RJ CODE
Advance

M1

# RJ CODE MODERN UI

Custom themes, styles, forms and controls for WinForm + Source code

## PROGRAMMING TUTORIALS

Simple, easy, fast and fun learning

# Content

# 1. INTRODUCTION

Hello welcome to **this tutorial written from RJ Code Modern UI project**, template with modern and flat design. For the design, **custom forms, custom controls** and the necessary components for the **application's appearance settings** are created.

This project includes:

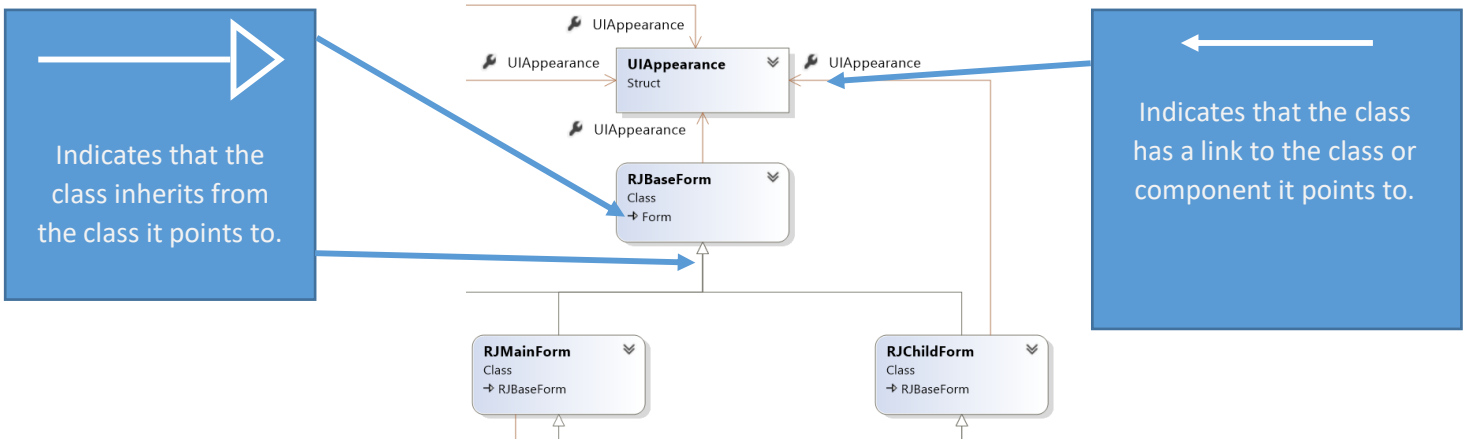
- ✓ Includes **3 Custom Base Forms**.
- ✓ Includes **18 Custom Controls**.
- ✓ Includes **2 Utilities**
- ✓ Includes **7 Components** for Appearance Setting (Including a pre-made form for application settings)
- ✓ Includes **8 pre-made Forms** (In adition, plus **7 test forms** for custom forms and controls with full demonstration of designs and styles).
- ✓ And this **complete documentation** is included which also includes the class diagrams for easy understanding.

This is a **project based on object-oriented programming** (OOP), so the pillar of **inheritance, encapsulation and polymorphism is constantly used**, in addition to including class diagrams, since it is the main component of object-oriented modeling.

**Before you start reading the documentation and looking at the source code** of the project I want you to take into account the following.

## Relationships - Class Diagram

It is very important to know the **meaning of the relationships in a class diagram**, it helps you understand and know what type of relationship the classes or components.




## This keyword

**Personally** I like to use the keyword **this**, to indicate that the field or property I access is in a base class (Superclass or Parent), I use it often in the project, therefore keep in mind that the **field or property is in a base class.**

# 2. FEATURES

- IDE: Visual Studio 2012
- Framework: .NET Framework 4.5
- Language: C # 5.0
- Platform: Windows Form
- Nuget Package 1: FontAwesome.Sharp 5.15.3

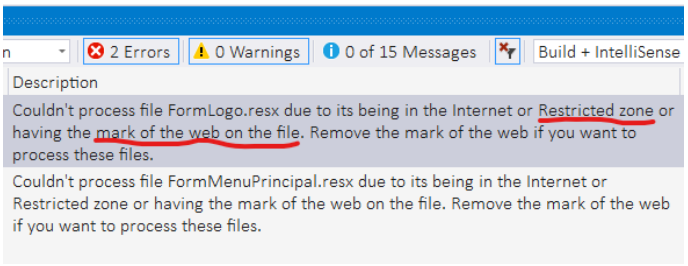
# 3. REQUIREMENTS

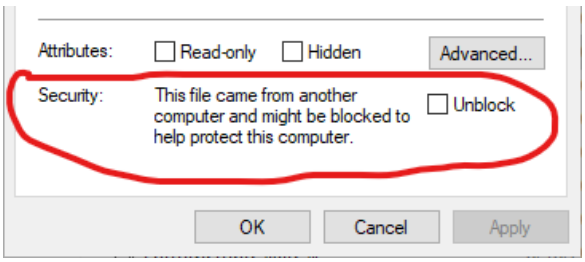
- ✓ Visual Studio 2012 or Higher (I do not recommend VS 2015 Update 1)
- ✓ .NET Framework 4.5 or Higher

# 4. SOLVE COMMON ERRORS

## 4.1. The file could not be processed - Web Mark

They often email me about the following error.




***"ERROR**: "Couldn't process file "AnyFile.resx" due to its being in the **Internet or Restricted zone** or having the **mark of the web** on the file. **Remove the mark of the web if you want to process these files."***

This is a very frequent error (**zone identifier or mark of the web**), it **happens when a Visual Studio project or any file is downloaded from the internet**, these files are marked with the zone identifier by means of alternative data streams.

### Solution:

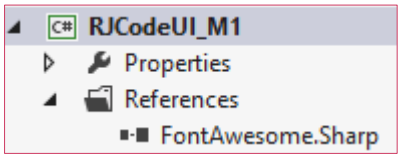
A. **Option 1:** Unlock the file **from the properties** (See image above-right), I recommend doing it in the compressed file (.rar, .zip) before unzipping any project, otherwise you will have to unlock each file of the project.

B. **Option 2:** Unlock all files in the entire project **using PowerShell**, run the following command: *Get-ChildItem -Recurse -Path 'Project folder path' | Unblock-File*

C. **Option 3:** Download and use the tool **RJ ZoneID Remover** that I created especially for these cases, it unlocks any file and / or folders en masse.

# 5. FONTAWESOME.SHARP LIBRARY

This project uses the library FontAwesome.Sharp to add icon to base forms (RJMainForm - RJChildForm) and user controls (RJComboBox - RJDatePicker), and extend the IconButton and IconPictureBox control.




This library was created by mkoertgen, It has constant updates and I hope it stays that way for a long time, you can download it from GitHub or Nuget to include it in your project (In this case it is not necessary, the library is already downloaded and integrated into the project).

**About the Author:**

- Name: Marcel Körtgen
- GitHub profile: https://github.com/mkoertgen
- Nuget Profile: https://www.nuget.org/profiles/mkoertgen
- Number of projects in Nuget: 18

**Download FontAwesome.Sharp library:**

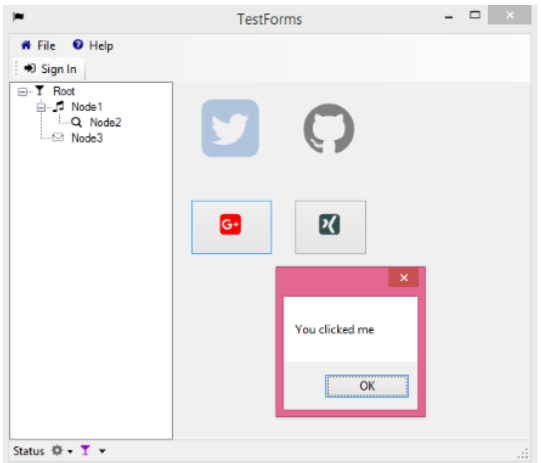
- GitHub: https://github.com/awesome-inc/FontAwesome.Sharp
- Nuget: https://www.nuget.org/packages/FontAwesome.Sharp/

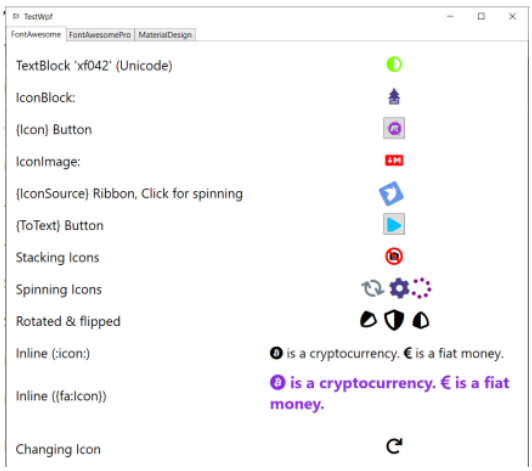
## 5.1. What is FontAwesome?

FontAwesome is a tool that allows you to **create CSS vector-based icons from a text font format**, where it is really easy to place icons with a custom color and size, I'm sure many of you have used it when creating web pages or applications web or at least have heard. Currently, FontAwesome has **1609** icons in its FREE version and **6256** icons in your PRO version. Well there are many more similar tools like; Material Icons, Captain Icons, Octicons, Typicons, Zondicons, Devicons, etc. However, none of these tools have support for desktop applications, **they can only be included in web pages** and style sheets.

## 5.2. What is FontAwesome.Sharp?

As stated above, the FontAwesome tool can only be used on web pages or web applications. Therefore, **FontAwesome.Sharp** is a **control library** that allows you to embed FontAwesome icons in your **WPF or Windows Forms desktop applications**, also if you have a FontAwesome PRO license, you can use the icons in this library without any problem.



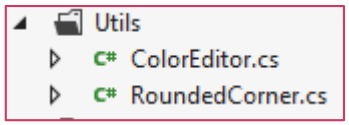



It currently has 2 controls that can be integrated into the toolbox and 4 Components that you can use through code.

- IconButton (Control)
- IconPictureBox (Control)
- IconSplitButton
- IconToolStripButton
- IconDropDownButton
- IconMenuItem

For **more information and examples** you can visit **Marcel Körtgen's repository** at GitHub, you can also see my little **video tutorials** on how to download, install and use FontAwesome.Sharp at C# or Visual basic with Windows Form.

# 6. UTILITIES

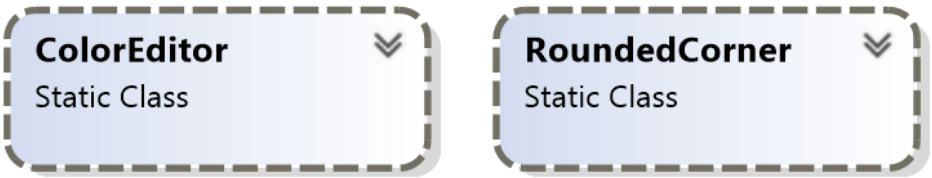



Generally, the utilities in a project are **static classes that define static functions/methods** that can be used repeatedly throughout the project or in a specific assembly to **perform operations without the need to instantiate**, this saves us a lot of time and avoids creating unnecessary objects that it can be counterproductive with performance, as I've always mentioned, it's not recommended to instantiate everywhere.

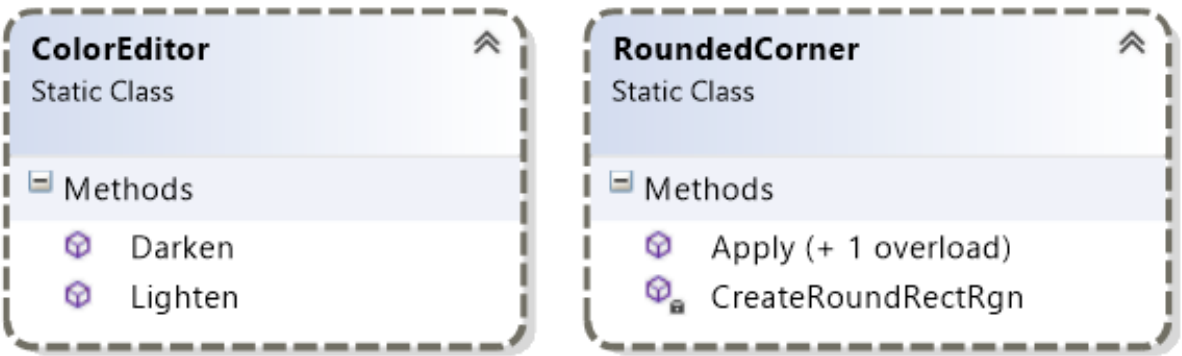
To create these classes/methods, **you can name the folder you prefer** (in my case Utils), there is no rule that indicates how to name it. Well in this case I created 2 classes of utilities; the **ColorEditor class** and the **RoundedCorner class**.

## 6.1. Class diagram

### Collapsed class diagram




### Expanded class diagram



## 6.2. ColorEditor Class

The Color Editor class (*ColorEditor.cs*) allows to **lighten or darken a specified color with a specified intensity value**, it is based on the example of Pavel vladov.

### Methods

| | |
|---|---|
| `Darken (ColourColour, ushort percentage)` | Allows you to darken a specific color with a specific darkening value, the maximum value is 100%. |
| `Lighten (ColourColour, ushort percentage)` | It allows to lighten a specific color with a specific lightening value, the maximum value is 100%. |

*Both methods return a color with the changes made.*

**How to use it?**

Using these methods is really simple, just call the static ColorEditor class, invoke one of its methods and send a color and intensity value you want, for example:

```
// Darken color 10%
this.ForeColor = ColorEditor.Darken (textColor, 10);
// Lighten color 21%
MyButton.BackColor = ColorEditor.Lighten (UIAppearance.StyleColor, 21);
```

## 6.3. RoundedCorner Class

The Rounded Corners class (*RoundedCorner.cs*) lets you create controls or forms with rounded corners.

### Methods

| | |
|---|---|
| `CreateRoundRectRgn (int nLeftRect, int nTopRect, int nRightRect, int nBottomRect, int nWidthEllipse, int nHeightEllipse)` | External method that creates a rectangle with rounded corners with a specified ellipse width and height. |
| `Apply (Control control, int radius)` | Create the region of a control with rounded corners with a specified radius. |
| `Apply (Form form, int radius)` | Create the region of a form with rounded corners with a specified radius. |

**How to use it?**

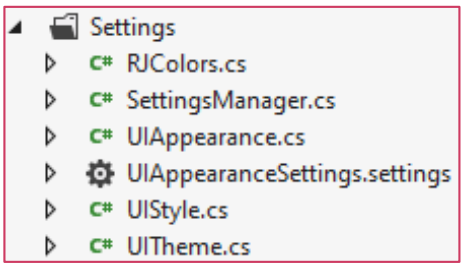
Call the RoundedCorner class, invoke the Apply method, and send a form or control instance and a radius as parameters, for example:

```
// Apply rounded corners with a radius of 15 to a control
RoundedCorner.Apply (MyControl, 15);
// Apply rounded corners with a radius of 8 to a form
RoundedCorner.Apply (MyForm, 8);
```

# 7. APPEARANCE SETTINGS

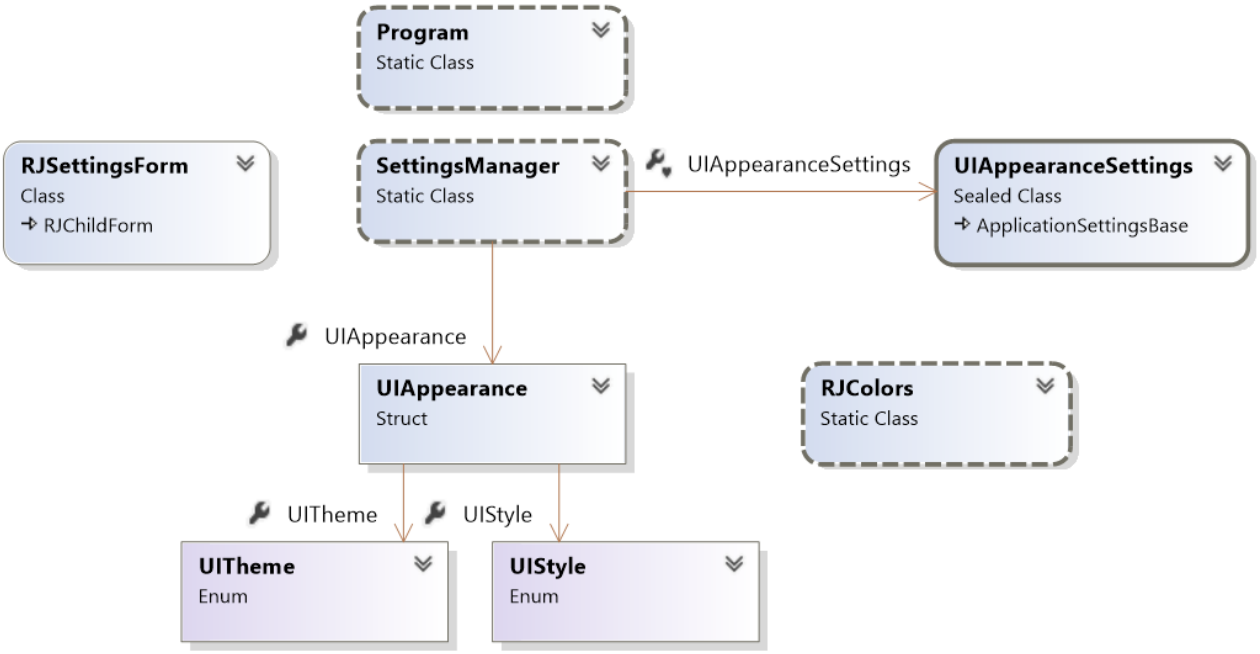
The vast majority of us like to customize the appearance of our Windows, Android, Apple or a specific application, changing the color of windows, background, icons, menu bar, title bar, etc. wherever we like and feel more comfortable.

Therefore, this project prioritizes and **integrates the function of configuring the appearance of the user interface**, allowing to **change the theme, color style, designs, border width, border color**, and among others, applying customization settings to **forms and controls.**

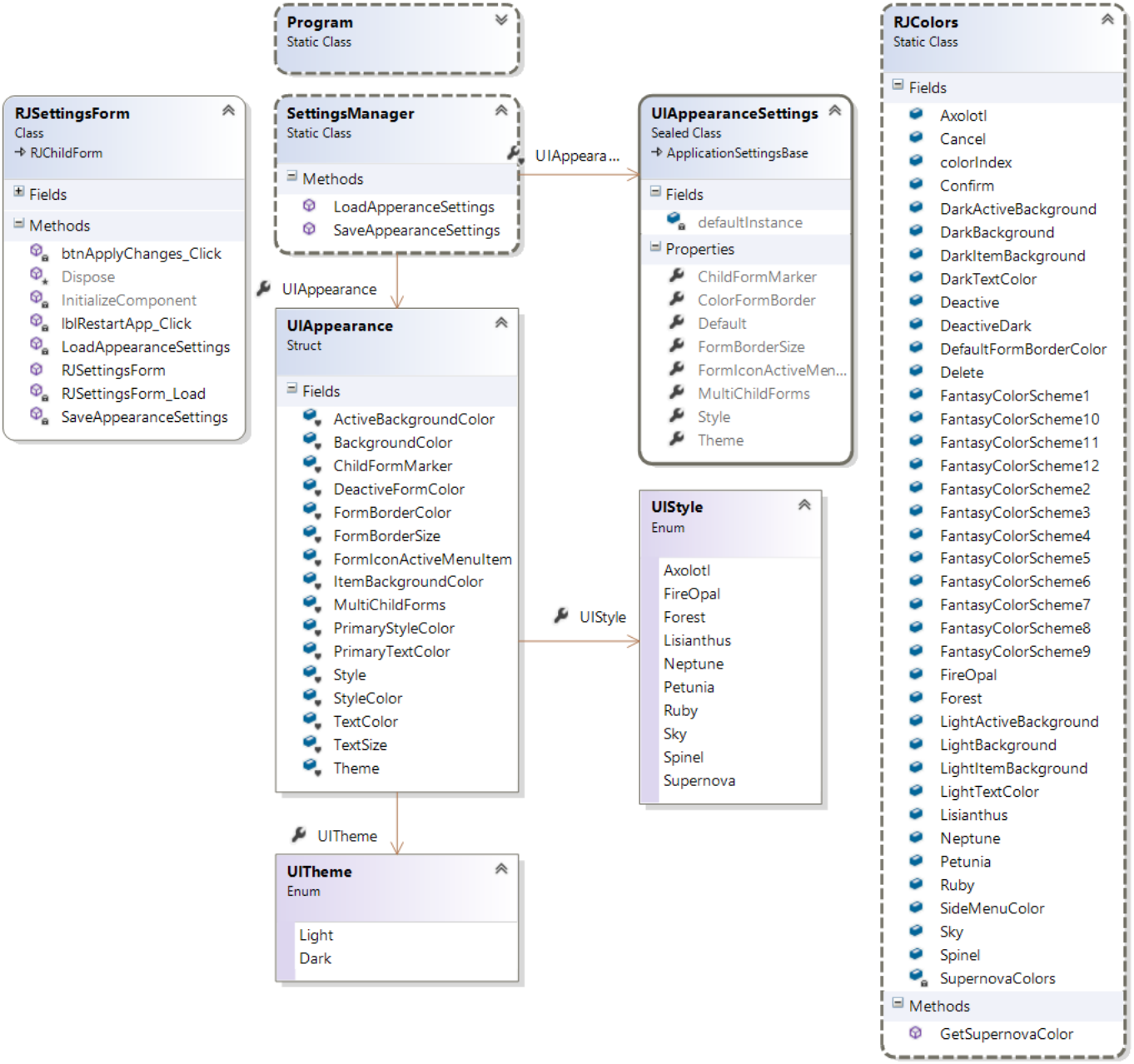



## 7.1. Class diagram

### Collapsed class diagram



## Expanded class diagram



## 7.2. Color List - RJColors Class

The static class RJColors simply defines a list of colors for the theme color, style color, text color, and others. Also implements a method to get a color for the supernova style.

