Flexible Modal Dialogs

Version 1.1.1
June 2023

onemanescapeplan@gmail.com

Overview

The **Flexible Modal Dialogs** package gives you a simple, hassle-free system for displaying customizable modal (and non-modal) dialogs with just a few lines of code.
Dialogs, sometimes also known as *alerts*, are small popups which display a message and one or more buttons. A ***modal*** dialog is a type of dialog that dims the app underneath the dialog and forces the user to interact with the dialog before returning to the app. Modal dialogs are widely used in mobile apps and websites. This package supports both modal and non-modal dialogs.

Features

Flexible Modal Dialogs offers the following features:

- Supports modal and non-modal dialogs
- Dialog automatically expands to fit the length of the message
- Support for multiple buttons with customizable labels and functions
- Customize font styles/colors from your code
- Optional icons alongside the message
- Optional checkbox with customizable label (e.g. for "Don't ask again")
- Supports legacy text and Text Mesh Pro

Basic Setup

1. Add the `DialogManager` or `DialogManagerTMP` prefab (the latter uses Text Mesh Pro) `from OneManEscapePlan/ModalDialogs/Prefabs/` to your scene.
2. That's it!

Displaying dialogs

To display a dialog, call the appropriate `ShowDialog()` method on the DialogManager instance:

```
DialogManager.Instance.ShowDialog("Alert", "This is a modal dialog");
```

or

```
GameObject.FindObjectOfType<DialogManager>().ShowDialog("Alert", "This is a modal
dialog");
```

There are many overloaded variations of the `ShowDialog()` method. I recommend exploring these in your IDE. Each version of `ShowDialog()` returns a `UIDialogBase` instance - this is a reference to the newly created dialog. You can use *method chaining* to customize the dialog:

```
DialogManager.Instance.ShowDialog("Alert", "This is a modal dialog")
            .SetCheckboxLabel("I understand")
            .SetLeftIcon(myIcon)
            .SetModal(false);
```

Dialog options and customization

Buttons

By default, a dialog will display a single button labeled "OK" which simply closes the dialog.

An overloaded version of `ShowDialog` allows you to create a two-button dialog with custom button labels and callbacks:

```
DialogManager.Instance.ShowDialog("Two-button dialog", "This dialog has two buttons.", "I
understand", "OK", OnPressedOK, "Cancel", OnPressedCancel)
```

You can also create custom buttons by passing in a collection of ButtonSettings objects:

```
List<ButtonSettings> settings = new List<ButtonSettings>();
settings.Add(new ButtonSettings("OK", OnPressedOK));
settings.Add(new ButtonSettings("Cancel", OnPressedCancel));
settings.Add(new ButtonSettings("Later", OnPressedLater));

DialogManager.Instance.ShowDialog("Three-button Dialog", "The dialog supports a variable
number of buttons with customizable labels. This example demonstrates three buttons.",
settings);
```

The `ButtonSettings` constructor requires two arguments, the button label as a string, and a callback that will be executed when the button is pressed (this can be null). You can optionally pass in a `FontSettings` object as a third argument (detailed below).

Fonts

You can use `FontSettings` instances to control basic font settings (size, style, color, and type face). Only the size argument is required.

```
FontSettings fontSettings = new FontSettings(15, FontStyle.Bold, Color.red);
DialogManager.Instance.ShowDialog("Font settings", "This dialog uses custom font
settings")
            .SetTitleFontSettings(fontSettings)
            .SetMessageFontSettings(fontSettings);
```

Note that FontSettings has two constructors, one for legacy text and one for TextMeshPro.

The `ButtonSettings` constructor can also accept a `FontSettings` argument to control the label font:

```
FontSettings fontSettings = new FontSettings(18, FontStyle.Bold, Color.blue);
buttonSettings.Add(new ButtonSettings("OK", null, fontSettings));
```

Checkbox

You can optionally display a checkbox by setting the checkbox label:

```
UIDialogBase dialog = DialogManager.Instance.ShowDialog("Alert", "This dialog has a
checkbox.", "I understand");
```

or

```
UIDialogBase dialog = DialogManager.Instance.ShowDialog("Alert", "This dialog has a
checkbox.").SetCheckboxLabel("I understand");
```

To find out if the user checked the checkbox, access the `CheckboxSelected` property of the dialog:

```
DialogManager.Instance.ShowDialog("Checkbox Alert", "This dialog has a checkbox.", "I
understand").ClosedEvent.AddListener((UIDialogBase dialog) =>
{
    statusText.text = "Checkbox selected: " + dialog.CheckboxSelected;
});
```

Icons

You can optionally display an icon on either side of the message:

```
DialogManager.Instance.ShowDialog("Icon dialog", "You can set icons on the left and right
side of the message")
            .SetLeftIcon(leftIcon)
            .SetRightIcon(rightIcon);
```

Customizing the prefab

You may wish to further customize the appearance of the dialog by editing the prefab. **I strongly recommend that you do not make edits to the original prefab, because these may be overwritten when you download an update for Flexible Modal Dialogs.** Instead, here is how to create a *prefab variant* which you can safely edit:

1. Drag the `ModalDialog` prefab into a scene
2. Change the name of the dialog GameObject to something that signifies that you changed it, e.g. `CustomModalDialog`
3. Customize the dialog GameObject as you see fit
4. Drag your customized dialog into the "Project" pane of the Editor (preferably into a folder where you keep prefabs for your game). When Unity asks if you want to create an "Original Prefab" or "Prefab Variant", choose "Prefab Variant"
5. Remove the dialog GameObject from the scene
6. When you place the `DialogManager` prefab in a scene, find the "Dialog Prefab" field in the Inspector. Drag your prefab from step 4 into this field to use it instead of the original dialog prefab

The prefab uses a very specific layout configuration to allow the dialog to resize to fit the message length and optional features such as icons and the checkbox. To avoid breaking the dialog layout, take care when editing the prefab:

You can safely edit:

- Colors and textures
- Font settings
- Layout padding
- Layout spacing

You should not:

- Remove any Layout Group component
- Remove any Content Size Fitter component
- Change the settings of "Controls Child Size" or "Child Force Expand"

Troubleshooting

After importing the package, I get an error about an unrecognized method `FindAnyObjectByType()`

This faster replacement for `FindObjectOfType()` was added in Unity 2021.3.18f1. If you have an earlier version of Unity 2021.3, you will get an error message. To fix this issue, I highly recommend updating to the latest version of Unity 2021.3. Alternatively, you can replace `FindAnyObjectByType()` with `FindObjectOfType()`