style

3.2.1 1. Enable Absolute Style in your main

To use the absolute style, you need to tell Qt that it should use the **Absolute** QQC2 Style:

You can do that by setting `QT_QUICK_CONTROLS_STYLE` environment variable or use QQuickStyle if you're working with C++.

Note: If you didn't installed Absolute in the Qt5 qml install directory, you'll have to specify an additional QML import path.

Example:

```
// Specify where qml can find the Absolute qml files
qputenv("QML2_IMPORT_PATH", ABSOLUTE_IMPORT_PATH);
// Use Absolute Quick Controls 2 Style
qputenv("QT_QUICK_CONTROLS_STYLE", "Absolute");
```

3.2.2 2. Configure Style Globally

The `Style` object (from `import Absolute.Style 0.1`) is a singleton object that holds a series of configuration settings:

- theme
- typography
- icon font

To help configure Style in a declarative way, you should use `StyleConfiguration` item.

Example:

```
import Absolute.Style 0.1

ApplicationWindow {
    id: mainWindow

    StyleConfiguration {
        // Use one of our ready made themes (see AvailableThemes) or define your own
        theme: Theme {
            dark: Palette {
                ...
            }
            light: Palette {
                ...
            }
            radius: 0
        }
        typography: Typography {
            medium: 13
        }
        iconFont: "FontAwesome"
    }
}
```

3.2.3 3. Choose to use the Dark or Light palette

By default, Absolute will always use the light palette of the configured theme. You can switch between the light and dark palette use the Material attached property.

Note: We use the **Material** attached property to customize controols instead of our own for two reasons:

- it is not possible to create an attached property in pure QML, you need C++ or Python
- the Material attached property already contains most of the properties we wanted to expose and it also has the nice feature of cascading style changes to children items.

For example, to use the dark palette of the current theme:

```
import QtQuick.Controls 2.12
import QtQuick.Controls.Material 2.12
import Absolute.Style 0.1

ApplicationWindow {
    id: mainWindow

    Material.theme: Material.Dark
}
```

3.2.4 4. Customize a specific control

To customize the appearance of a specific control, you may use the **Material** attached property as you would do if you were using the Material Style from Qt.

For example, to customize the background of a Button:

```
Button {
    text: "Blue Button"
    highlighted: true

    Material.accent: "blue"
}
```

3.2.5 5. The icon property

We use an icon font for rendering icons. Contrarily to the official Qt API; we don't render icons as images but as text.

The default icon font can be set from the StyleConfiguration. Default is **Material Icons**.

This means that the meaning of the icon properties is changed compared to what you'd do with other QQC2 styles:

- `icon.name`: specifies the glyph to use to render the icon. You may want to use our builtin `MaterialIcons` singleton object that provides easy access to all the material icons.
- `icon.source`: specifies the font family of the font to use to render the icon. Default is **Material Icons, Regular** but you can use another icon font if you want.

3.2.6 6. Additional properties

Radius

We added the `radius` property to button, pane and frame so that you can customize the radius of those elements independantly from the global radius property from the global Theme.

Warning: Using the `radius` property will break your application if you're not using Absolute Style.

CHAPTER 4

Contributing

You are here to help on Absolute? Awesome, feel welcome and read the following sections in order to know how to ask questions and how to work on something.

4.1 Reporing bugs or Wishes

Report any bugs you encountered or any wishes on our issue tracker.

If you're reporting a bug, **make sure to provide the following information**:

- Information about your **Operating system** (e.g. Windows 8.1, Mac OSX Yosemite,...). If you're on Linux, you'll need to specify the name of the distribution and the desktop environment you're using.
- The Qt Version you're using
- A **clear description** of the bug with **steps to reproduce**.
- You should use **English** to describe your issue. French is also accepted.

4.2 Setting up a development environment:

Install the following tools and libraries:

- QtCreator
- Qt5 (require QtQuickControls2 and QtGraphicalEffects)

You can run the example from withing QtCreator

4.3 Submitting a merge request

Here are the steps you need to follow to start working on Absolute and submit your work for evaluation or integration into the main project:

1. Fork the Repo on gitlab.
2. Create a feature or a bugfix branch before you start coding.
3. Use the standard Qt QML code formatting: http://doc.qt.io/qt-5/qml-codingconventions.html
4. Make sure you added a test for the new feature you've developped.
5. Push to your fork and submit a pull request to **the master branch**

CHAPTER 5

API Documentation

5.1 Absolute.Style

The `Absolute.Style` package contains all the classes used to customize the QQC2 Style.

Import statement:

```
import Absolute.Style 0.1
```

5.1.1 MaterialIcons

MaterialIcons is a singleton object that has properties for all the icons that can be used from the font.