### Fields

| | |
|---|---|
| `colorIndex` | Gets or sets the color index of the supernova style color list. |
| `SideMenuColor` | Gets or sets the color of the side menu. |
| `DefaultFormBorderColor` | Sets the default border color for a form. |
| `Axolotl` | Gets or sets the color of the styles. |
| `FireOpal` | |
| `Forest` | |
| `Lisianthus` | |
| `Neptune` | |
| `Petunia` | |
| `Ruby` | |
| `Sky` | |
| `Spinel` | |
| `FantasyColorScheme1` | Gets or sets the fancy color scheme. |
| `FantasyColorScheme2` | |
| `FantasyColorScheme3` | |
| `FantasyColorScheme4` | |
| `FantasyColorScheme5` | |
| `FantasyColorScheme6` | |
| `FantasyColorScheme7` | |
| `FantasyColorScheme8` | |
| `FantasyColorScheme9` | |
| `FantasyColorScheme10` | |
| `FantasyColorScheme11` | |
| `FantasyColorScheme12` | |

| | |
|---|---|
| DarkBackground | Gets or sets the color for the dark theme. |
| DarkItemBackground | |
| DarkActiveBackground | |
| DarkTextColor | |
| LightBackground | Gets or sets the color for the light theme. |
| LightItemBackground | |
| LightActiveBackground | |
| LightTextColor | |
| Delete | Gets or sets the color of actions |
| Confirmed | |
| Cancel | |
| Deactive | |
| DeactiveDark | |

## Methods

| | |
|---|---|
| GetSupernovaColor () | Get a color from the supernova style color list. |

**How to use?**

You can get a color and set to any color property of the control, for example:

```
MyButton.BackColor = RJColors.GetSupernovaColor ();
```

## 7.3. Settings file - UIAppearanceSettings

The appearance settings setting file (*UIAppearanceSettings.settings*) allows you to **store and retrieve user appearance property settings.**

### Properties

| | |
|---|---|
| `ChildFormMarker` | Gets or sets whether the child form marker will be displayed on the menu button on the side menu of the main form. |
| `ColorFormBorder` | Gets or sets whether the border of the forms will have a color, otherwise it will be displayed with a default color. |
| `Default` | Gets or sets the value of the setting file properties. |
| `FormBorderSize` | Gets or sets the border width of the forms. |
| `FormIconActiveMenuItem` | Gets or sets whether the associated form icon will be displayed in the menu item (`ToolStripMenuItem`) active. |
| `MultiChildForms` | Gets or sets whether the user can open multiple child forms within the desktop pane of the main form, otherwise only a single form can be opened. |
| `Style` | Gets or sets the style of the user interface. |
| `Theme` | Gets or sets the theme of the user interface. |

## 7.4. Themes - UITheme Enumeration

The themes enumeration (*UITheme.cs*) **defines themes for the application**.

### Fields

| | |
|---|---|
| `Dark` | The background color of forms and items appear dark. |
| `Light` | The background color of forms and items appear light in color. |

## 7.5. Styles - UIStyle Enumeration

The Styles enumeration (*UIStyle.cs*) **defines styles for the application**, a style defines an accent or appearance color for the elements of the application, for example, the title bar and border of the forms, the background, border, icon and other elements of the controls.

### Fields

| | |
|---|---|
| `Axolotl` | The title bar of the form and the appearance of the controls are dull pink. |

| | |
|---|---|
| `FireOpal` | The title bar of the form and the appearance of the controls are colored orange. |
| `Forest` | The title bar of the form the appearance of the controls have a green tone color. |
| `Lisianthus` | The title bar of the form and the appearance of the controls are colored in purple. |
| `Neptune` | The title bar of the form and the appearance of the controls are colored in a blue tone. |
| `Petunia` | The title bar of the form and the appearance of the controls are colored purple. |
| `Ruby` | The title bar of the form and the appearance of the controls are colored in red. |
| `Sky` | The title bar of the form and the appearance of the controls have a light blue color. |
| `Spinel` | The title bar of the form and the appearance of the controls are colored light pink. |
| `Supernova` | The title bar of the form has a similar color to the theme, and the appearance of some main controls can be any color from the supernova style color list. |

To get a preview of the exact color of each style, see point 7.2 - Color list

## 7.6. User Interface Appearance - UIAppearance Structure

The User Interface Appearance structure (*UIAppearace.cs*) **stores appearance settings and other appearance settings** (that is, they are not stored in the setting file, but are set from it, such as: color of primary text, normal text color, primary style color, normal style color, background color, among others, for more information see next point (7.7)) for the appearance of forms and controls at runtime.

### Fields

| | |
|---|---|
| `ActiveBackgroundColor` | Gets or sets the background color of the application in its active or highlighted state. |
| `BackgroundColor` | Gets or sets the background color of the application. |
| `DeactiveFormColor` | Gets or sets the color of the form's title bar when it is in the deactivated state or loses focus. |
| `ChildFormMarker` | Gets or sets whether the child form marker will be displayed on the menu button on the side menu of the main form. |

| | |
|---|---|
| FormBorderColor | Gets or sets the border color for forms. |
| FormBorderSize | Gets or sets the border width of the forms. |
| FormIconActiveMenuItem | The title bar of the form and the appearance of the controls are colored in red. |
| ItemBackgroundColor | The title bar of the form and the appearance of the controls have a light blue color. |
| MultiChildForms | Gets or sets whether the user can open multiple child forms within the desktop pane of the main form, otherwise only a single form can be opened. |
| PrimaryStyleColor | The title bar of the form has a similar color to the theme, and the appearance of some main controls can be any color from the supernova style color list. |
| PrimaryTextColor | |
| Style | Gets or sets the style of the application's user interface. |
| StyleColor | Gets or sets the style color for the application elements (Title bars and controls appearance) |
| TextColor | Gets or sets the text color of the user interface, mainly applied to paragraph type text for example: Labels, RadioButton, TextBoxs. |
| TextSize | Gets or sets the text size, applies to most of the custom controls when they are added to the form for the first time, then it can be changed from the properties, except the RJLabel control with Normal style. |
| Theme | Gets or sets the theme of the application's user interface. |

## 7.7.Setting Manager - SettingsManager Class

The setting manager class is responsible for saving the appearance settings in the setting file and setting all the setting settings in the UI appearance structure.

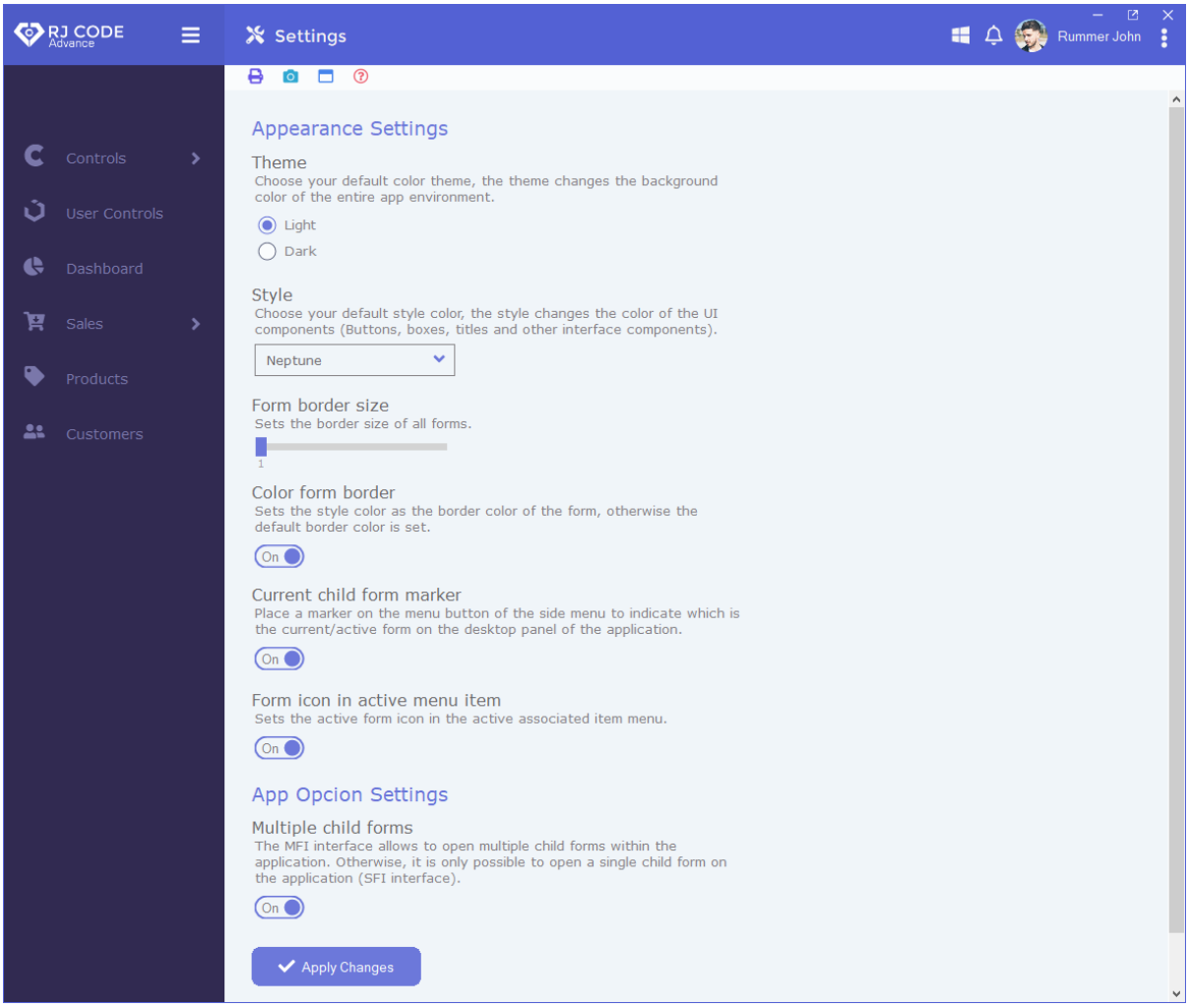
### Methods

| | |
|---|---|
| SaveAppearanceSettings (<br>int theme,<br>int style,<br>int formBorderSize,<br>bool colorFormBorder,<br>bool childFormMarker,<br>bool formIconActiveMenuItem,<br>bool multiChildForms) | **Save setting data** appearance in the setting file (*UIAppearanceSettings.settings*) permanently. |

| | |
|---|---|
| `LoadApperanceSettings ()` | It is responsible for **obtaining the appearance settings** data from the setting file, and **set the values of the User Interface Appearance structure**, such as: setting the primary and normal style color, setting or not the border color of the form, set the primary and normal text color, and enter others, all according to the established theme. |

## 7.8. Settings Form - RJFormSettings Class

The setting form is responsible for displaying the current setting, making changes and saving the modified setting in the setting file *(UIAppearanceSettings.settings).*




### Methods

| | |
|---|---|
| `LoadApperanceSettings ()` | It is responsible for obtaining the data of the current setting of the User Interface Appearance structure (UIAppearance.cs) and displaying it in the form interface, see previous image. |

| | |
|---|---|
| `SaveAppearanceSettings()` | **Save the changes made** in the setting file (UIAppearanceSettings.settings) via the setting manager. |
| `RJSettingsForm_Load ( object sender, EventArgs and)` | Invokes the LoadApperanceSettings () method to load the current appearance settings. |
| `btnApplyChanges_Click ( object sender, EventArgs and)` | Invokes the SaveAppearanceSettings () method to save the changes. |
| `lblRestartApp_Click ( object sender, EventArgs and)` | Restart the application to see the changes. |

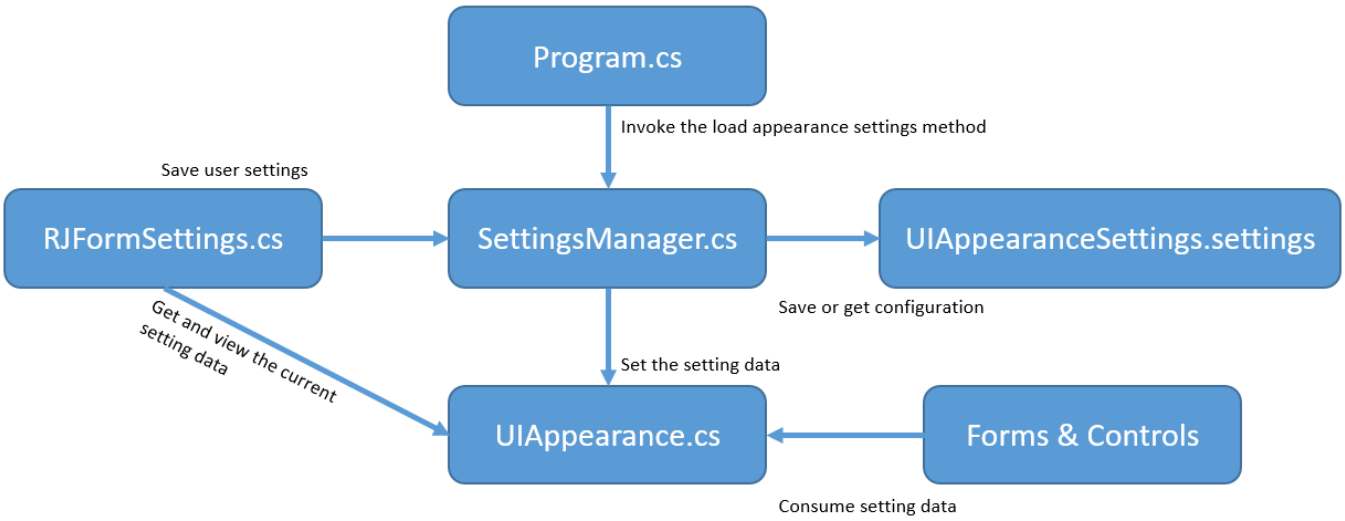
## 7.9. Program class

The Program class simply takes care of invoking the LoadAppearanceSettings () method of the setting manager class to load the data from the setting file into the UI appearance structure before displaying a form.

```
static void Main ()
{
 Settings.SettingsManager.LoadApperanceSettings ();
...
```

## 7.10. Conclusions

The operation of the appearance settings is simple:



# 8. CUSTOM CONTROLS

You can create custom controls by extending (Inheritance) an existing .NET Framework control. It can be done in 3 ways: Extended, Custom, or Composite.

## Extended controls

An extended control is simply when any existing Windows Forms control is inherited. This method allows you to preserve all the functionality of the control, and then extend it by adding properties, methods and optionally overriding the Paint event to fully extend or redraw the appearance of the control.

```
public class MyCustomButton : Button
```

## Custom controls

You can create a control from scratch by inheriting the Control class, this class provides all the basic functionality for the controls, however it does not include the graphical interface, so it is necessary to draw the control and add the necessary functions / properties, this requires a lot of effort .

```
public class MyZoomControl : Control
```

## Composite Controls or User Control

A compound control is created through multiple existing Windows Form controls, for this you can add a UserControl from the designer and add controls, or you can inherit the UserControl class and add controls through code, the constituent controls of the user control retain all their functionality where you can expose and bind properties, as well as associate or create the necessary events.

The UserControl class is basically a container for other controls, I recommend using this method when it is really necessary and as a last option as it is heavier than an extended or custom control.

```
public class MyImageSlider : UserControl
```

For more information and recommendations you can click here.

Well, in this project I created a total of **18 custom controls, 3** of them are **user controls** (created by several existing controls), **14 of them are extended controls** (Inherits from a single existing control), for this case the documentation is divided into 5 categories so as not to get entangled with so many classes in the diagrams, to be able to visualize it and understand it better, finally **1 Component** (The components are not controls, since they do not have a graphical interface and it is not a secondary element of the form, but it can be treated as one are already integrated into the visual studio toolbox and can be dragged into the forms).

NOTE:

***The appearance color of the controls is set by the appearance settings**, however, the appearance color can be customized by setting the **Customizable property** to true (Customizable = true), or the **Design property** to customizable (Desing = Customizable) according to the case.*

## A) Extended controls

**-Common controls**

1. RJ Button
2. RJ CheckBox
3. RJ Label
4. RJ RadioButton

**-Controls containers**

5. RJ ImageColorOverlay
6. RJ Panel

**-Data controls**

7. RJ DataDridView
8. RJ Chart

**-Menu controls**

9. RJ DropdownMenu
10. RJ MenuButton
11. RJ MenuIcon

**-Special controls**

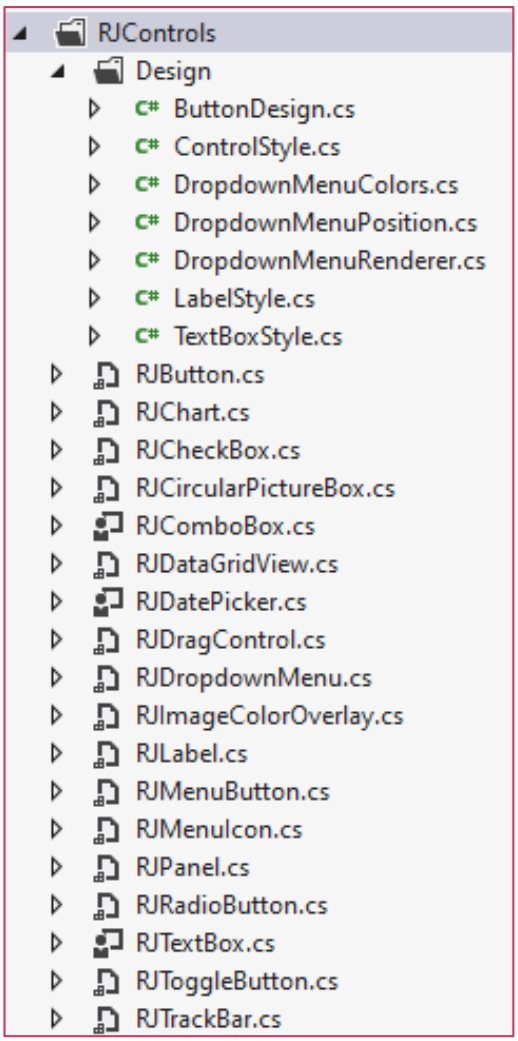
12. RJ ToogleButton
13. RJ TrackBar
14. RJ CircularPictureBox

## B) Compound Controls or User Controls

15. RJ Combo Box
16. RJ Date Picker
17. RJ Text Box

## C) Components

18. RJ DragControl



## 8.1. Control Styles - ControlStyle Enumeration

The Control Styles enumeration (ControlStyle.cs) **defines appearance styles** for most custom controls in the project, for example: buttons, combo boxes, date pickers, check boxes, and more.

| | |
|---|---|
| `Glass` | The background color of the control is **transparent** and with a colored border, in this style you can change the border size. |
| `Solid` | The background color of the control is a **solid color** without a border, this style allows you to apply rounded corners to the control, for this it uses the RoundedCorner class of the utilities. |

## 8.2. Extended controls

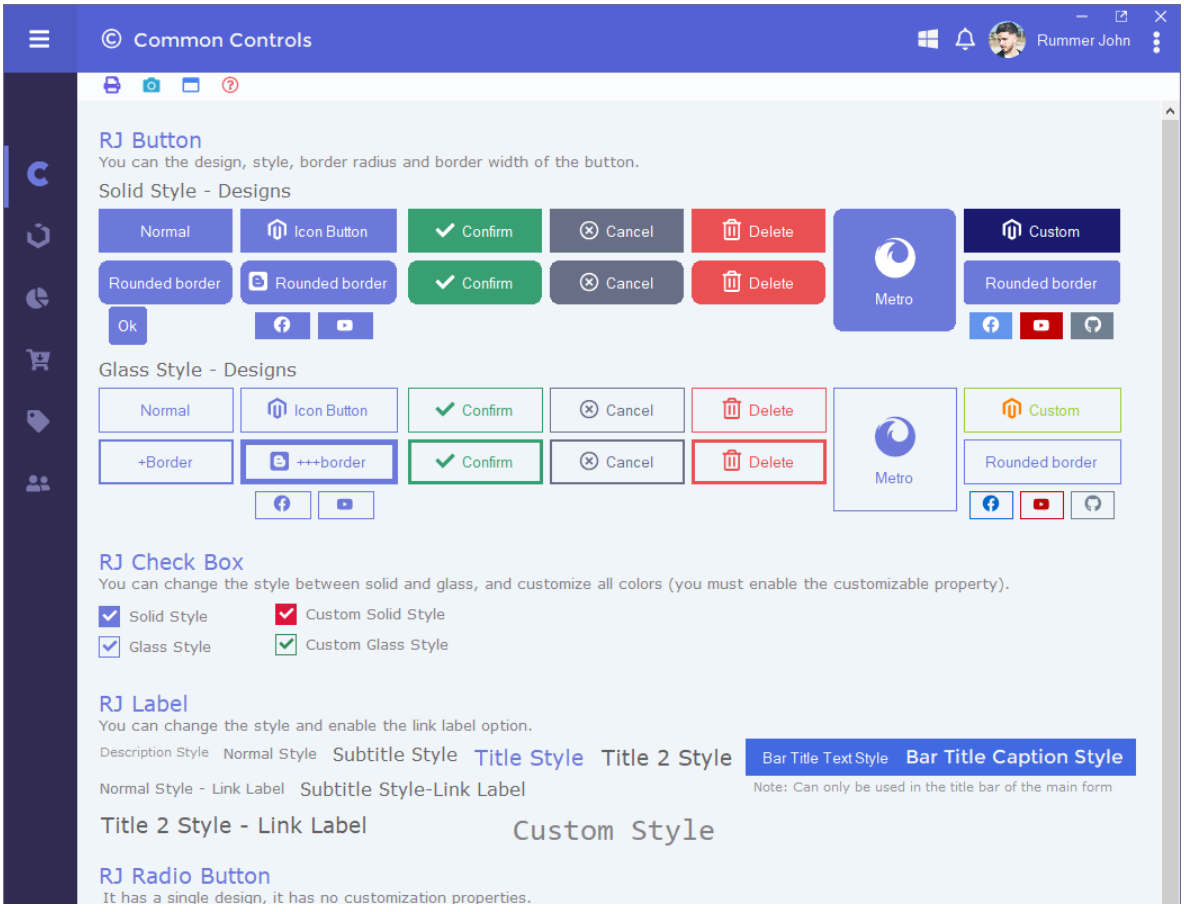
As I said before, an extended control is when it **inherits from any other existing Windows Forms control**, **preserves all the functionality** of the control, and then extends it by adding properties, methods, In addition to **override the Paint event to expand or completely redraw** the appearance of control.

**The appearance color of the controls is set by the application's appearance settings,** however, the appearance color can be customized by setting the **Customizable property**.

In this project there are **14 extended controls**, divided into **5 categories**:

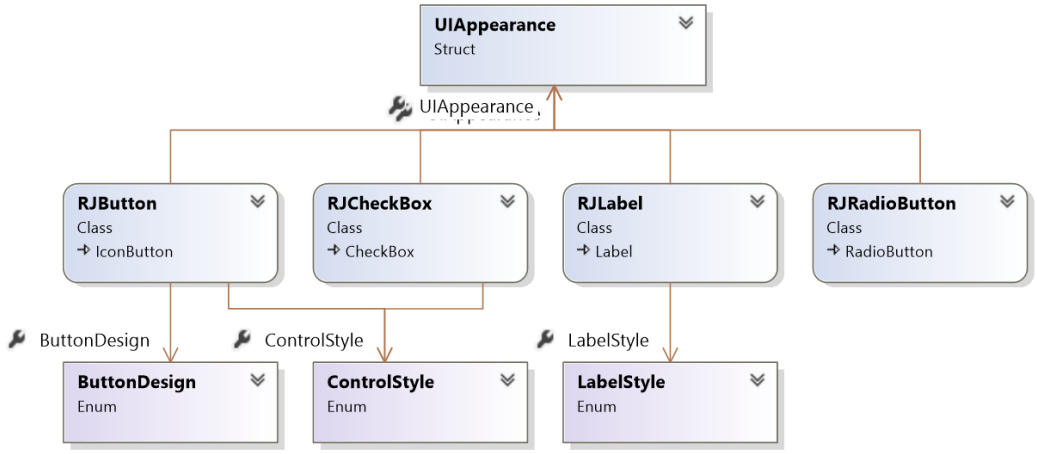
### 8.2.1. Common controls

They are common controls and the most used in any form, for example: **TextBox, Label, RadioButton and CheckBox**. Below is a **screenshot** of these controls with all the available designs and styles.

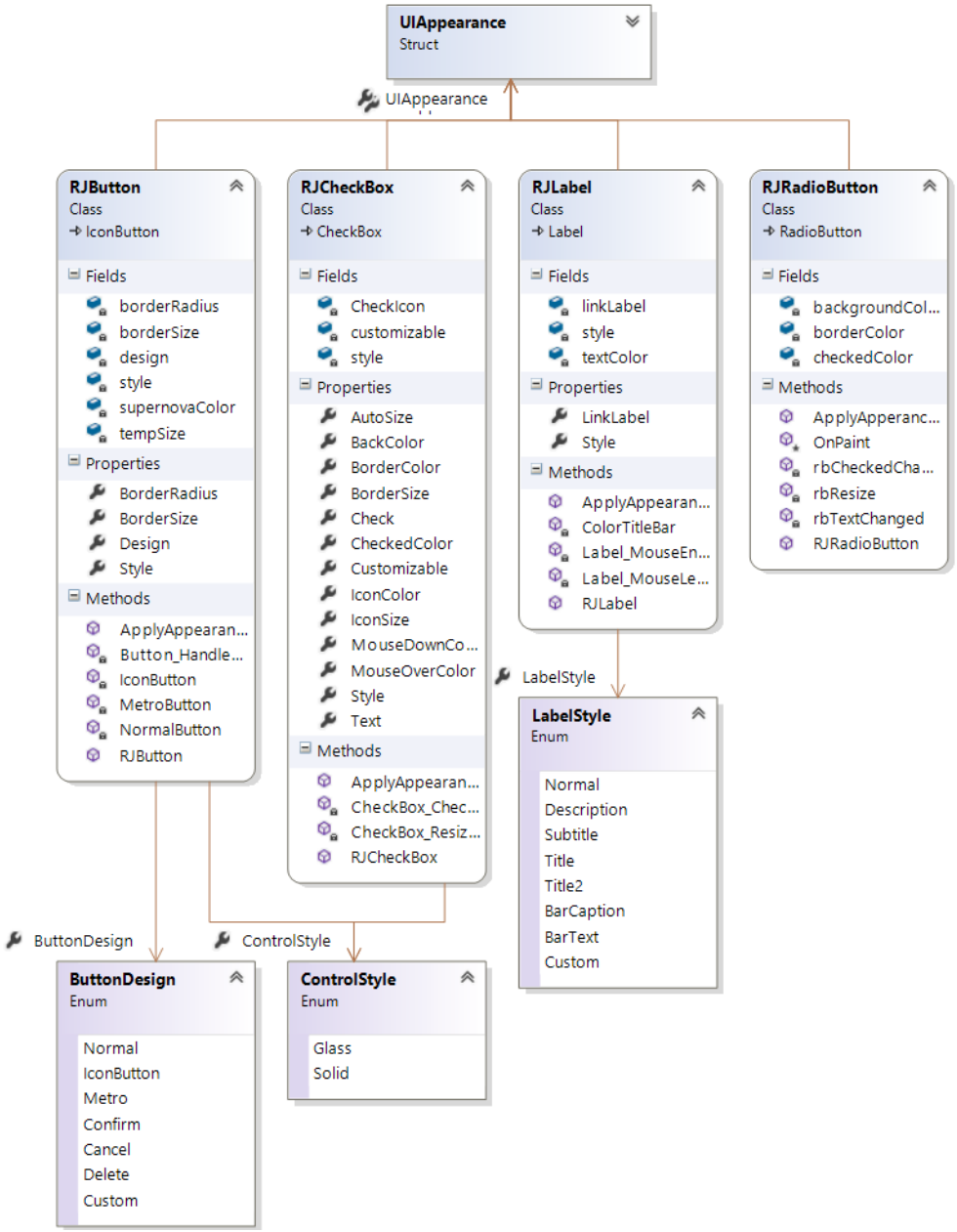


### 8.2.1.1. Class diagram

## Collapsed class diagram




## Expanded class diagram



#### *8.2.1.2. RJ Button class*

This class **inherits** from the class **IconButton** from the **FontAwesome.Sharp** library, to it, IconButton **inherits** from the **Button** class of the **Windows.Forms** library.

The RJButton control implements **4 main appearance properties**:

- ✓ Allows you to change the **button style** between **Glassy or Solid**.
- ✓ Allows you to change the **button design** between **Normal, Icon Button, Metro, Confirm, Cancel, Delete, or Custom.**
- ✓ Allows you to set **rounded corners to the button** (Only for solid style).
- ✓ Allows you to change the **button border size** (Only for glassy style)

### Properties

| | |
|---|---|
| `BorderRadius` | Gets or sets the radius of the border to apply rounded corners to the button. |
| `BorderSize` | Gets or sets the border size of the button. |
| `Design` | Gets or sets the button design (Normal, Icon Button, Subway, Confirm, Cancel, Delete, or Custom). |
| `Style` | Gets or sets the style of the button (glassy or solid). |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `Button_HandleCreated ( object sender, EventArgs and)` | Invokes the ApplyAppearanceSettings method to apply the appearance settings. |
| `IconButton ()` | Sets the properties required for the icon button design. |
| `MetroButton ()` | Sets the properties required for the button's metro design. |
| `NormalButton ()` | Sets the properties required for the normal button design. |

### Button Design Enumeration

The button designs enumeration (*ButtonDesing.cs*) **defines the different appearance designs** for the button, for example, Metro, confirm, normal, and others described below.

**Fields**

| | |
|---|---|
| `Normal` | The button has a **common, flat design**, generally the appearance color is the **same color as the application style color** set by the appearance settings. |
| `IconButton` | The button has a **flat and iconic design**, generally the appearance color is the **same color as the application style color** set by the appearance settings. |

| | |
|---|---|
| `Metro` | The button has a **similar design to the Windows 8 start menu** buttons, generally the appearance color is the **same color as the application style color** set by the appearance settings. |
| `Confirm` | The button has an **icon and flat design**, the **appearance color is green**, set in the RJColors color list. |
| `Cancel` | The button has a **flat and iconic design**, the **appearance color is dark gray**, set in the RJColors color list. |
| `Delete` | The button has a **flat and iconic design**, the **appearance color is red**, set in the RJColors color list. |
| `Custom` | The button has an **icon and flat or normal design**, according to its previous design. With this option you can customize both the design and colors of the button (In this mode the application's **appearance settings are not applied**). |

#### *8.2.1.3. RJ CheckBox class*

This class **inherits** from the **CheckBox** class in the **Windows.Forms** library.

It implements many customization properties, such as changing the style of the control and customizing the appearance color, for example, the background color, border, icon, verified status, etc. This control does not support having text.

### Properties

| | |
|---|---|
| `BackColor` | Gets or sets the background color. |
| `BorderColor` | Gets or sets the border color. |
| `BorderSize` | Gets or sets the border size. |
| `Check` | Gets or sets a value that indicates whether the checkbox is checked or not. |
| `CheckedColor` | Gets or sets the background or border color when the button is checked. |
| `Customizable` | Gets or sets whether the control's appearance colors are customizable, otherwise the appearance color is set by the appearance settings. |
| `IconColor` | Gets or sets the color of the check icon. |
| `IconSize` | Gets or sets the size of the check icon. |
| `MouseDownColor` | Gets or sets the background color when the control is clicked. |
| `MouseOverColor` | Gets or sets the background color when the mouse passes over the control. |

| | |
|---|---|
| `Style` | Gets or sets the appearance style. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `CheckBox_CheckChanged ( object sender, EventArgs and)` | It is responsible for setting or removing the check icon when the control changes state or value. |
| `CheckBox_Resized ( object sender, EventArgs and)` | It takes care of keeping the set size or setting a fixed size when the AutoSize property is set to true. |

#### *8.2.1.4. RJ Label class*

This class **inherits** from the **Label** class of the **Windows.Forms** library.

This control implements **two appearance properties** that allow you to **change the style** (Normal, Title, Description, etc.) and if it is a **link label**.

### Properties

| | |
|---|---|
| `LinkLabel` | Gets or sets if the label is a link (If true the label changes pointer and text color when the mouse passes over it) |
| `Style` | Gets or sets the appearance style. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `Label_MouseEnter (object sender, EventArgs and)` | It is responsible for highlighting the color of the text if the link type label. |
| `Label_MouseLeave (object sender, EventArgs and)` | It is responsible for reset the original text color if the label is a link type. |

#### *8.2.1.5. RJ RadioButton Class*

This class **inherits** from the **RadioButton** class in the **Windows.Forms** library.

This control **overrides the paint event completely** and a **new radio button design is drawn** with **the colors assigned in the appearance settings.**

### Fields

| | |
|---|---|
| `backgroundColor` | Gets or sets the background color. |
| `borderColor` | Gets or sets the border color. |

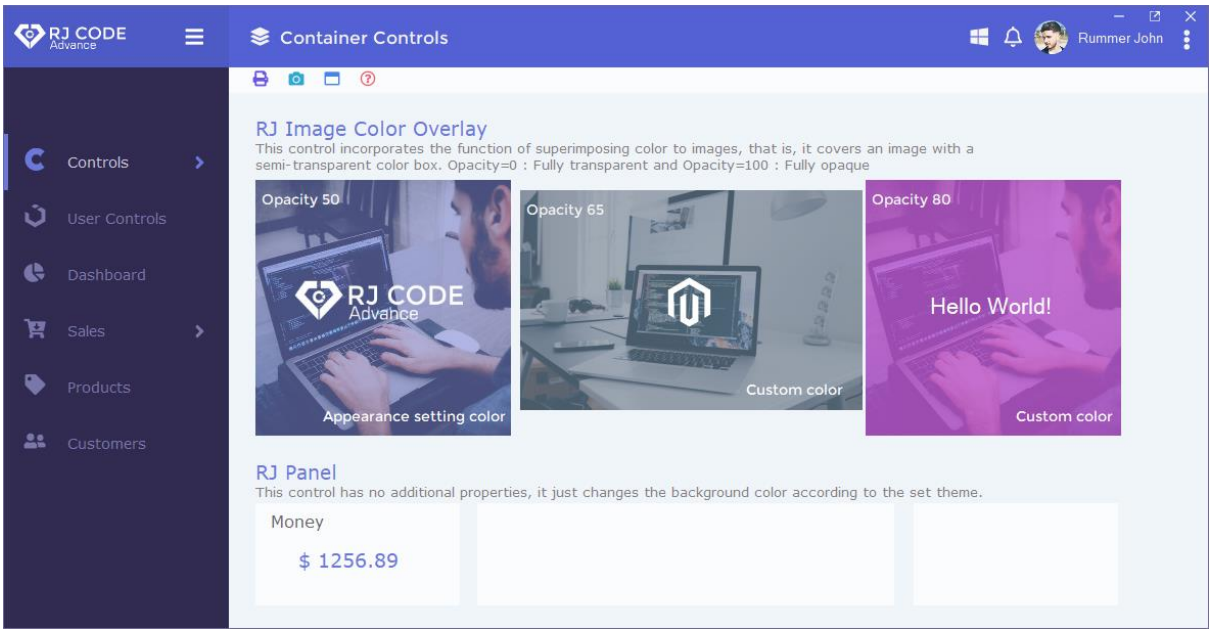
| | |
|---|---|
| `checkedColor` | Gets or sets the color of the selector point in its checked state. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `OnPaint (PaintEventArgs and)` | It is responsible for drawing the graphical interface of the control from scratch and the text. |
| `rbCheckedChanged (object sender, EventArgs and)` | It is responsible for redrawing the graphical interface when the control changes value: checked state or unchecked state. |
| `rbResize (object sender, EventArgs and)` | It is responsible for calculating and set the appropriate size of the control each time a change in size occurs. |
| `rbTextChanged (object sender, EventArgs and)` | It takes care of redrawing the text when the text in the control changes. |

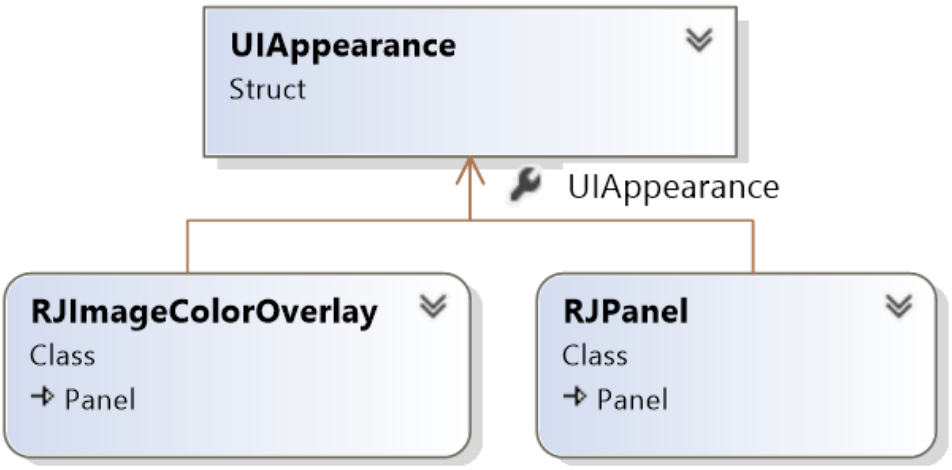
## 8.2.2. Container controls

Container controls are those controls **that can have other controls inside them**, for example in the Windows Forms library we have: GroupBox, Panel, TabControl and among others. **Two container controls are available in this project.**

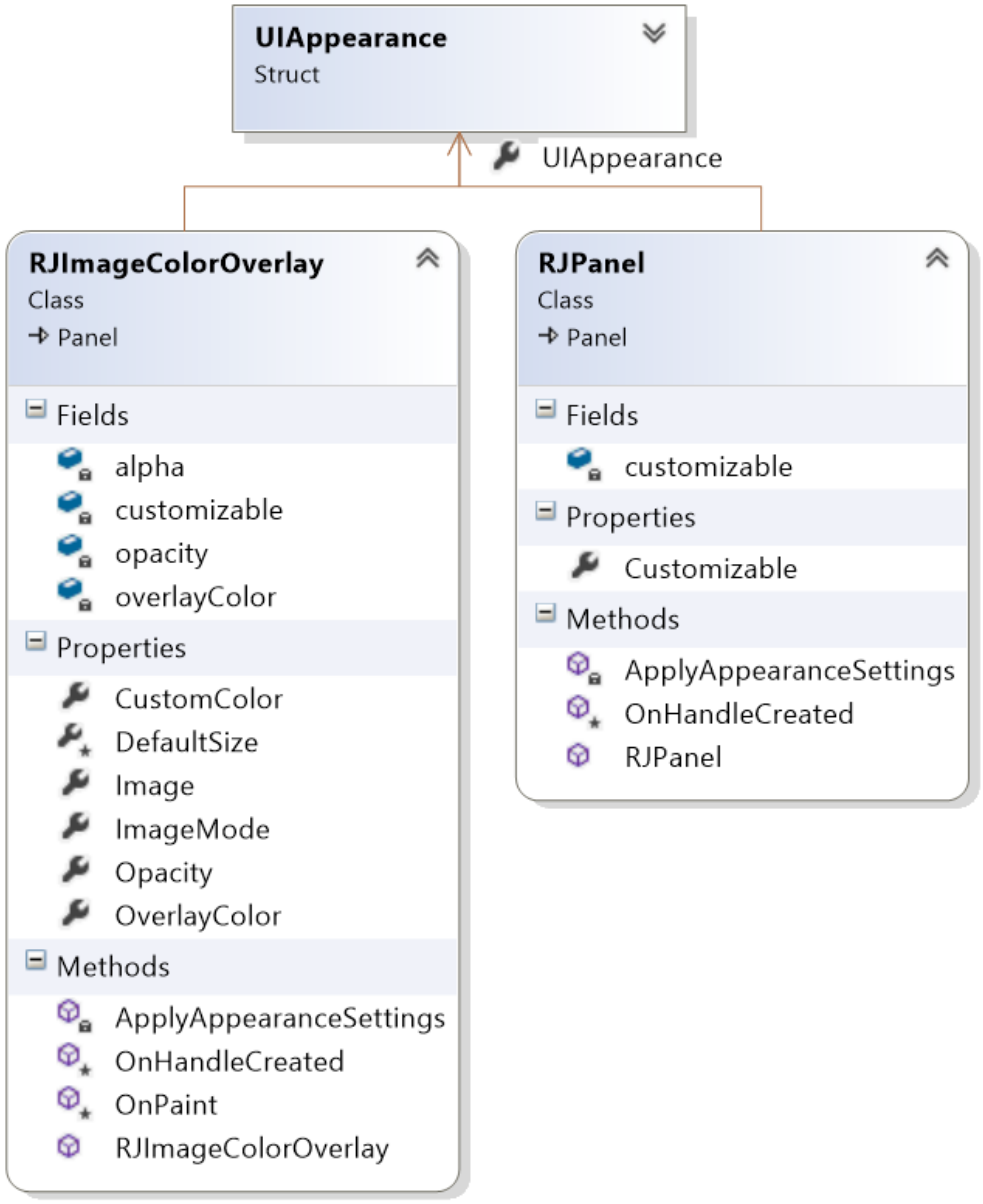



### *8.2.2.1. Class diagram*

#### Collapsed class diagram




Expanded class diagram



#### 8.2.2.2. RJ ImageColorOverlay class

This class **inherits** from the **Panel** class of the **Windows.Forms** library.

This control incorporates the function of **overlay color to an image**, that is, it covers an image with a **semi-transparent color** box, you can control the intensity of transparency, where **opacity 0 is completely transparent and opacity 100 is completely opaque or solid.** In addition, **it allows you to add other controls inside** (As shown in the screenshot).

### Properties

| | |
|---|---|
| `Customizable` | Gets or sets whether the control's appearance colors are customizable, otherwise the appearance color is set by the appearance settings. |
| `Image` | Gets or sets the image. |
| `ImageMode` | Gets or sets the image design (Centered, Stretch, Tile, or focused). |
| `Opacity` | Gets or sets the intensity of transparency. |
| `OverlayColor` | Gets or sets the overlay color. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `OnPaint (PaintEventArgs and)` | It is responsible for drawing the semi-transparent color box. |
| `OnHandleCreated ( EventArgs and)` | Invokes the ApplyAppearanceSettings method. |

#### 8.2.2.3. RJ Panel Class

This class **inherits** from the **Panel** class of the **Windows.Forms** library.

This control has no additional customization properties, it simply **sets the background color according to the theme** set by the appearance settings.