Properties

- `icon_xxx`: readonly properties for all the icons that can be found in the Material Icons font
- `family`: the name of the Material Icons font family

5.1.2 Palette

Contains all the color properties used to render controls.

Properties

- `base`: Color used as the background color for root containers such as Window/ApplicationWindow, Drawer, Popup and Dialog.
- `contrast`: Color used as the background color of internal containers such as Pane and Frame with a very low opacity and as the foreground and border color of most controls with a very high opacity.

- `power`: Factor used to customize the amount of opacity of the contrast color. Default is 1.0.
- `primary`: Color used for the background of ToolBar and MenuBar. Default is `base`.
- `primaryContrast`: Color used by ToolButton and MenuItem text rendering. Default is `contrast`.
- `separator`: Color used for separators (dialog/popup/menu borders).
- `accent`: Color used to hihglight specific part of the control when it is checked,enabled,highlighted,...
- `info`: The color to use to highlight some important information to the user.
- `success`: The color to use to highlight a successful action to the user.
- `warning`: The color to use to show a warning the user.
- `danger`: The color to use to highlight a dangerous action to the user.

5.1.3 Style

Singleton object that can be used to access the Absolute style properties (theme, typography,...)

Properties

- `theme`: The color theme used by the style. Default value is to the Default theme.
- `typography`: Typography options (font size,...)
- `iconFont`: Icon font family used to render icons. Default is **Material Icons**

5.1.4 StyleConfiguration

Convenience item to configure Absolute Style declaratively.

Properties

The below properties are propageted to the `Style` singleton.

- `theme`: The color theme used by the style. Default value is to the Default theme.
- `typography`: Typography options (font size,...)
- `iconFont`: Icon font family used to render icons. Default is **Material Icons**

5.1.5 Theme

A theme is composed by a dark and a light palette as well as some other properties such as default radius and default elevation.

Properties

- `dark`: the dark `Palette` of the Theme.
- `light`: the light `Palette` of the Theme.
- `radius`: Radius value used in various controls. Set it to 0 to disable radius globally.

Methods

- `palette(bool isDark)`: utility function to either get the `dark` or `light` palette of the theme. Typical usage:

```
Material.accent: Style.theme.palette(Material.theme === Material.Dark).
↪info
```

5.1.6 Typography

Properties

- `smaller`: Smaller font size. Default is 9
- `small`: Small font size. Default is 11
- `medium`: Medium font size. Default is 13. *This is the size used in most controls.*
- `big`: Big font size. Default is 16
- `bigger`: Bigger font size. Default is 18

5.1.7 Absolute.Style.Themes

The `Absolute.Style.Themes` package contains the implementation of our builtin themes as well as a convenience item to list all available themes and register custom ones.

Import statement:

```
import Absolute.Style 0.1
```

AvailableThemes

Convenience object to expose the list of available themes.

Builtin themes:

- "Default"
- "Native

Properties

- `model`: A ListModel that contains all the builtin themes as well as those manually register. Can be used to feed a Menu/ComboBox.

Methods

- `byName(str: name) -> Theme`: Returns a Theme instance by name. Will return undefined if the theme does not exist.
- `register(str: name, Theme: theme)`: Registers a custom theme and make it available to the `model` and `byName`.

Default

The default Absolute theme.

Native

A native theme that tries to use the native colors using the `SystemPalette`

> **Warning:** The native theme may not work correctly under GNome if the current gtk 3 theme does not include a gtk2 theme version.

5.2 Absolute.Controls

The `Absolute.Controls` package contains some extra controls

Import statement:

```
import Absolute.Controls 0.1
```

5.2.1 FlatMenu

FlatMenu has the same API as `Menu` but allow to be placed inside the main window (Qt's `Menu` is always a popup)

5.2.2 FlatMenuItem

FlatMenuItem has the same API as `MenuItem` but allow to be placed inside a `FlatMenu`

5.2.3 TabView

TabView combines a TabBar and StackLayout to facilitate the creation of tab based interfaces.

TabView implements the Container API, you can either add `TabItem` declaratively or programmatically (using `addItem`, `insertItem` and so on)

5.2.4 TabItem

Item that can be added to a TabView.

Properties

- `string text`: Text used for the TabButton `text` property
- `Icon icon`: icon group used for the TabButton `icon` property
- `Component view`: the view component that will be inserted in the StackLayout
- `bool closable`: specifies whether tab is closable. If True, a close button will be show next to the tab button's text

- `bool autoClose`: true to automatically close the tab when the close button is pressed. Otherwise, you need to call `close()` manually.
- `bool closeAccepted`: specifies whether the close request was acccepted. The TabButton and it's corresponding view will be removed immediately when this property becomes `True`.

Signals

- `onCloseRequest`: this signal is emitted when the user clicked on the close button.

Methods

- `close()`: close this tab

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

CHAPTER 7

Indices and tables

- genindex
- modindex
- search