### Properties

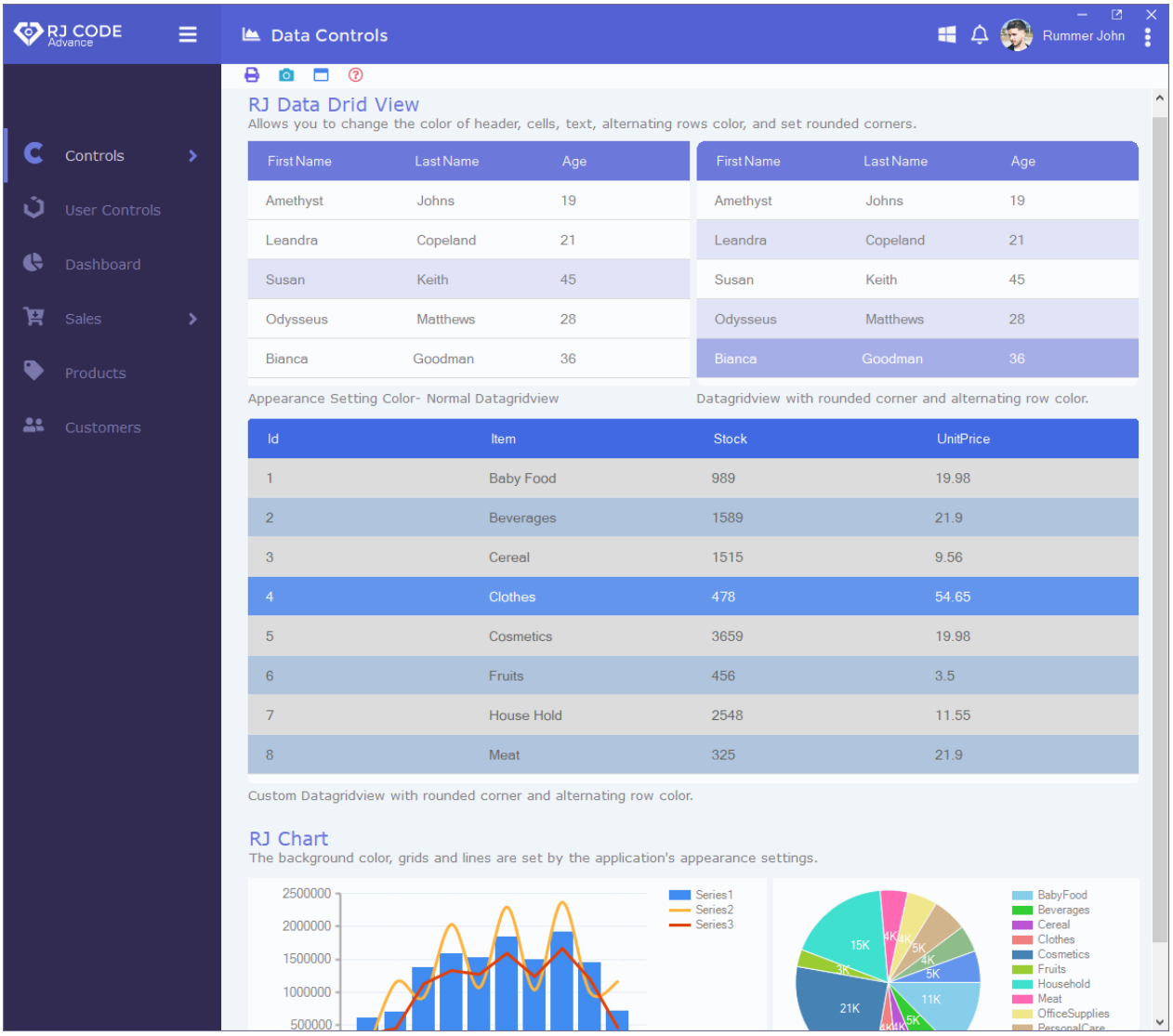
| | |
|---|---|
| `Customizable` | Gets or sets whether the background color of the control is customizable, otherwise the background color is set by the appearance settings. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `OnHandleCreated ( EventArgs and)` | Invokes the ApplyAppearanceSettings method. |

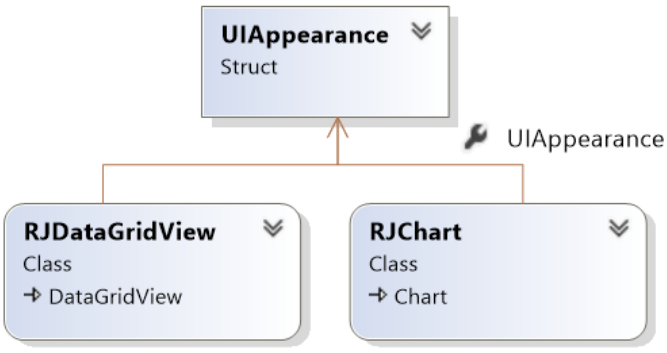
### 8.2.3. Data controls

This type of control allows you to **represent data in the graphical interface from a data source.**

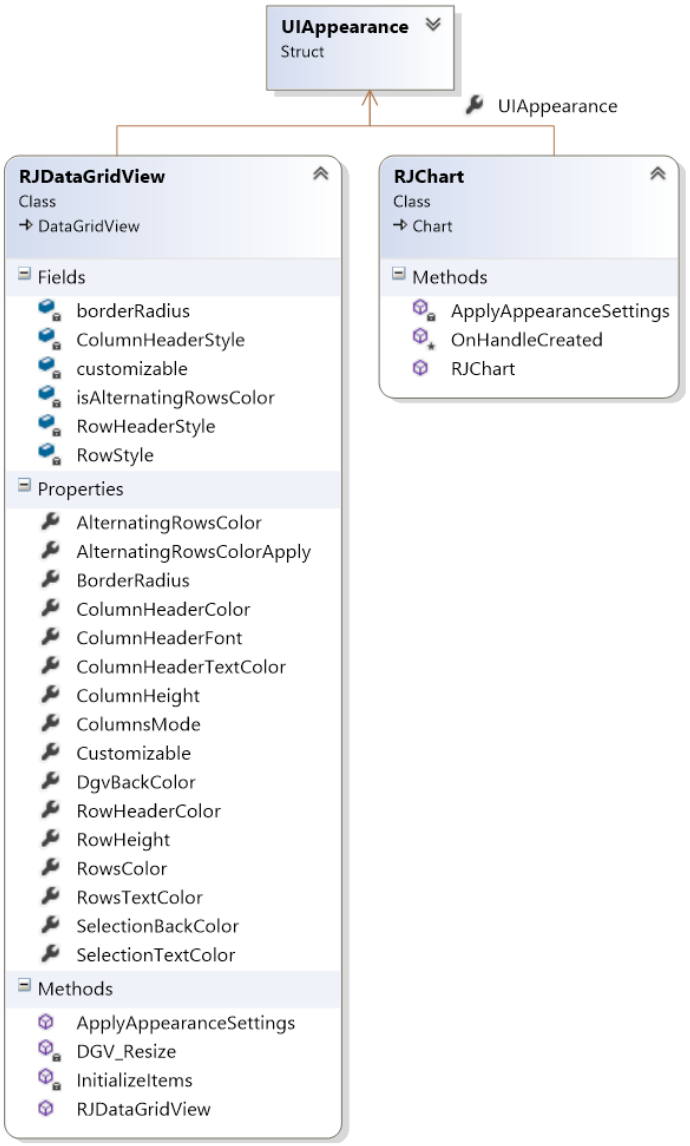



#### *8.2.3.1. Class diagram*

Collapsed class diagram




Expanded class diagram




#### *8.2.3.2. RJ DataGridView class*

This class **inherits** from the **Datagridview** class in the **Windows.Forms** library.

This control implements different **customization properties**, such as: **changing the color** of the header, rows, grids, background and among others, also being able to enable the **color of alternate rows** and setting a color for it. In addition to being able to apply **rounded corners** to the control. By default these colors are set by the appearance settings, if you want to change them set the Customizable property to true.

### Properties

| | |
|---|---|
| `AlternatingRowsColor` | Gets or sets the color for the alternate rows. |
| `AlternatingRowsColorApply` | Gets or sets whether or not to apply the alternating row color. |
| `EdgeRadius` | Gets or sets the radius for rounded corners. |
| `ColumnHeaderColor` | Gets or sets the color of the column head. |

| | |
|---|---|
| ColumnHeaderFont | Gets or sets the font for the column headings. |
| ColumnHeaderTextColor | Gets or sets the text color of the header. |
| ColumnHeaderHeight | Gets or sets the height of the column head. |
| ColumnsMode | Gets or sets the auto-size mode for columns. |
| Customizable | Gets or sets whether the background color of the control is customizable, otherwise the background color is set by the appearance settings. |
| DgvBackColor | Gets or sets the background color. |
| RowHeaderColor | Gets or sets the background color of the row header. |
| RowsColor | Gets or sets the color of the rows. |
| RowsTextColor | Gets or sets the text color of the rows. |
| SelectionBackColor | Gets or sets the background color of the selected row. |
| SelectionTextColor | Gets or sets the text color for the selected row. |

### Methods

| | |
|---|---|
| ApplyAppearanceSettings () | Sets the appearance settings to the properties of the control. |
| DGV_Resize (object sender, EventArgs and) | It is responsible for applying the appearance settings at runtime and validating the minimum height of the column header. |

#### *8.2.3.3. RJ Chart Class*

This class **inherits** from the **Chart** class of the **Windows.Forms** library.

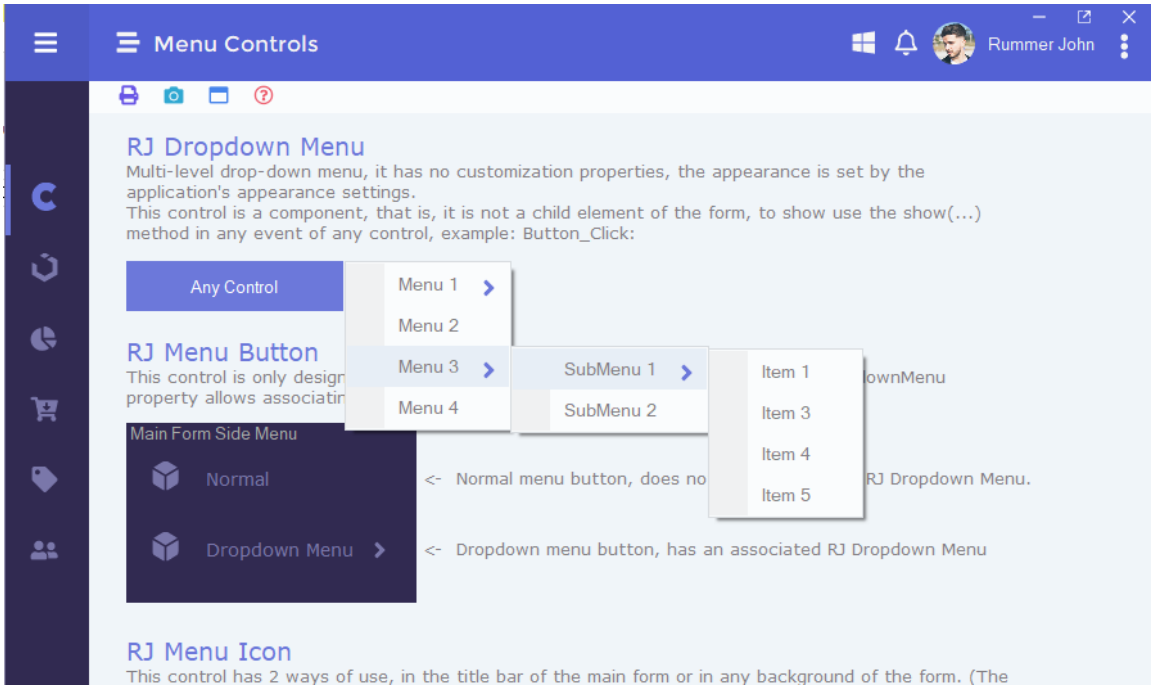
This control does not implement appearance customization properties, **the background color, grids, lines and text are implemented by the appearance settings**. However, you can modify the other original properties of the Chart control, such as **changing the palette color, changing the size of the lines,** and among others.

### Methods

| | |
|---|---|
| ApplyAppearanceSettings () | Sets the appearance settings to the properties of the control. |
| OnHandleCreated ( EventArgs and) | It is responsible for applying appearance settings at run or design time. |

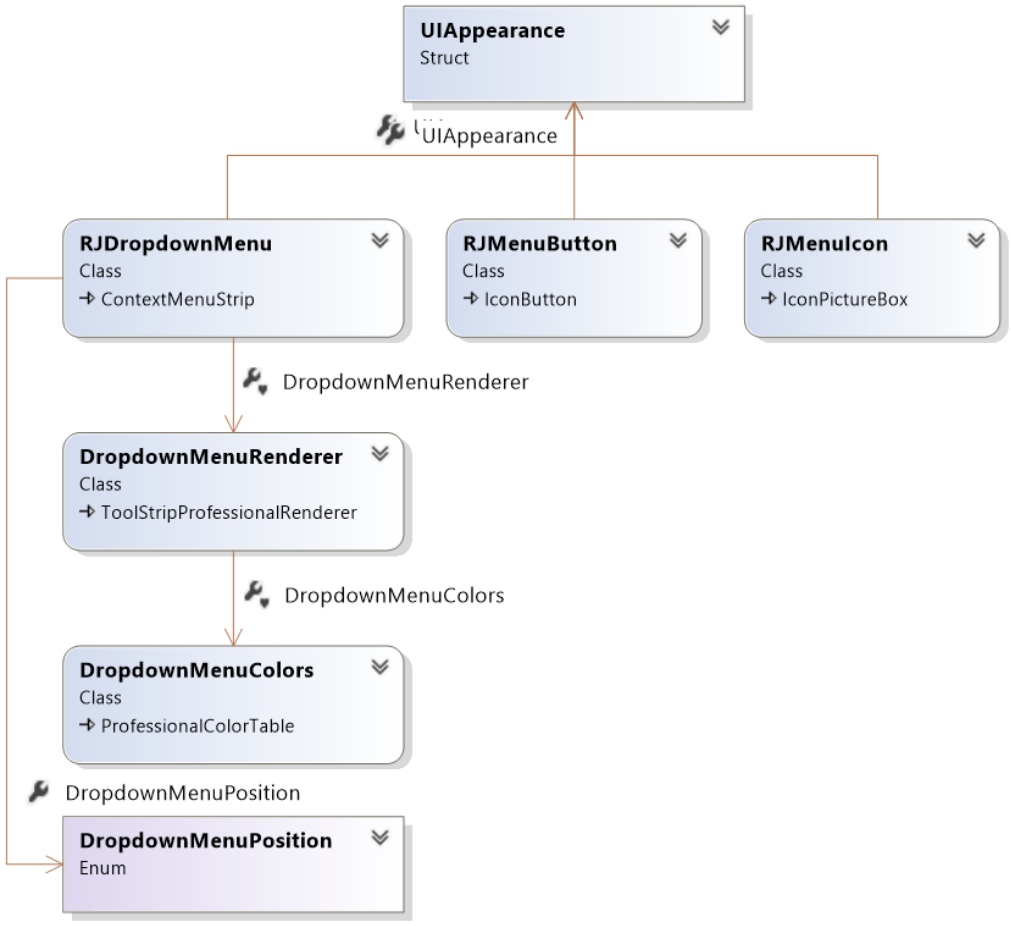
### 8.2.4. Menu controls

The menu controls **allow the hierarchical organization of elements in a multilevel** way, and / or can be used in the **side menu and in the title bar of the main form.**

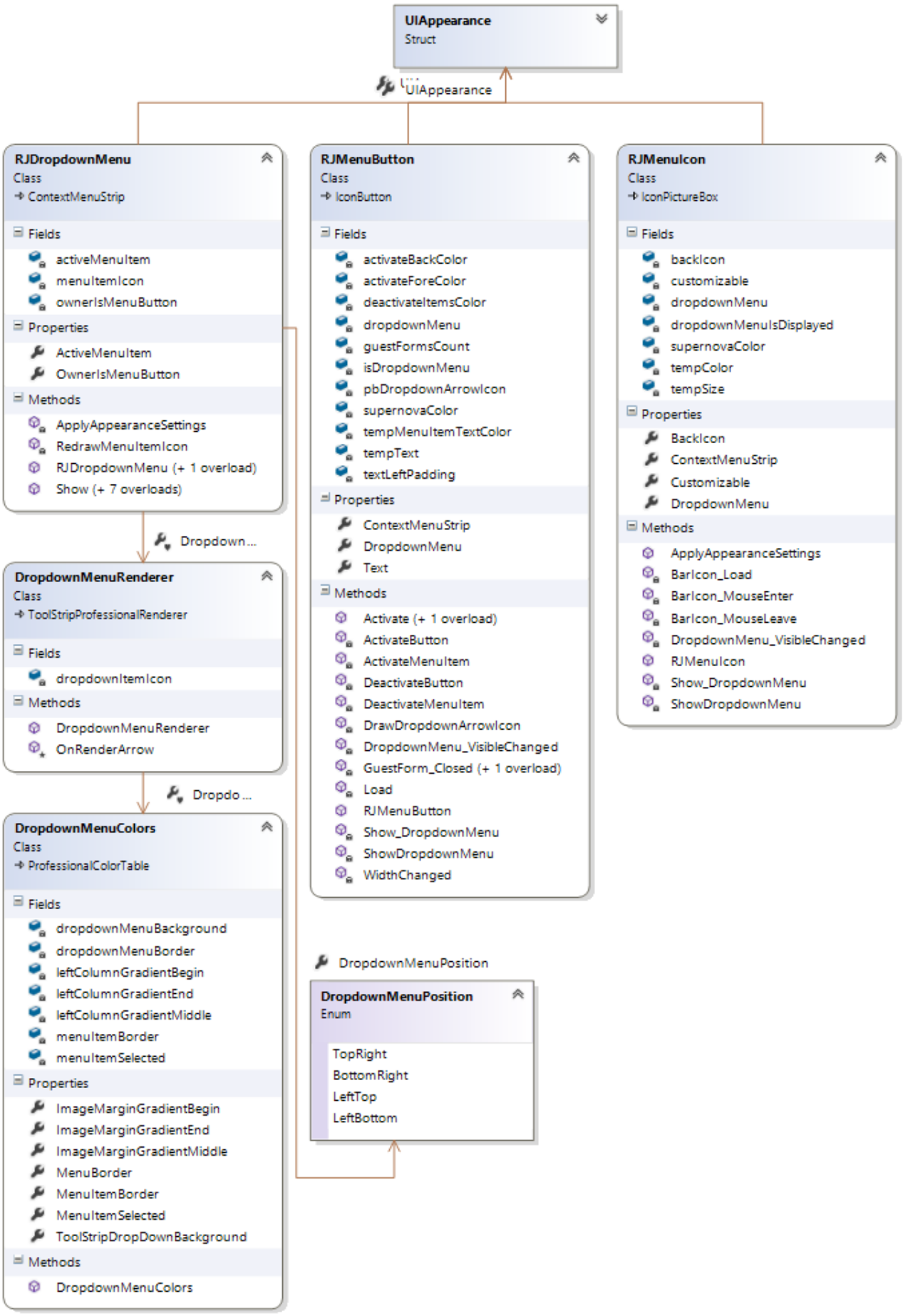



#### *8.2.4.1. Class diagram*

Collapsed class diagram



## Expanded class diagram



#### *8.2.4.2. RJ DropdownMenu class*

This class **inherits** from the **ContextMenuStrip** class in the **Windows.Forms** library.

This **dropdown menu** control allows to have **several multilevel elements** (Hierarchically organized), it does not expose customization properties. **The appearance color is established according to the theme and style established** by the appearance settings in the application, for this it **is necessary the DropdownMenuRendereder and DropdownMenuColors classes.**

Keep in mind that this control **is also a component**, make sure to **manually dispose** it if you instantiated it by code, or you can **add it to a component object** and dispose it later. If you dragged the control onto the form, it **is automatically instantiated by its constructor that implements IContainer**, therefore it will be disposed of along with the form and the other components where it is not necessary to dispose of it manually (See the **Dispose(bool disposing)** method of any **form.designer.cs**), for more information, see **point 8.4** (Components).

### Properties

| | |
|---|---|
| `ActiveMenuItem` | Gets or sets whether the drop-down menu has an active menu item. |
| `OwnerIsMenuItem` | Gets or sets whether the owner of the drop-down menu is a **menu button** (RJMenuButton). |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Sets the appearance settings to the properties of the control. |
| `RedrawMenuItemIcon ( Image itemImage)` | It is responsible for drawing the icon / image of the menu item with a suitable size and centered. |
| `RJDropdownMenu ()` | Constructor without parameters, when using this constructor make sure to manually dispose the control. |
| `RJDropdownMenu ( IContainer container)` | Constructor with IContainer parameter, **this constructor is invoked automatically in the form designer** when the control is dragged from the toolbox to the form.<br>This constructor **ensures that the object is removed correctly**, as it is not a child of the form. |
| `Show(Control ownerControl, DropdownMenuPosition position)` | It allows to **display and position** (see the DropdownMenuPosition.cs enumeration) the drop-down menu more quickly. |
| `Show (+6 overloads)` | Show the drop-down menu. |

## DropdownMenuRenderer class

This class **inherits** from the **ToolStripProfessionalRenderer** class in the **Windows.Forms** library.

This class controls the functionality of the drawing, by applying a custom palette and an optimized style.

### Fields

| | |
|---|---|
| `dropdownItemIcon` | Gets or sets the arrow icon for the drop-down menu item. |

### Methods

| | |
|---|---|
| `DropdownMenuRenderer ()` | It is responsible for sending the professional color table object that is used to draw the drop down menu. |
| `OnRenderArrow ( ToolStripArrowRenderEventArgs and)` | It is responsible for drawing the drop-down arrow icon of a menu item if it is the parent of other menu items. |

## DropdownMenuColors class

This class **inherits** from the **ProfessionalColorTable** class in the **Windows.Forms** library.

This class **sets the drop-down menu appearance colors** from the application's appearance settings.

### Fields

| | |
|---|---|
| `dropdownMenuBackground` | Gets or sets the background color. |
| `dropdownMenuBorder` | Gets or sets the border color. |
| `leftColumnGradientBegin` | Gets or sets the start color for the left column of the drop-down menu. |
| `leftColumnGradientMiddle` | Gets or sets the color of the center of the left column of the drop-down menu. |
| `leftColumnGradientEnd` | Gets or sets the end color for the the left column of the drop-down menu. |
| `menuItemSelected` | Gets or sets the background color of the selected menu item. |
| `menuItemBorder` | Gets or sets the border color of the menu item. |

### Methods

| | |
|---|---|
| `DropdownMenuColors ( bool menuButtonOwner)` | It is responsible for initializing the appearance fields according to the application's appearance settings. |

#### *8.2.4.3. RJ MenuButton class*

This class **inherits** from the **IconButton** class of the **FontAwesome.Sharp** library.

This is a special control that **is only designed to be used in the side menu of the main form**. It can work as a **normal menu button or as a drop down menu button**, for this it **is necessary to add a drop down menu** (RJDropdownMenu) from the **DropdownMenu property** of this control, the click event will be created automatically to show the drop down menu.

It has 2 essential appearance methods:

- ✓ Like a **normal menu button**; allows you to **associate a form and the button is activated / highlighted** until the form is closed (See the Activate (RJChildForm) method).
- ✓ Like a **drop down menu button**; allows **associating many forms, the button and the menu item are activated / highlighted** until the form is closed (See Activate(RJChildForm, ToolStripMenuItem) method)

### Properties

| | |
|---|---|
| `DropdownMenu` | Gets or sets an RJDropdownMenu control to the menu button, the click event is automatically created to display the dropdown menu. |
| `Text` | Gets or sets the text of the control, its special function is to add a padding before the text to keep some distance between the icon and the text. |

### Methods

| | |
|---|---|
| `Activate (RJChildForm guestForm)` | It is responsible for associating a secondary form and activating the menu button. |
| `Activate (RJChildForm guestForm, ToolStripMenuItem menuItem)` | It is responsible for associating a child form and activating the menu button and the menu item that owns the form. |
| `ActivateButton ()` | It is responsible for activating or highlighting the menu button. |
| `ActivateMenuItem (RJChildForm guestForm, ToolStripMenuItem menuItem)` | It is responsible for activating or highlighting the menu item from the drop down menu. |
| `DeactivateButton ()` | Disable the button. The appearance of the button is restored. |
| `DeactivateMenuItem ( ToolStripMenuItem menuItem)` | Disable the menu item. Appearance is restored. |
| `DrawDropdownArrowIcon (bool expandedMenu)` | It is responsible for drawing the arrow icon for the drop-down menu button in its expanded or collapsed state. |

| | |
|---|---|
| `DropdownMenu_VisibleChanged ( object sender, EventArgs and)` | Activates the drop-down menu button when the associated drop-down menu is displayed, it also takes care of invoking the method DrawDropdownArrowIcon(). |
| `GuestForm_Closed ( object sender, FormClosedEventArgs and)` | It is responsible for deactivating the menu button when the form is closed. |
| `GuestForm_Closed ( object sender, FormClosedEventArgs and, ToolStripMenuItem menuItem)` | It is responsible for deactivating the menu item when the form is closed, and deactivating the menu button when all forms associated with it are closed. |
| `MB_HandleCreated (object sender, EventArgs and)` | It is responsible for storing the text of the menu button in a temporary field. |
| `ShowDropdownMenu ()` | It is responsible for **displaying the dropdown menu** (RJDropdownMenu) |
| `DropdownMenuButton_Click (object sender, EventArgs and)` | Invokes the ShowDropdownMenu method when the control is clicked. |
| `WidthChanged (object sender, EventArgs and)` | It is responsible for hiding or displaying the menu button text when the main form side menu is collapsed or expanded. |

#### *8.2.4.4. RJ MenuIcon class*

This class **inherits** from the **IconPictureBox** class of the **FontAwesome.Sharp** library.

This control is **designed primarily on the title bar of the main form**, however you can disable it by setting the **BackIcon property to true**.

Like the RJMenuButton control, this control can work as a **normal menu icon or as a drop-down menu icon**, for this it **is necessary to add a drop-down menu** (RJDropdownMenu) from the **DropdownMenu property** of this control, the click event will be created automatically to display the drop-down menu.

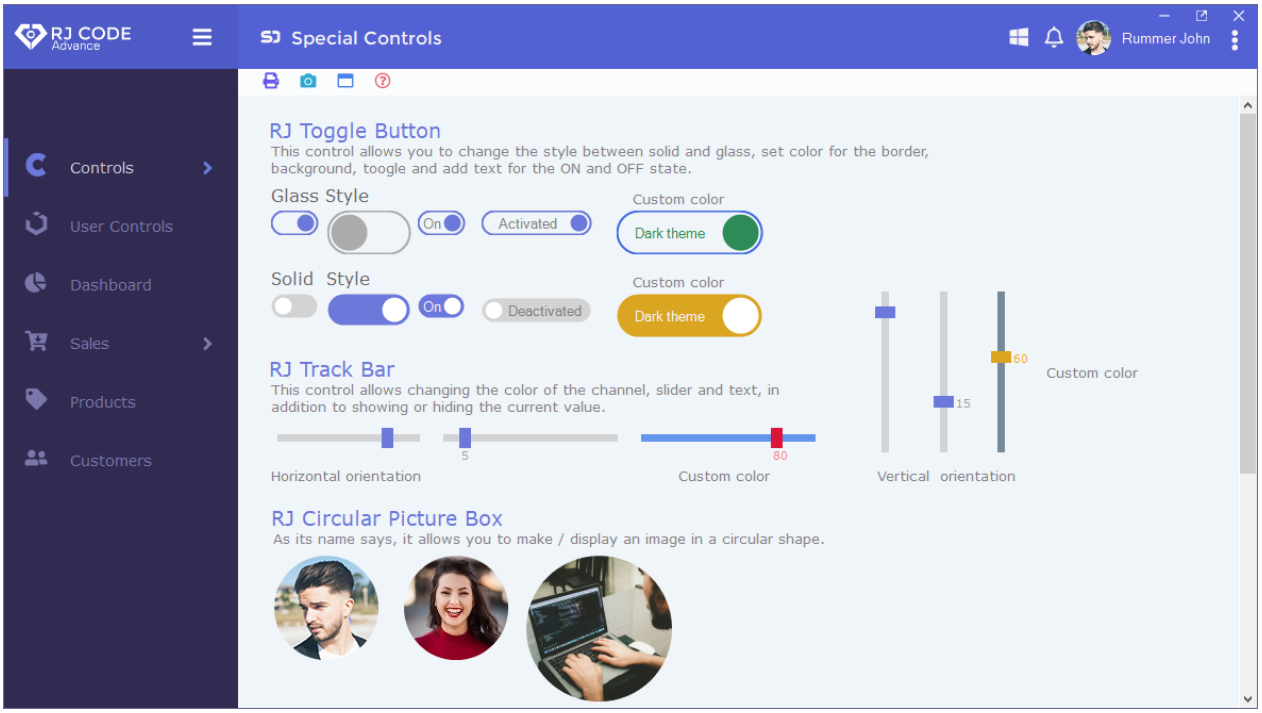
Allows you to change the icon and background color if you set the Customizable property to true.

**Properties**

| | |
|---|---|
| `BackIcon` | Gets or sets whether the control is a menu icon in the title bar of the main form (**False**), or is a menu icon in any client area of the form (**True**). |
| `Customizable` | Gets or sets whether the background and icon color of the control is customizable. |
| `DropdownMenu` | Gets or sets an **RJDropdownMenu** control to the menu button, the click event is automatically created to display the dropdown menu. |

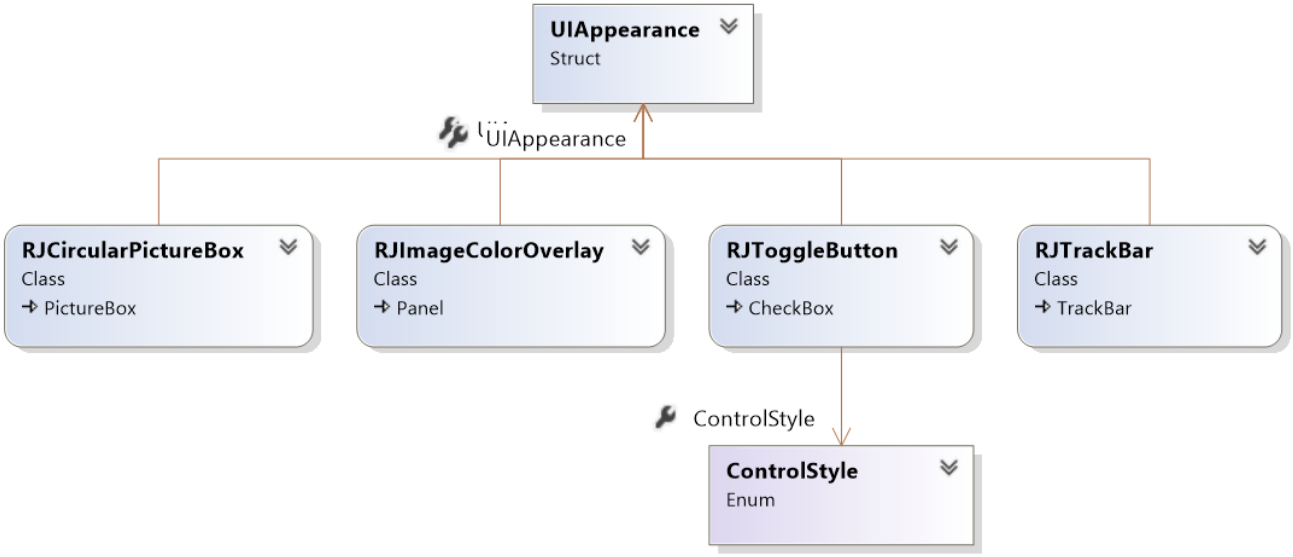
### 8.2.5. Special controls

They are controls that do not belong to any of the previous categories or **are not found in the conventional controls** of the **Windows.Forms** library.

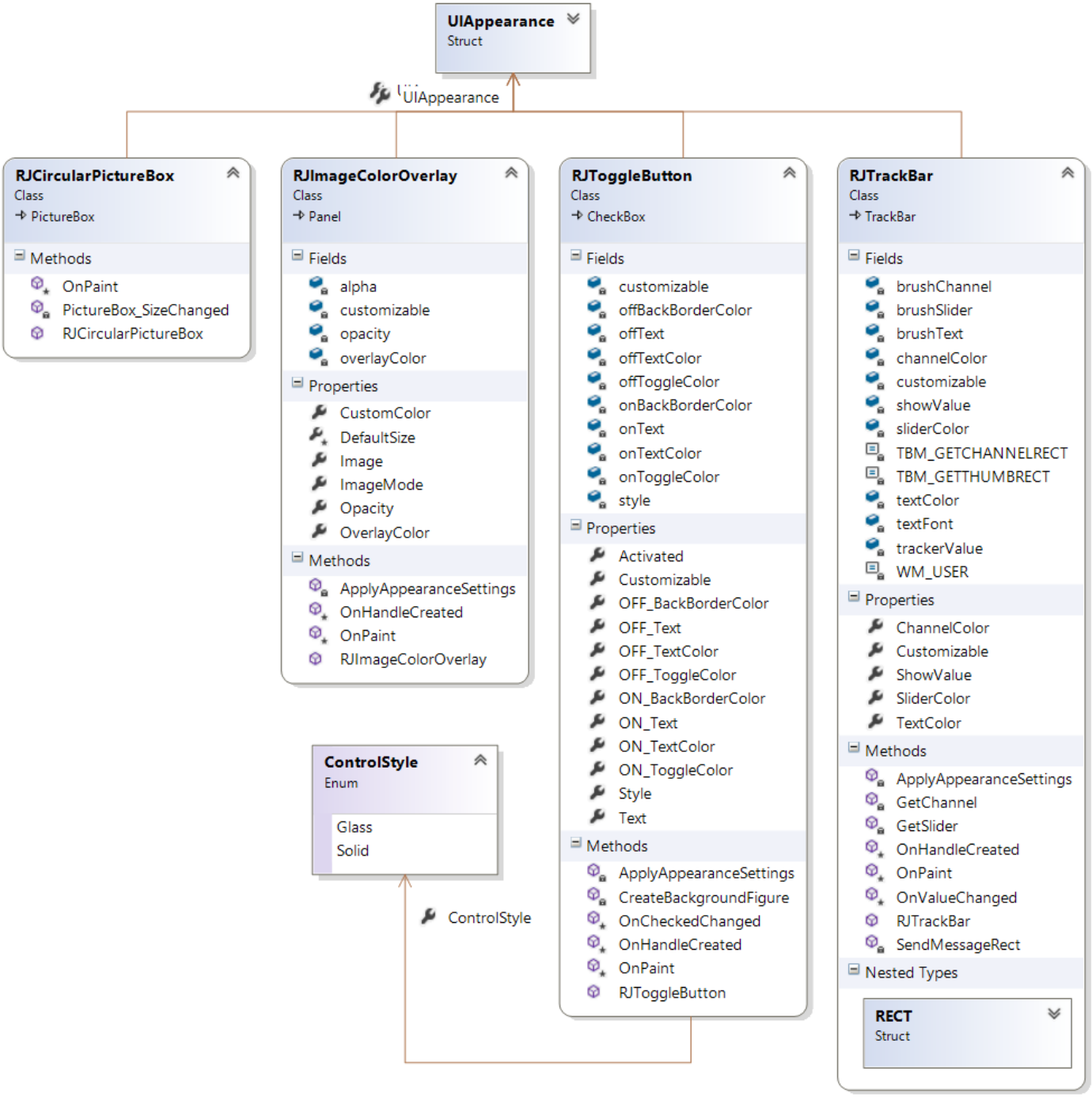



#### *8.2.5.1. Class diagram*

Collapsed class diagram



## Expanded class diagram



#### 8.2.5.2. RJ CircularPictureBox Class

This class **inherits** from the **PictureBox** class in the **Windows.Forms** library.

This control allows you to make and display an **image in a circular way.**

### Methods

| | |
|---|---|
| `OnPaint (PaintEventArgs and)` | It is responsible for creating the control in a circular fashion. |
| `PictureBox_SizeChanged ( object sender, EventArgs and)` | It is in charge of keeping the circle with the same radius (Perfect circle - same height and width). |

#### 8.2.5.3. RJ ToggleButton class

This class **inherits** from the **CheckBox** class in the **Windows.Forms** library.

This control completely cancels the paint event and a **new design of the control is drawn**. It implements a wide **variety of appearance customization properties**. By default it is set according to the application's appearance settings, you can change it by setting the Customizable property to true.

### Properties

| | |
|---|---|
| `Activated` | Gets or sets a value that indicates whether the ToggleButton control is enabled or disabled (On or Off). |
| `Customizable` | Gets or sets whether the appearance colors are customizable. |
| `OFF_BackBorderColor` | Gets or sets the background or border color for the disabled state. |
| `OFF_Text` | Gets or sets the text for the disabled state. |
| `OFF_TextColor` | Gets or sets the text color for the disabled state. |
| `OFF_ToggleColor` | Gets or sets the toggle color for the disabled state. |
| `ON_BackBorderColor` | Gets or sets the background or border color for the on state. |
| `ON_Text` | Gets or sets the text of the on state. |
| `ON_TextColor` | Gets or sets the text color for the on state. |
| `ON_ToggleColor` | Gets or sets the toggle color for the on state. |
| `Style` | Gets or sets the style of the control (Solid or Glassy). |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Set the application appearance settings in the appearance properties of the control. |
| `CreateBackgroundFigure ()` | Responsible for drawing the rounded bottom or border of the ToggleButton control. |
| `OnCheckedChanged (EventArgs and)` | Responsible for redrawing the control if the state or value changes (On State or Off State). |
| `OnHandleCreated (EventArgs and)` | It is responsible for loading and applying the appearance settings when the control handle is created. |
| `OnPaint (PaintEventArgs and)` | Responsible for drawing the new control design. |

#### *8.2.5.4. RJ TrackBar class*

This class **inherits** from the **TrackBar** class in the **Windows.Forms** library.

This control completely cancels the paint event and a **new design of the control is drawn**. Implements **appearance customization properties**. By default it is set according to the application's appearance settings, you can change it by setting the Customizable property to true.

This control is **based** on the **suggested example** of Hans passant.

### Properties

| | |
|---|---|
| `ChannelColor` | Gets or sets the channel color of the control. |
| `Customizable` | Gets or sets whether the appearance colors are customizable. |
| `ShowValue` | Gets or sets whether the value label is displayed. |
| `SliderColor` | Gets or sets the color of the slider. |
| `TextColor` | Gets or sets the text color of the value label. |

### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Set the application appearance settings in the appearance properties of the control. |
| `GetChanel ()` | Responsible for getting the channel size and location from the Windows track bar. |
| `GetSlider ()` | Responsible for getting the size and location of the Windows track bar slider. |

| | |
|---|---|
| `OnHandleCreated (EventArgs and)` | It is responsible for loading and applying the appearance settings when the control handle is created. |
| `OnPaint (PaintEventArgs and)` | Responsible for drawing the new control design. |
| `OnValueChanged (EventArgs and)` | Responsible for redrawing the control if the value changes. |
| `SendMessageRect (IntPtr hWnd, int msg, IntPtr wParam, ref RECT lParam)` | External method, responsible for sending Windows messages. |

## 8.3. Composite Controls (User Control)

As I mentioned earlier, a compound control **is created by multiple existing controls** using the **user control class** (UserControl), for this you can **add a UserControl from the designer and add controls**, or you can **inherit the UserControl class and add controls** through code, **the Constituent controls of the user control retain all their functionality** where you can expose and bind properties, as well as associate or create the necessary events.

The UserControl class is basically a container for other controls and provides keyboard routing for child controls and allows child controls to function as a group.

**I recommend using this method when it is really necessary and as a last option** as it is heavier than an extended or custom control.

Microsoft recommends using user controls for the following case and I also agree with it:

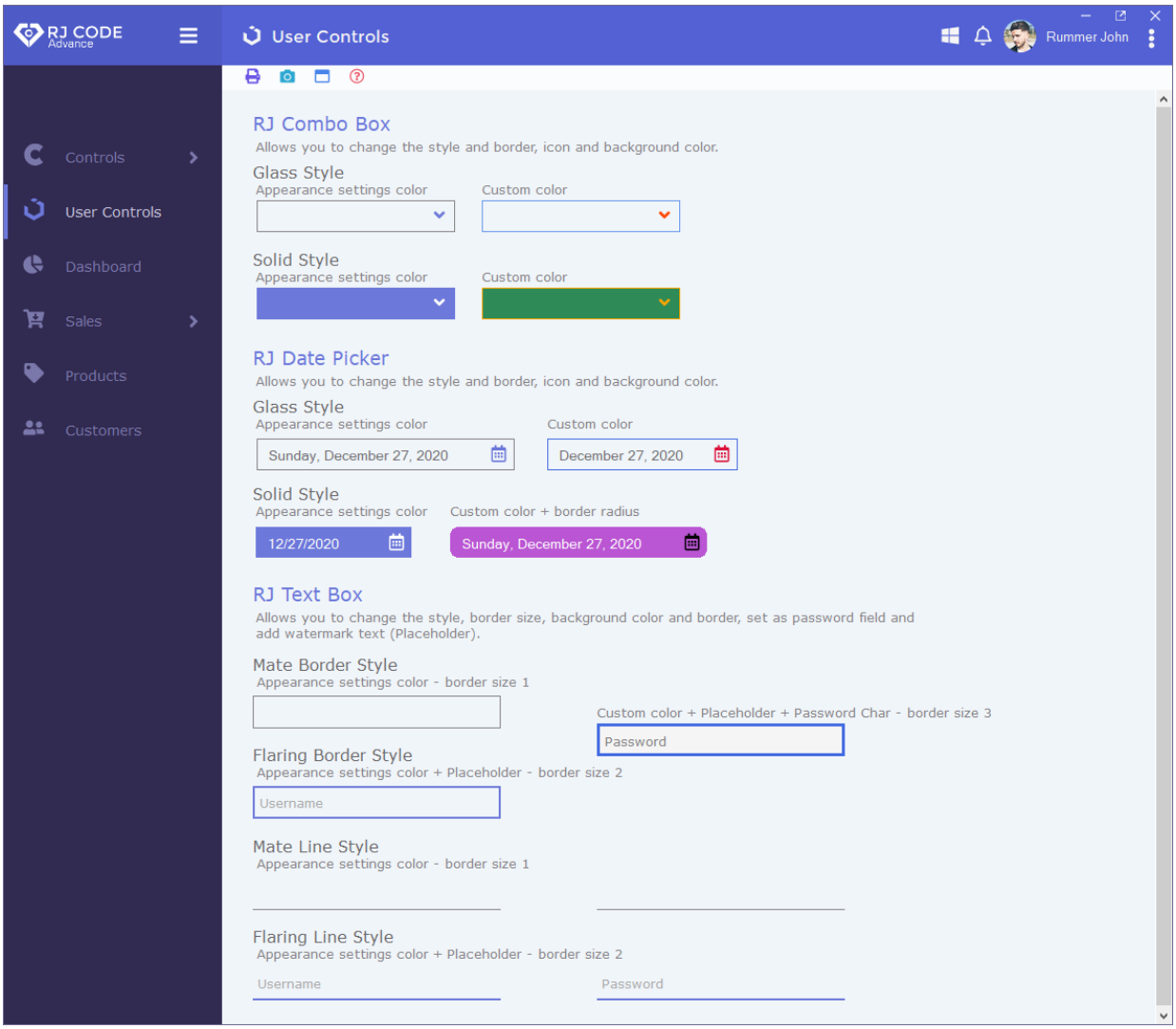
*"Inherited from the class UserControl if:*

- *You want to combine the functionality of several Windows Forms controls into a single reusable unit "*

Well, in this project I only created **3 user controls**:

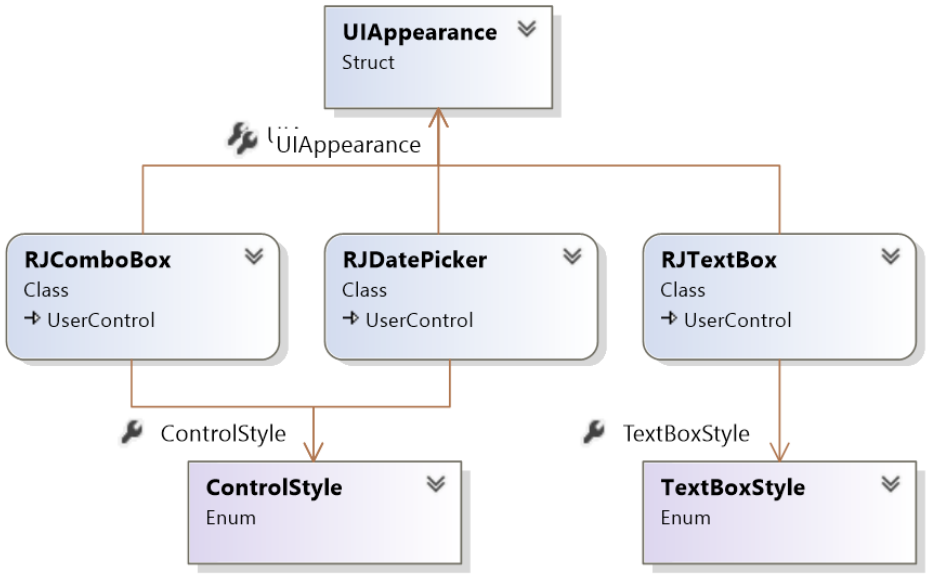
1. Combo box
2. Date picker
3. Text's box

Below is a **screenshot** of these user controls with all the available styles and designs:

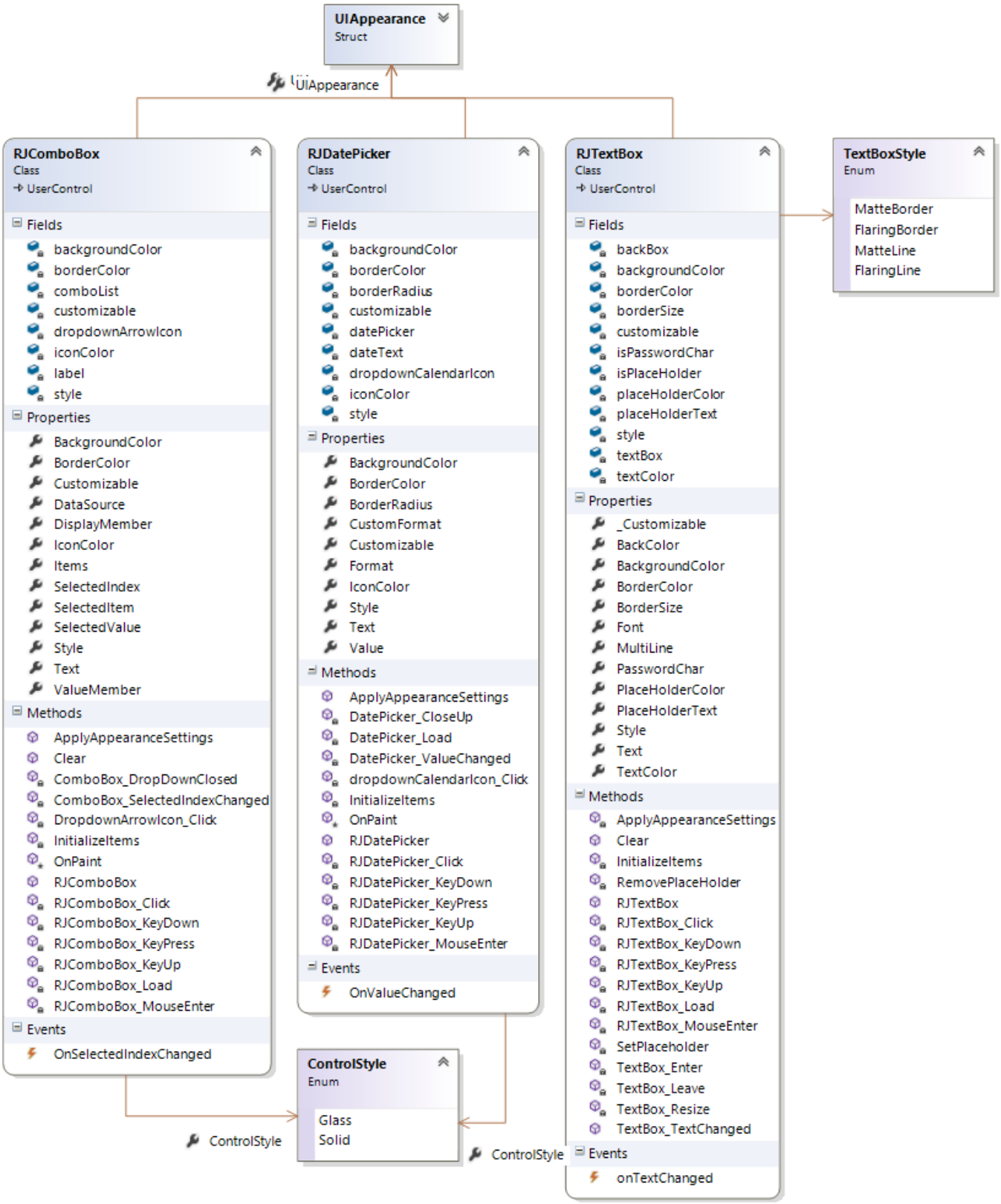



### 8.3.1. Class diagram

## Collapsed class diagram



## Expanded class diagram



### 8.3.2. RJ ComboBox Class

This class **inherits** from the **UserControl** class of the **Windows.Forms** library.

To create the RJComboBox control, add the **ComboBox and Label** control from the Windows.Forms library, and also add the **IconButton** control from the **FontAwesome.Sharp** library. It implements **appearance customization properties and exposes most of the essential properties and events of the Windows ComboBox** control.

#### Properties

| | |
|---|---|
| `BackgroundColor` | Gets or sets the background color |
| `BorderColor` | Gets or sets the border color. |
| `Customizable` | Gets or sets whether the appearance colors are customizable. |
| `DataSource` | Gets or sets the data source for the combo box. |
| `DisplayMember` | Gets or sets the property to display for this ListControl. |
| `IconColor` | Gets or sets the drop-down arrow icon color. |
| `Items` | Gets an object that represents the collection of items contained in this ComboBox. |
| `SelectedIndex` | Gets or sets the index that the currently selected item specifies. |
| `SelectedItem` | Gets or sets the currently selected item in the ComboBox. |
| `SelectedValue` | Gets or sets the value of the member property specified by the ValueMember property. |
| `Style` | Gets or sets the style of the control (Solid or Glassy) |
| `Text` | Gets or sets the text of the user control. |
| `ValueMember` | Gets or sets the path of the property to use as the actual value for the ListControl items. |

#### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Set the application appearance settings in the appearance properties of the control. |
| `Clear ()` | Remove all items from the ComboBox. |
| `ComboBox_DropDownClosed ( object sender, EventArgs and)` | Responsible for resetting the appearance of the control when the dropdown list is closed. |
| `ComboBox_SelectedIndexChanged ( object sender, EventArgs and)` | Responsible for updating the text of the control when the selection index changes. |

| | |
|---|---|
| `DropdownArrowIcon_Click ( object sender, EventArgs and)` | Responsible for displaying the ComboBox drop-down list and setting the active status appearance. |
| `RJComboBox_Click (object sender, EventArgs and)` | Responsible for linking the click event of the constituent front control to the click event of the user control. |
| `RJComboBox_MouseEnter ( object sender, EventArgs and)` | Responsible for binding the MouseEnter event of the constituent front control to the MouseEnter event of the user control. |
| `RJComboBox_KeyUp (object sender, KeyEventArgs and)` | Responsible for binding the KeyUp event of the constituent front control to the KeyUp event of the user control. |
| `RJComboBox_KeyPress ( object sender, KeyPressEventArgs and)` | Responsible for binding the KeyPress event of the constituent front control to the KeyPress event of the user control. |
| `RJComboBox_KeyDown ( object sender, KeyEventArgs and)` | Responsible for binding the KeyDown event of the constituent front control to the KeyDown event of the user control. |

### Events

| | |
|---|---|
| `OnSelectedIndexChanged` | Default event of the user control recreating and binding to the original SelectedIndexChanged event of the ComboBox control. |

#### 8.3.3. RJ DatePicker class

This class **inherits** from the **UserControl** class of the **Windows.Forms** library.

To create the RJDatePicker control, add the **DateTimePicker and Label** control from the Windows.Forms library, and also add the **IconButton** control from the **FontAwesome.Sharp** library. It implements **appearance customization properties and exposes most of the essential properties and events of the Windows DateTimePicker** control.

### Properties

| | |
|---|---|
| `BackgroundColor` | Gets or sets the background color |
| `BorderColor` | Gets or sets the border color. |
| `BorderRadius` | Gets or sets the radius to apply rounded corners to the control. |
| `CustomFormat` | Gets or sets the custom date format string. |
| `Customizable` | Gets or sets whether the appearance colors are customizable. |

| | |
|---|---|
| Format | Gets or sets the format of the date that is displayed in the control. |
| IconColor | Gets or sets the icon color. |
| Style | Gets or sets the style of the control (Solid or Glassy) |
| Text | Gets the text of the user control. |
| Value | Gets or sets the date value assigned to the control. |

## Methods

| | |
|---|---|
| ApplyAppearanceSettings () | Set the application appearance settings in the appearance properties of the control. |
| DatePicker_CloseUp ( object sender, EventArgs and) | Responsible for restoring the appearance of the control when the calendar is closed. |
| DatePicker_ValueChanged ( object sender, EventArgs and) | Responsible for updating the control text when the date picker value changes. |
| DropdownCalendarIcon_Click ( object sender, EventArgs and) | Responsible for displaying the date picker calendar and setting the active status appearance. |
| RJDatePicker_Click (object sender, EventArgs and) | Responsible for linking the click event of the constituent front control to the click event of the user control. |
| RJDatePicker_MouseEnter ( object sender, EventArgs and) | Responsible for binding the MouseEnter event of the constituent front control to the MouseEnter event of the user control. |
| RJDatePicker_KeyUp (object sender, KeyEventArgs and) | Responsible for binding the KeyUp event of the constituent front control to the KeyUp event of the user control. |
| RJDatePicker_KeyPress ( object sender, KeyPressEventArgs and) | Responsible for binding the KeyPress event of the constituent front control to the KeyPress event of the user control. |
| RJDatePicker_KeyDown ( object sender, KeyEventArgs and) | Responsible for binding the KeyDown event of the constituent front control to the KeyDown event of the user control. |

## Events

| | |
|---|---|
| OnValueChanged | Default event of the user control, recreating and binding to the original ValueChanged event of the DateTimePicker control. |

### 8.3.4. RJ TextBox class

This class **inherits** from the **UserControl** class of the **Windows.Forms** library.

To create the RJTextBox control, add the **TextBox and Panel** control from the **Windows.Forms** library. It implements **appearance customization properties**, add a water mark, better known as a **placeholder**, and **exposes most of the essential properties and events of the Windows TextBox** control.

## Properties

| | |
|---|---|
| `BackgroundColor` | Gets or sets the background color |
| `BorderColor` | Gets or sets the border color. |
| `BorderSize` | Gets or sets the border size. |
| `Customizable` | Gets or sets whether the appearance colors are customizable. |
| `Font` | Gets or sets the font. |
| `Multiline` | Gets or sets a value that indicates whether it is a multi-line TextBox control. |
| `PasswordChar` | Gets or sets the character used to mask the characters in a password. |
| `PlaceHolderColor` | Gets or sets the text color of the placeholder (Watermark). |
| `PlaceHolderText` | Gets or sets the text for the placeholder (Watermark). |
| `Style` | Gets or sets the style of the control (MatteBorder, FlaringBorder, MatteLine, and FlaringLine) |
| `Text` | Gets the text of the text box. |
| `TextColor` | Gets or sets the text color of the text box. |

## Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Set the application appearance settings in the appearance properties of the control. |
| `Clear ()` | Empty or remove the text from the text box. |
| `RemovePlaceHolder ()` | Remove the placeholder. |

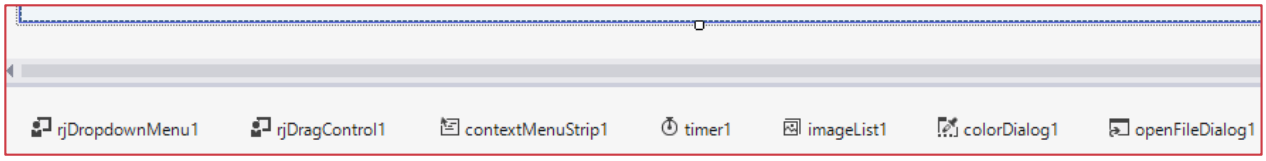
| | |
|---|---|
| `RJTextBox_Click (object sender, EventArgs and)` | Responsible for linking the click event of the constituent front control to the click event of the user control. |
| `RJTextBox_MouseEnter ( object sender, EventArgs and)` | Responsible for binding the MouseEnter event of the constituent front control to the MouseEnter event of the user control. |
| `RJTextBox_KeyUp (object sender, KeyEventArgs and)` | Responsible for binding the KeyUp event of the constituent front control to the KeyUp event of the user control. |
| `RJTextBox_KeyPress ( object sender, KeyPressEventArgs and)` | Responsible for binding the KeyPress event of the constituent front control to the KeyPress event of the user control. |
| `RJTextBox_KeyDown ( object sender, KeyEventArgs and)` | Responsible for binding the KeyDown event of the constituent front control to the KeyDown event of the user control. |
| `SetPlaceHolder ()` | Sets the placeholder. |
| `TextBox_Enter (object sender, EventArgs and)` | Highlights control when in focus. |
| `TextBox_Leave (object sender, EventArgs and)` | Turn off highlighting when the courses leave in control. |
| `TextBox_TextChanged (object sender, EventArgs and)` | Responsible for binding and executing the recreated default event. |

## Events

| | |
|---|---|
| `onTextChanged` | Default event of the user control, recreating and binding to the original TextChanged event of the TextBox control. |

## 8.4. Components

The components do **not have a visual representation** (Except for **ContextMenuStrip** since it implements the class control and component), but it is possible to select, set properties and delete it. When adding components to the form using the designer they **are placed at the bottom**:




When you instantiate a component by code you must explicitly (Manually) release resources through calls to its **Dispose()** method, or the components have and must implement **IContainer in your constructor** that allows to associate with an **object Container**.

So when you add the component via the designer (dragging the component from the toolbox to the form), it **is automatically instantiated by its constructor which implements IContainer**, thus it will be discarded along with the form and the other components when the form be closed. This is useful if you use multiple components in your application and want to get rid of all of them **simultaneously and safely.**

See the Dispose (…) method of any Form.Designer.cs, for example:

```
partial class FormComponentControls
 {
// Code automatically generated by the designer.
 private System.ComponentModel.IContainer components = null;
 protected override void Dispose (bool disposing)
 {
      if (disposing && (components! = null))
      {
           components.Dispose ();
      }
   base.Dispose (disposing);
 }
 private void InitializeComponent ()
 {
 this.components = new System.ComponentModel.Container();
 this.dmExample = new RJDropdownMenu(Este.components);
 this.dragControl =newRJDragControl(Este.components);
```

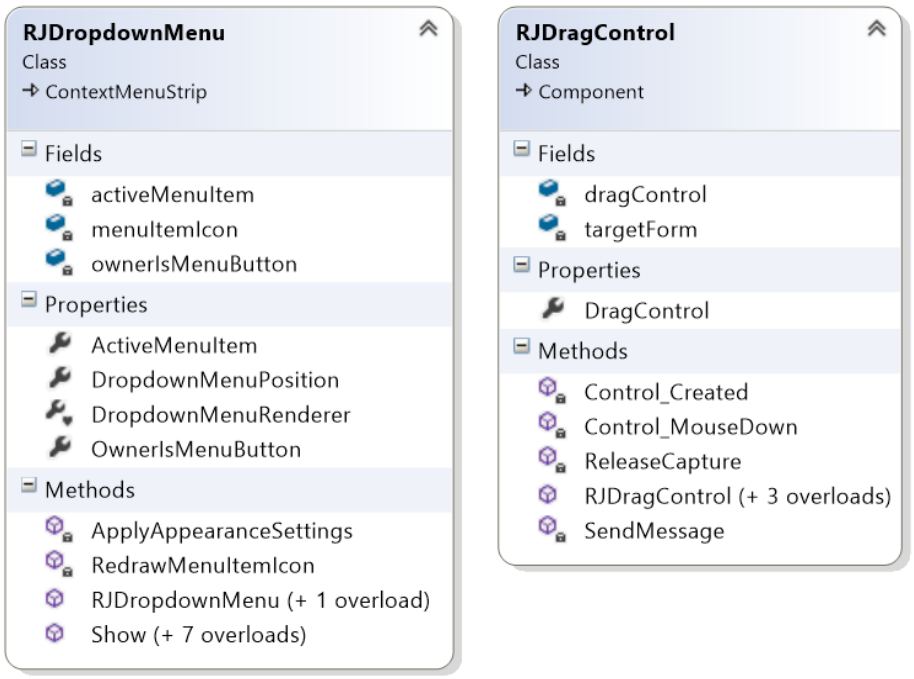
More information:

- ✓ Component
- ✓ ContextMenuStrip
- ✓ Timer
- ✓ Dispose - IContainer

### 8.4.1. Class diagram

#### Collapsed class diagram


#### Expanded class diagram




**Note:**

The **RJDropdownMenu class inherits from the ContextMenuStrip** class in the **Windows.Forms** library, and **ContextMenuStrip derives** from the **Control** class and the **IComponent** interface.

So ContextMenuStrip and RJDropdownMenu **also belong to component group**, **both classes implement IContainer in their constructor**, you don't need to drop the objects manually if you added to the form via designer, as mentioned above.

For a **description of the properties and methods of the RJDropdownMenu** control, see the point **8.2.4.2.**

### 8.4.2. RJ DragControl class

This class **inherits** from the **Component** class of the **System.ComponentModel** library.

This component allows you to drag or **move the form using any specified control**.

#### Properties

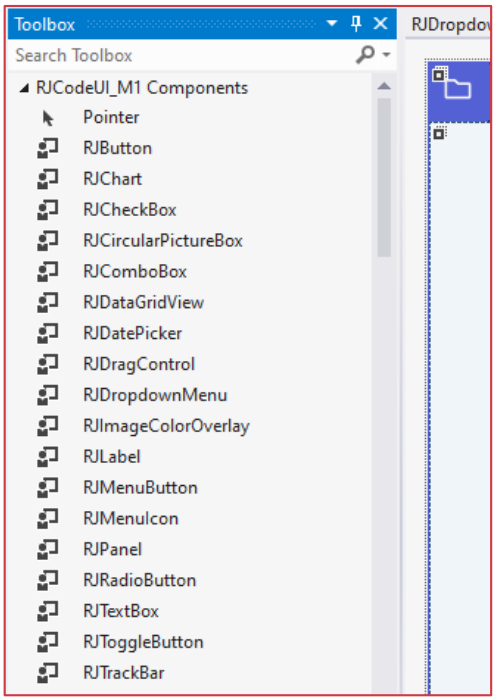
| | |
|---|---|
| `DragControl` | Gets or sets the control responsible for dragging the form. |

### Methods

| | |
|---|---|
| `Control_Created (object sender, EventArgs and)` | Responsible for setting the target form to drag. |
| `Control_MouseDown (object sender, MouseEventArgs and)` | Responsible for dragging the form when the mouse button is pressed. |
| `ReleaseCapture ()` | External method responsible for freeing the mouse catch on the current thread. |
| `SendMessage (IntPtr hWnd, int wMsg, int wParam, int lParam)` | External method responsible for sending messages to a window or windows. |
| `RJDragControl ( IContainer container)` | Constructor with IContainer parameter, **this constructor is invoked automatically in the form designer** when the control is dragged from the toolbox to the form.<br>This constructor **ensures that the object is removed correctly.** |
| `RJDragControl ( Control _dragControl, Form _targetForm)` | Initializes a new instance with the specified form and drag control. |
| `RJDragControl ( Control _dragControl, Form _targetForm, IContainer container)` | Initializes a new instance with the drag and form control and associates with a specified container. |

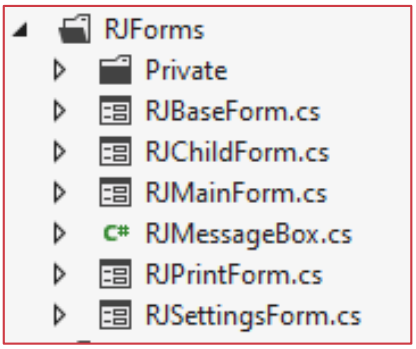
## 8.5. How to use?

When making changes to the classes, be sure to **compile the project to apply the changes**, the custom controls will be added automatically in the visual studio toolbox, finally start **dragging them to the form** like any Windows Form control.



# 9. CUSTOM FORMS

To create custom forms you **simply inherit an existing form**, then **set or extend properties, and add controls** (I recommend doing it through code).

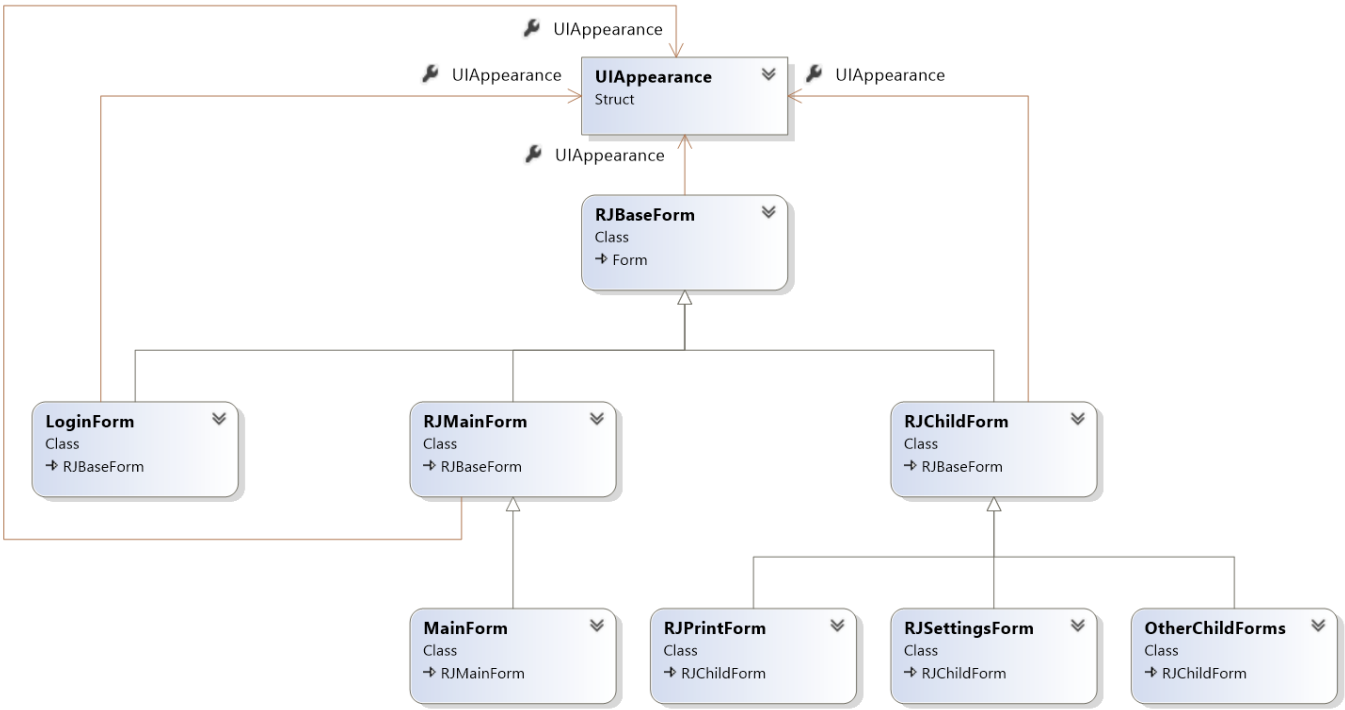



In this project **3 base custom forms** are created (RJBaseForm, RJMainForm and RJChildForm) where you can inherit any of them according to your needs and add your controls.

And **4 derived forms** are created to these base forms, already with a **finalized design and with a specific function** (RJSettingsForm, RJPrintForm, LoginForm and MainForm).

## 9.1. Class Diagram (Collapsed)

### Collapsed class diagram

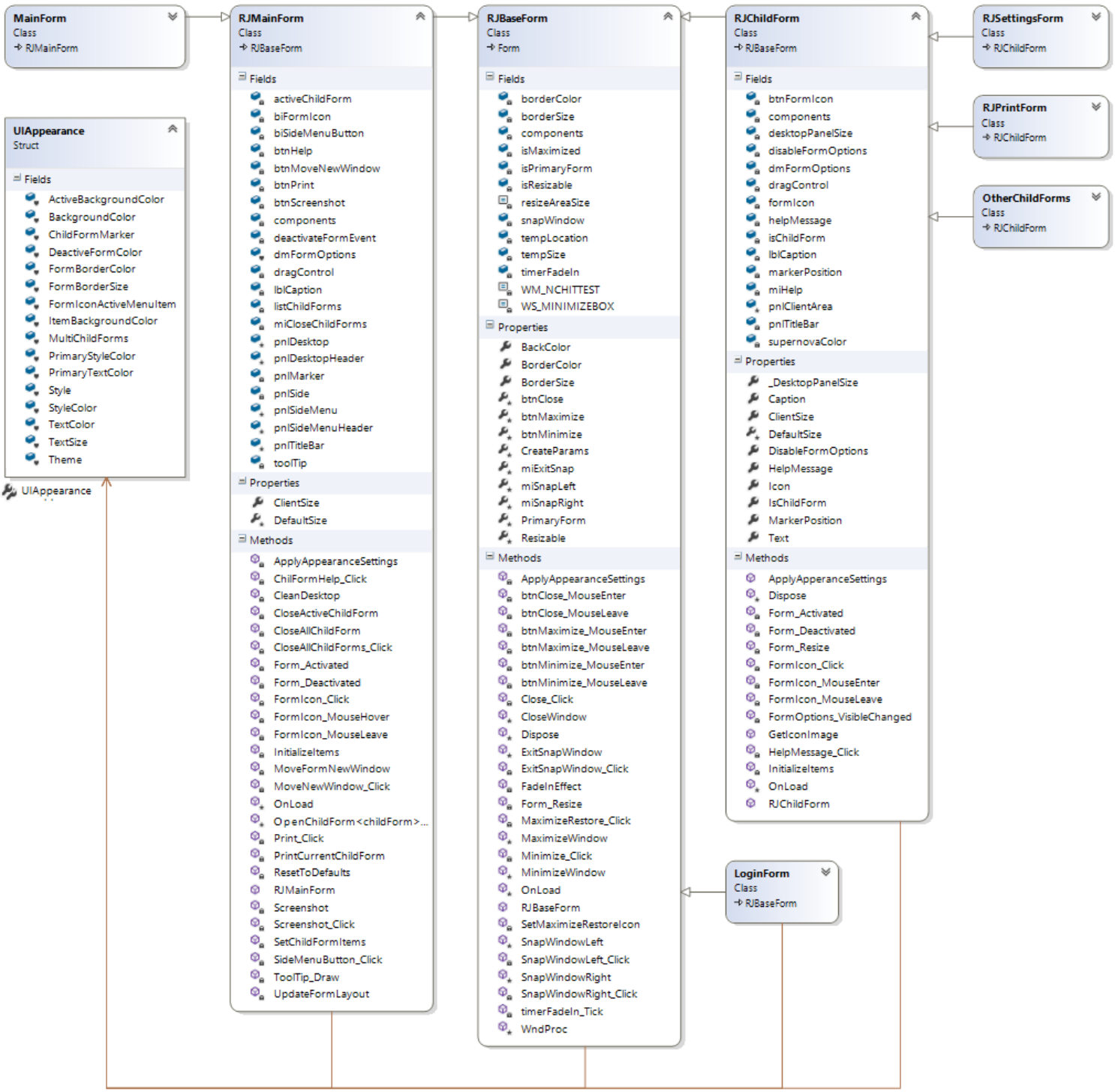


## 9.2. Base custom forms

They are **pre-built forms that you can extend** (Inherit) to **quickly create your parent or child forms**.

RJBaseForm, RJMainForm, and RJChildForm.

### 9.2.1. Class Diagram (Expanded)

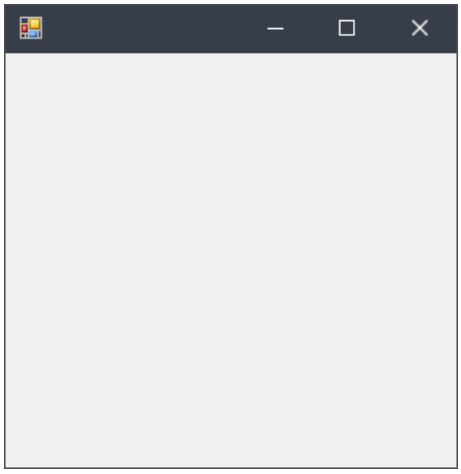


### 9.2.2. Base Form - RJBaseForm Class

This class **inherits** from the **Form** class of the **Windows.Forms** library.

This form is the **foundation** (base) **for the other forms**, it is responsible for **implementing all the functionalities of a conventional form**, such as:

- ✓ The **resize** function from the border form.
- ✓ The function of **maximize, restore, minimize and close.**
- ✓ The **snap function** (Snap Windows)



It is also responsible for implementing some **basic appearance properties**, such as:

- ✓ Buttons for control box
- ✓ Border size
- ✓ Border color
- ✓ Gradual appearance effect

#### Properties

| | |
|---|---|
| `BorderColor` | Gets or sets the border color of the form. |
| `BorderSize` | Gets or sets the border size of the form. |
| `btnClose` | Gets or sets the close button. |
| `btnMaximize` | Gets or sets the maximize button. |
| `btnMinimize` | Gets or sets the minimize button. |
| `CreateParams` | Gets or sets the form creation parameters. |
| `miExitSnap` | Gets or sets the menu item Exit Docking (Restores the form). |
| `miSnapLeft` | Gets or sets the dock window menu item left. |
| `miSnapRight` | Gets or sets the dock window right menu item. |
| `PrimaryForm` | Gets or sets whether it is the parent form. |
| `Resizable` | Gets or sets whether the form is resizable (resized) from the edges of the form. |

#### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Responsible for set the application appearance settings to the form properties. |
| `CloseWindow ()` | Responsible for closing the form or exiting the application if it is a primary form. |

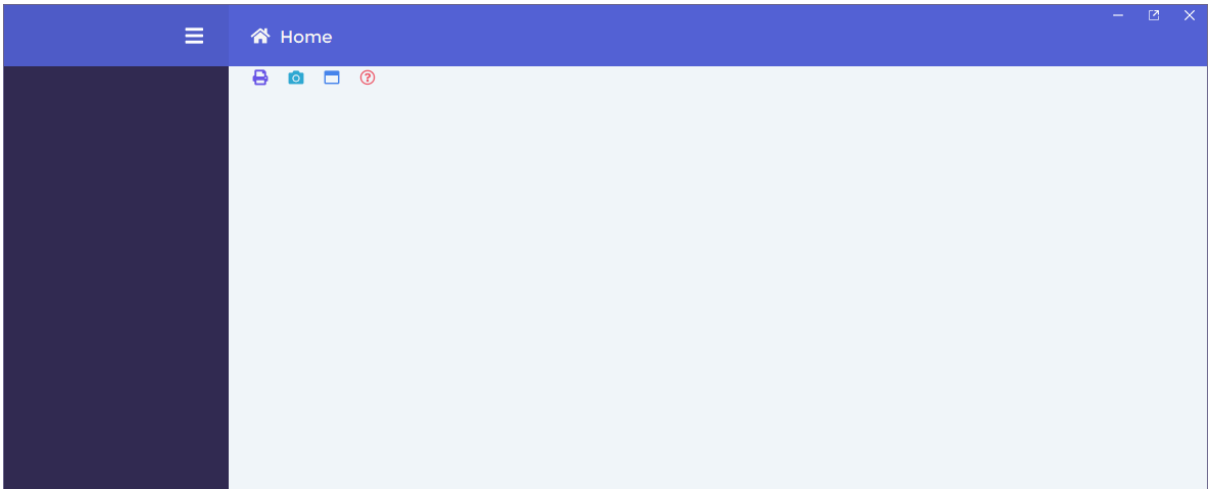
| | |
|---|---|
| `Dispose (bool disposing)` | Responsible for disposing of the form and components. |
| `ExitSnapWindow ()` | Responsible for restoring the form if the window has been docked to one side (Snap Windows). |
| `FadeInEffect ()` | Responsible for making the form appear gradually when the form is first displayed. |
| `MaximizeWindow ()` | Responsible for maximizing or restoring the form. |
| `MinimizeWindow ()` | Responsible for minimizing the form. |
| `OnLoad (EventArgs and)` | Responsible for applying appearance settings. |
| `SetMaximizeRestoreIcon ()` | Responsible for toggling the maximize and restore icon as the case may be and according to the established theme (Dark - Light). |
| `SnapWindowLeft ()` | Responsible for docking the form to the left side of the desktop. |
| `SnapWindowRight ()` | Responsible for docking the form to the right side of the desktop. |
| `WndProc (ref Message message)` | Responsible for handling Windows message processing at the operating system level for form resizing. |

### 9.2.3. Main Form - RJMainForm Class

This class **inherits** from the **RJBaseForm** class.

This is a **base form for the main form** of your application, with a **pre-designed graphical interface** (See **screenshot**), it **implements the title bar** along with the **control buttons** and a **label to display the title of the currently open form**, the **menu Sidebar** to add menu buttons along with the menu icon that allows you to collapse or expand the side menu.

Finally, it **implements the desktop panel** to show a secondary form inside it and the menu strip with the options for **screenshot, printing, help and moving the secondary form to a new window.**



## Properties

| | |
|---|---|
| `ClientSize` | Gets the size of the form's client area. |
| `DefaultSize` | Gets or sets the default size of the form. |

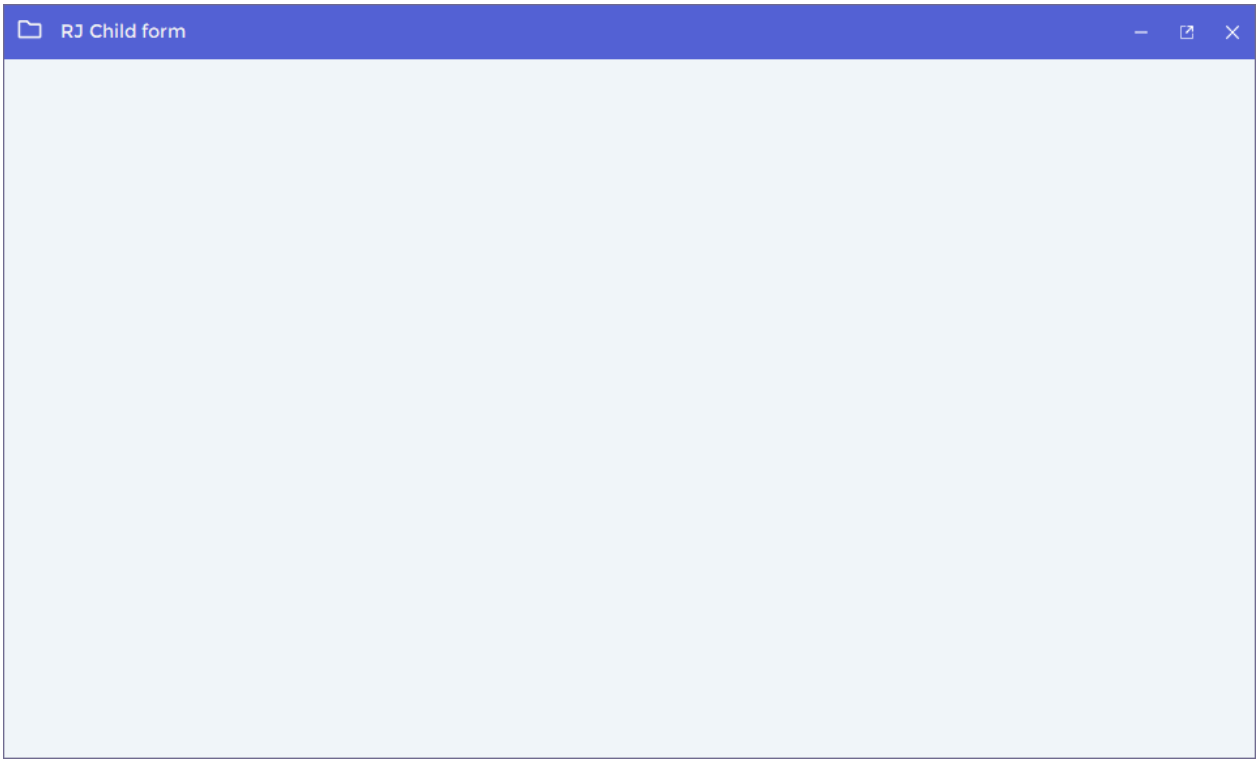
## Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Responsible for setting the application appearance settings to the form properties. |
| `CleanDesktop ()` | Responsible for cleaning up the desktop, remove the current child form from the desktop panel. |
| `CloseActiveChildForm ()` | Responsible for closing the child form currently open in the desktop panel. |
| `CloseAllChildForm ()` | Responsible for closing all open child forms. |
| `InitializeItems ()` | Responsible for initialize all components of the main form. |
| `MoveFormNewWindow ()` | Responsible for move the current form to a new window. |
| `OnLoad (EventArgs and)` | Responsible for applying appearance settings. |
| `OpenChildForm <childForm> ( Func<childForm> _delegate)` | Responsible for opening a child form (RJChildForm) in the desktop panel. |
| `OpenChildForm <childForm> ( Func<childForm> _delegate, object senderMenuButton)` | Responsible for opening a child form (RJChildForm) in the desktop panel and associating it with a menu button and then highlighting it and set the bookmark if it is the case. |
| `OpenChildForm <childForm> ( Func<childForm> _delegate, object senderMenuItem, RJMenuButton ownerMenuButton)` | Responsible for opening a child form (RJChildForm) in the desktop panel and associating it with a menu item and menu button that owns this, and then highlighting it and setting the marker if it is the case. |
| `PrintCurrentChildForm ()` | Responsible for printing the child form currently open on the desktop panel. |
| `ResetToDefaults ()` | Responsible for resetting the form defaults. |
| `Screenshot ()` | Responsible for taking a screenshot of the form. |
| `SetChildFormItems ()` | Responsible for establishing the elements of the secondary form in the main form such as the icon, title and others. |
| `UpdateFormLayout ()` | Responsible for updating the main form, it is in charge of verifying if there is any secondary form open, if it is the case it shows it again in the desktop panel, otherwise it will load its default values of the main form. |

### 9.2.4. Child Form - RJChildForm Class

This class **inherits** from the **RJBaseForm** class.

This is a **base form for the child forms** of your application, with a **pre-designed graphical interface** (See screenshot), it **implements the title bar** along with the **control buttons**, a **label to display the form's title** and the **drop-down menu with the form options.**




## Properties

| | |
|---|---|
| `_DesktopPanelSize` | Gets or sets whether the form should be the same size as the desktop pane of the main form. |
| `Caption` | Gets or sets the title of the form. |
| `ClientSize` | Gets the client area size of the form. |
| `DefaultSize` | Gets or sets the default size. |
| `DisableFormOptions` | Gets or sets whether the form options drop-down menu will be disabled. |
| `HelpMessage` | Gets or sets the help text for the form. |
| `Icon` | Gets or sets the icon for the form. |
| `IsChildForm` | Gets or sets whether it is a child form of the main form (Makes the form display on the desktop panel) |
| `MarkerPosition` | Gets or sets the marker position for the menu button. |

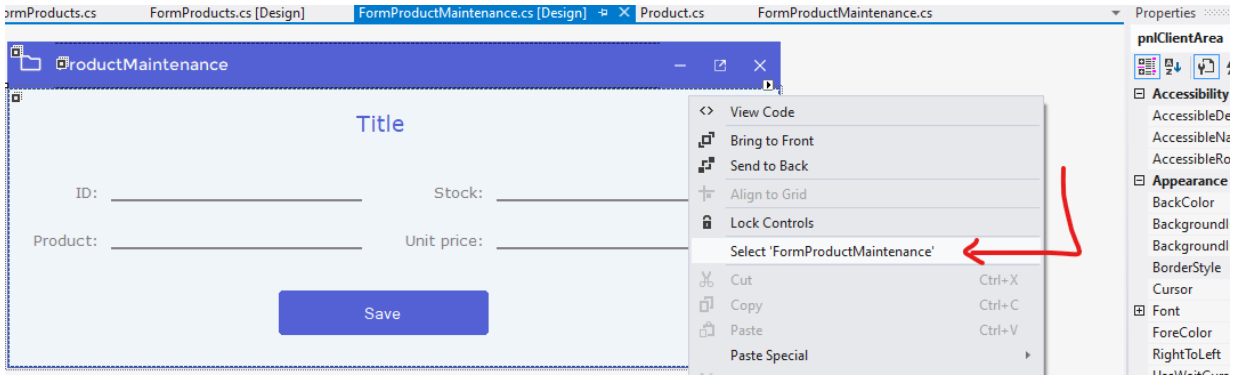
### Methods

| | |
|---|---|
| `ApplyAppearanceSettings ()` | Responsible for setting the application appearance settings to the form properties. |
| `Dispose ()` | Responsible for removing the form and components correctly. |
| `GetIconImage (int iconSize, Colour iconColor)` | Responsible for obtaining the icon of the form with size and color specified in image format. |
| `InitializeItems ()` | Responsible for initializing all components of the main form. |
| `OnLoad (EventArgs and)` | Responsible for applying appearance settings. |

#### 9.2.5. How to use?

To use them **simply inherit any of the base forms according to your needs**, as they are done below in the next point.

To **select the properties of a form, right-click on the designer and select the form**, as in the following image.



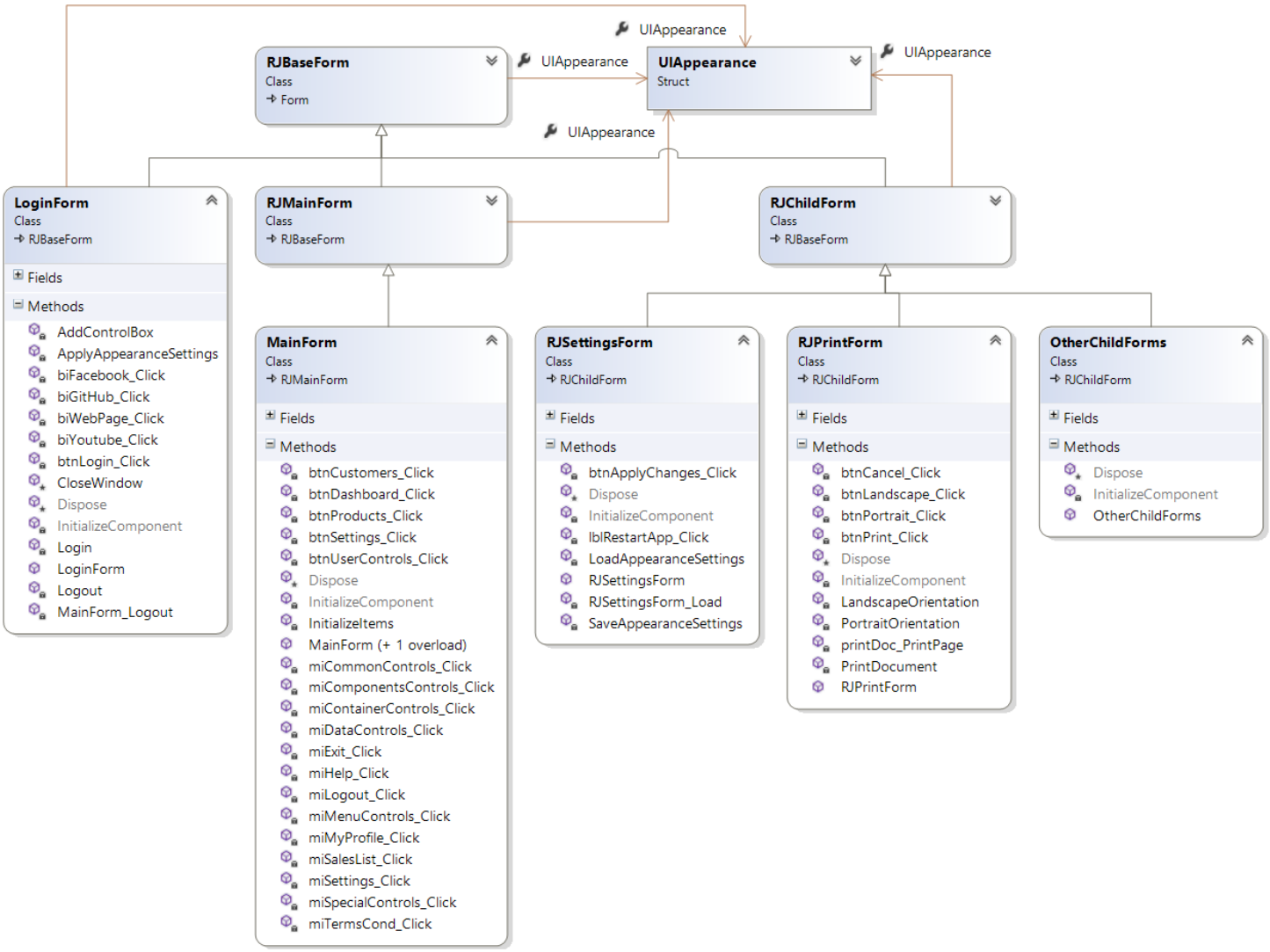

## 9.3. Derivative custom forms

They are forms already designed with a specific purpose, for example:

- ✓ **LoginForm**: Form for user login.
- ✓ **MainForm**: Main form of the application.
- ✓ **RJSettingsForm**: Form for the application's appearance setting.
- ✓ **RJPrintForm**: Form to print a specific image.

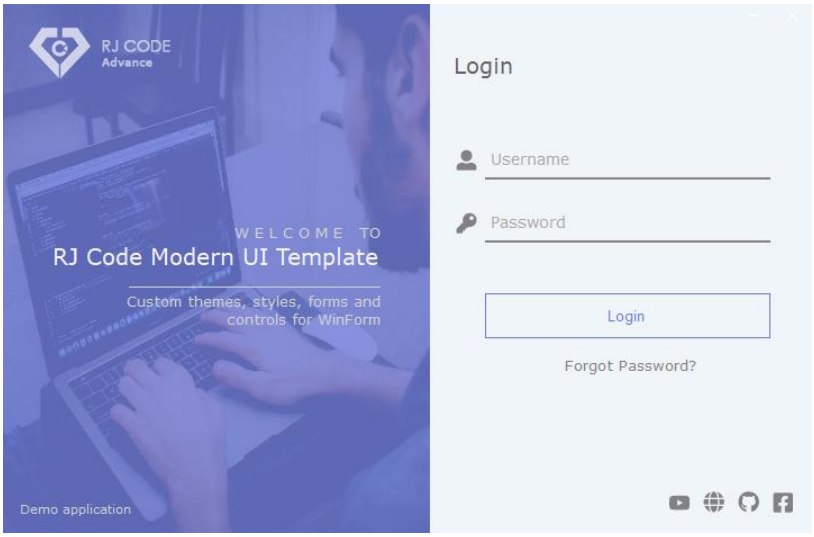
You could consider them examples of how to use the base custom forms.

### 9.3.1. Class Diagram (Expanded)




### 9.3.2. Login Form - LoginForm Class

This class **inherits** from the **RJBaseForm** class. User login form, created by designer.

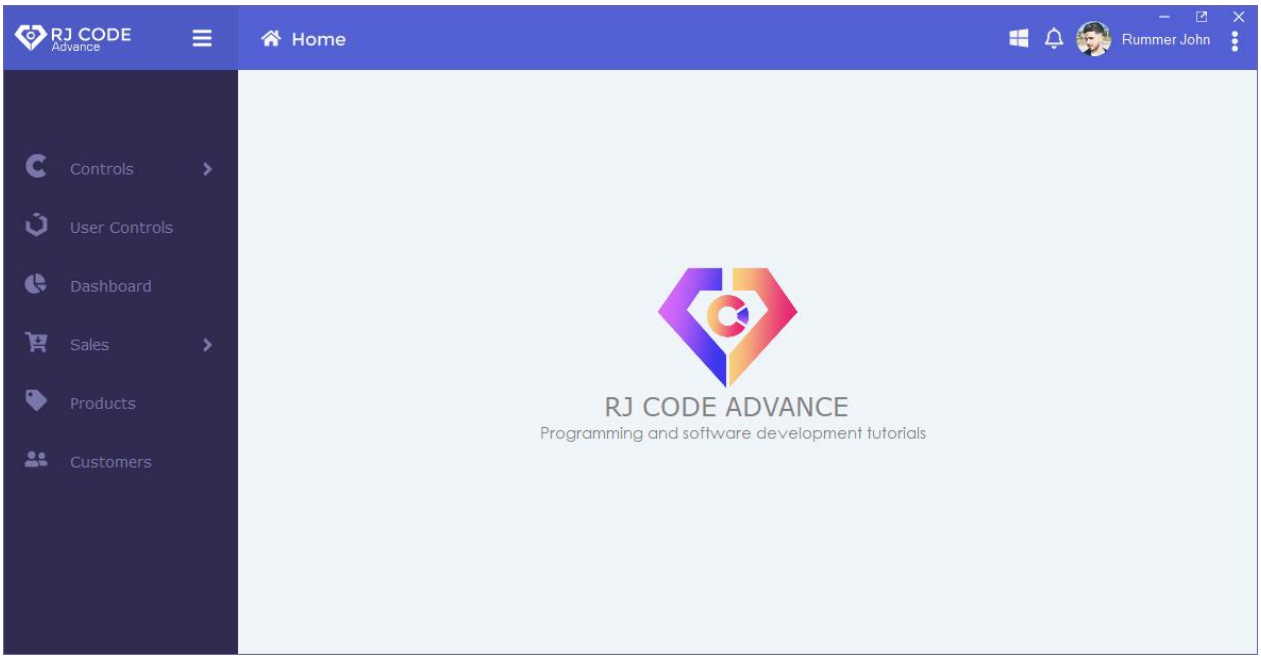


### Methods

| | |
|---|---|
| `ApplyAppearanceSettings()` | Responsible for setting the application appearance settings to the form properties. |
| `AddControlBox ()` | Responsible for removing the form and components correctly. |
| `Login()` | Responsible for obtaining the icon of the form with size and color specified in image format. |
| `Logout ()` | Responsible for initializing all components of the main form. |

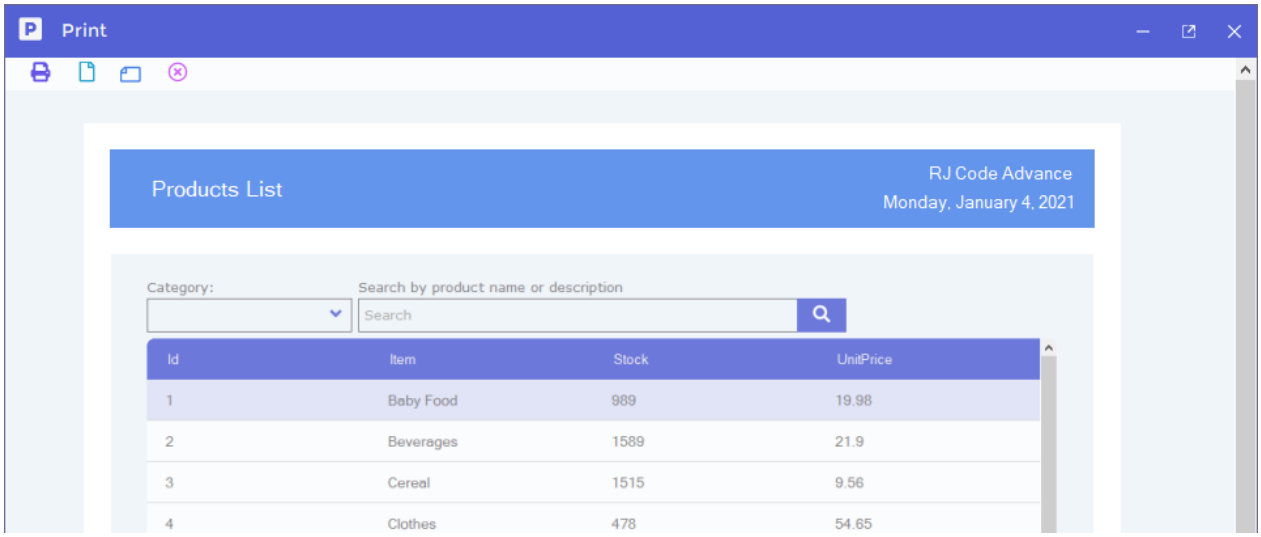
#### 9.3.3. Main Form - MainForm Class

This **class inherits** from the **RJMainForm** class. **Main application form,** created by designer.




#### 9.3.4. Print Form - RJPrintForm Class

This class **inherits** from the **RJChildForm** class. Form to print the child forms on the desktop.



### Fields

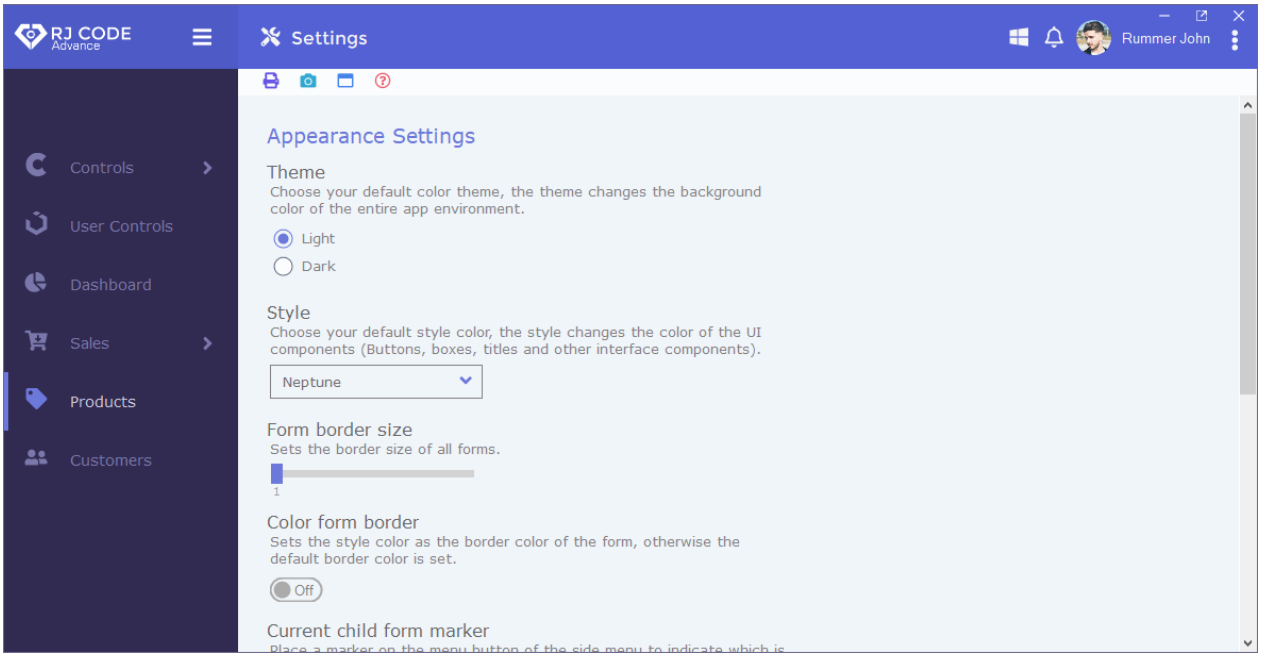
| | |
|---|---|
| `screenshot` | Gets or sets the screenshot to print. |
| `sizeA4` | Gets or sets the size of the document. |
| `imgDocument` | Gets or sets the document in image format. |

### Methods

| | |
|---|---|
| `LandscapeOrientation ()` | Responsible for placing the document horizontally. |
| `PortraitOrientation ()` | Responsible for placing the document vertically. |
| `PrintDocument ()` | Responsible for printing the document. |
| `RJPrintForm (Image _screenshot, string docTitle)` | Constructor with image and title parameters to print. |

#### 9.3.5. Setting Form - RJSettingsForm Class

This class **inherits** from the **RJChildForm** class. Form for application appearance setting, created by designer.

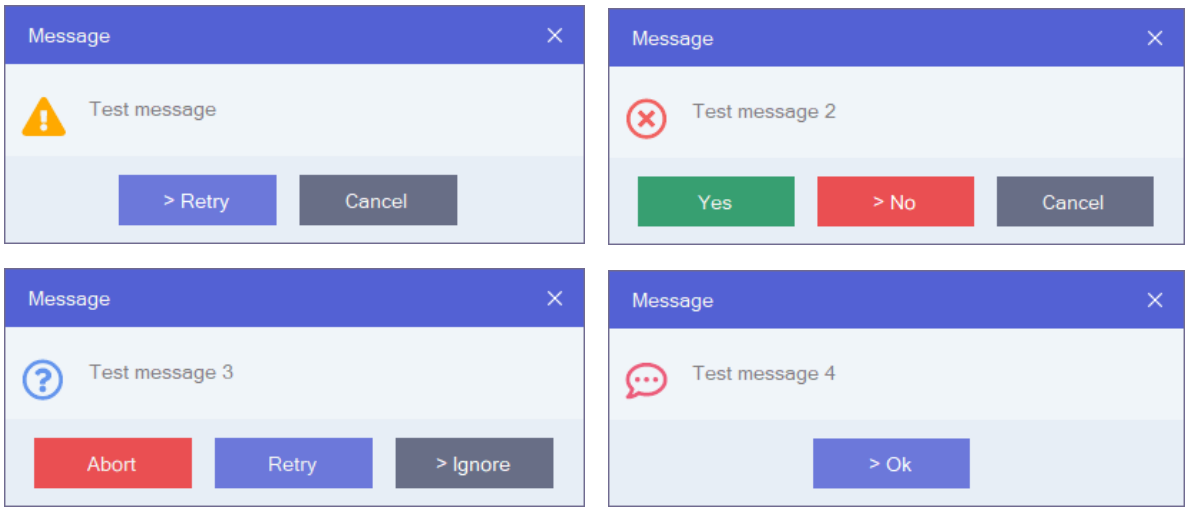



### Methods

| | |
|---|---|
| `LoadApperanceSettings ()` | It is responsible for **obtaining and displaying the current setting data.** |
| `SaveAppearanceSettings()` | **Save the changes made**. |

# 10. CUSTOM MESSAGE BOX

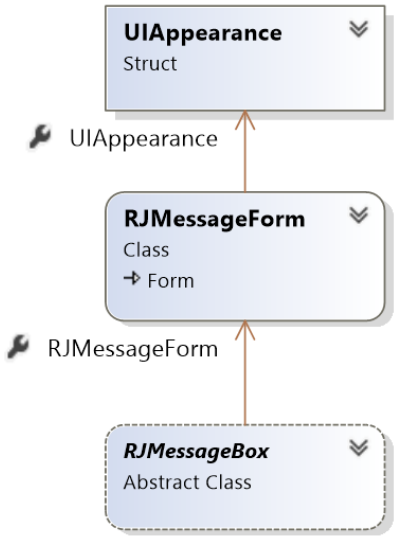
A **custom message box** (RJMessageBox) with **all the common functions of the conventional Windows message box** (MessageBox).



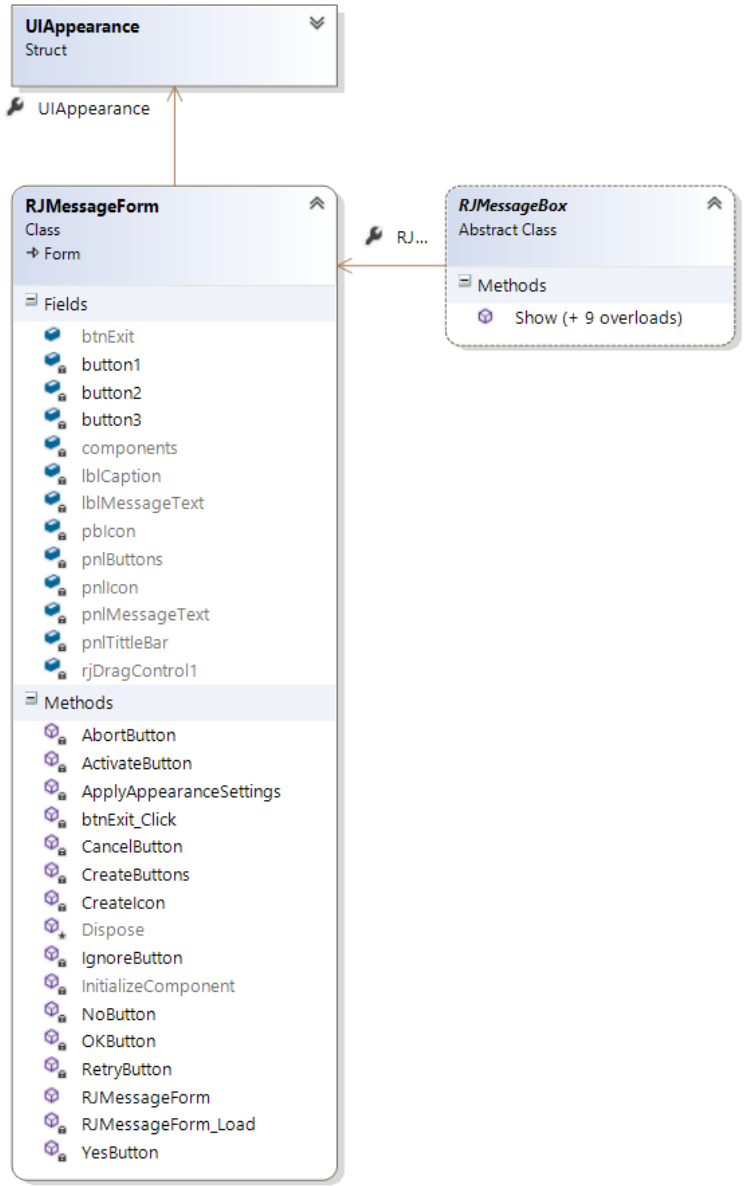

```
//Example 1
RJMessageBox.Show("Test message","Message",MessageBoxButtons.RetryCancel,
MessageBoxIcon.Exclamation);
// Example 2
RJMessageBox.Show("Test message 2", "Message",
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button2);
// Example 3
RJMessageBox.Show("Test message 3", "Message",
MessageBoxButtons.AbortRetryIgnore,
MessageBoxIcon.Information,MessageBoxDefaultButton.Button3);
// Example 4
RJMessageBox.Show("Test message 4");
```

## 10.1. Class diagram

### Collapsed class diagram



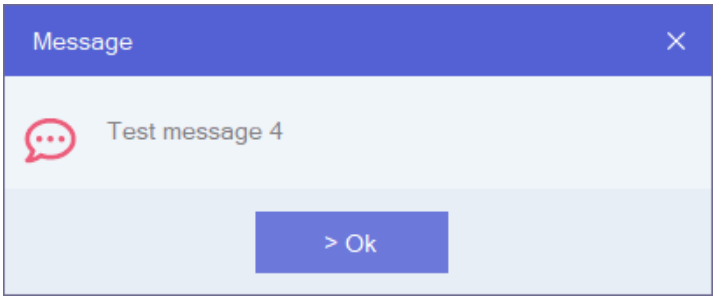
### Expanded class diagram




## 10.2. RJMessageForm class

This class **inherits** from the **Form class** of the **Windows.Forms** library, it was created by the designer.

This form takes care of the **visual representation** (User Interface) of the message box.



## Fields

| | |
|---|---|
| button1 | Button 1 of the message box. |
| Button2 | Button 2 of the message box. |
| Button3 | Button 3 of the message box. |

## Methods

| | |
|---|---|
| ApplyAppearanceSettings () | Responsible for setting the application appearance settings to the form properties. |
| AbortButton (int location, int locationY) | Responsible for creating the abort button. |
| ActivateButton ( MessageBoxDefaultButton defaultButton) | Responsible for selecting (Focus) the default button. |
| CancelButton (int locationX, int locationY) | Responsible for creating the cancel button. |
| CreateButtons ( MessageBoxButtons buttons, MessageBoxDefaultButton defaultButton) | Responsible for creating the specified buttons. |
| CreateIcon ( MessageBoxIcon icon) | Responsible for creating the specified icon. |
| IgnoreButton (int locationX, int locationY) | Responsible for creating the ignore button. |
| | |
| NoButton (int locationX, int locationY) | Responsible for creating the NO button. |
| OKButton (int locationX, int locationY) | Responsible for creating the OK button. |
| RetryButton (int locationX, int locationY) | Responsible for creating the retry button. |
| YesButton (int locationX, int locationY) | Responsible for creating the YES button. |

## 10.3. RJMessageBox class

Abstract class that is responsible for **displaying the custom message box** with the specified parameters (See examples).

I integrated most of the display methods (Show (...)) of the Windows message box, which has 21 overloads of this method, and each one of them returns a DialogResult.

### Methods

| | |
|---|---|
| `Show(string text)` | Displays a message box with the specified text. |
| `Show(string text, string caption)` | Displays a message box with the specified title and text. |
| `Show(string text, string caption, MessageBoxButtons buttons)` | Displays a message box with specified text, title, and buttons |
| `Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)` | Displays a message box with text, title, buttons, and the specified icon |
| `Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, MessageBoxDefaultButton defaultButton)` | Displays a message box with the specified text, title, buttons, icon, and default button. |
| `Show(IWin32Window owner, string text)` | Displays a message box in front of the specified object and with the specified text |
| `Show(IWin32Window owner, string text, string caption)` | Displays a message box in front of the specified object and with the specified title and text. |
| `Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons)` | Displays a message box in front of the specified object and with the specified text, title, and buttons |
| `Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon)` | Displays a message box in front of the specified object and with the specified text, title, buttons, and icon |
| `Show(IWin32Window owner, string text, string caption, MessageBoxButtons buttons, MessageBoxIcon icon, MessageBoxDefaultButton defaultButton)` | Displays a message box in front of the specified object and with the specified text, title, buttons, icon, and default button. |

Well that's it, I hope you liked it and learned more from this written tutorial.

☺

Until next time.