Easy Modal Dialog

Overview

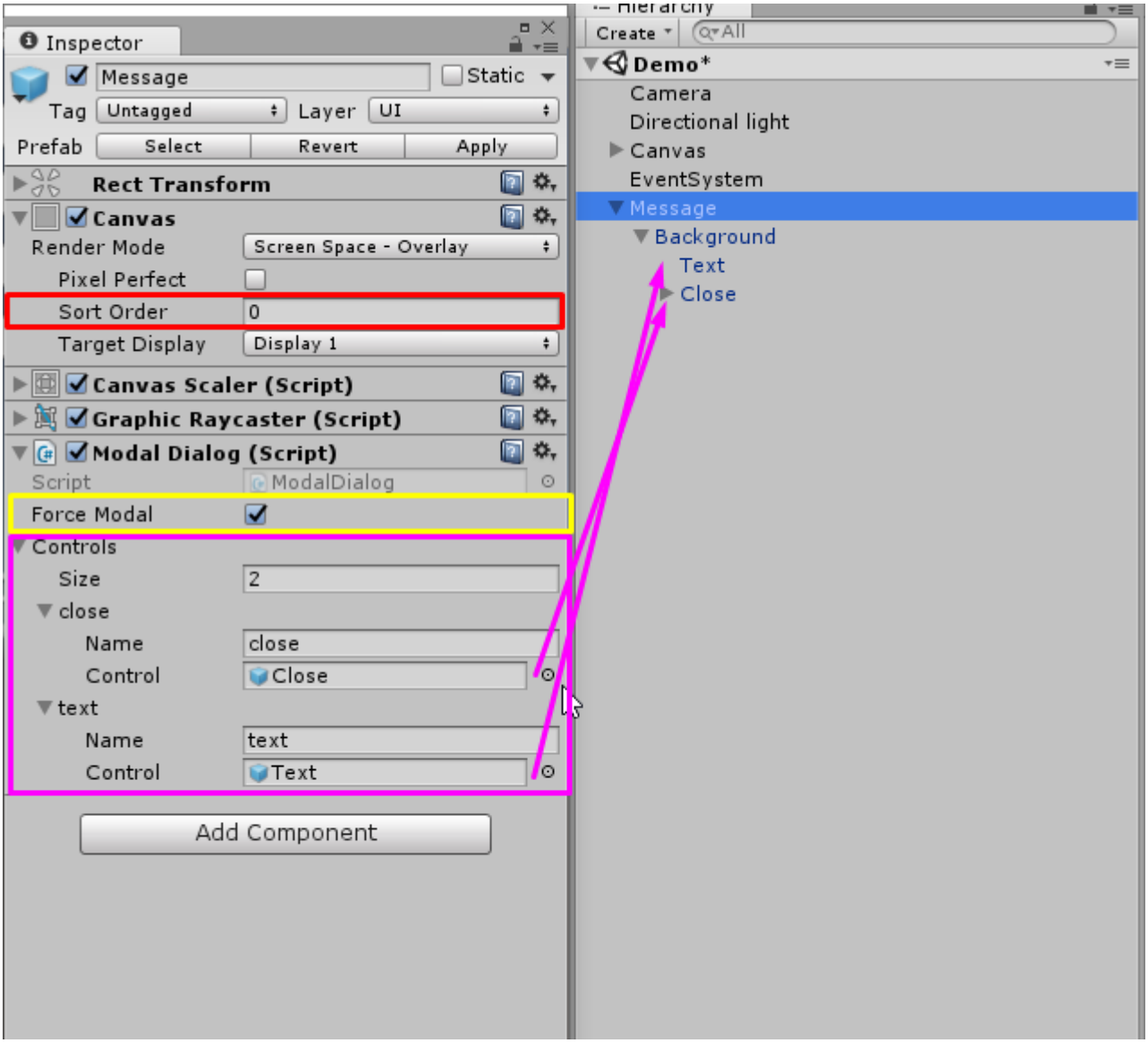
This asset contains some useful scripts to create modal dialogs (via code) based on prefabs you'll create yourself. To help you setup your first modal, I compiled a little guide/documentation for a better understanding on how it works and what is needed on your modal prefab to get it working flawlessly.

Prefab

Each modal dialog has its own prefab, which defines the structure of the dialog and its appearance. The prefab needs to follow some simple patterns, which will help you set your content later when invoking the dialog.

Canvas

Each modal prefab needs a Canvas as its root game object. This is required because you need the sort order of the Canvas component to make it appear over any other UI element on the screen. This root object also needs an instance of **ModalDialog** component, in which you'll add named references to inner game objects for easy access and setup later. On version **1.01** the "Force Modal" option was added to automatically set the Sort Order value for you based on what canvases you have already instantiated on the hierarchy, to make it easier and trully modal.

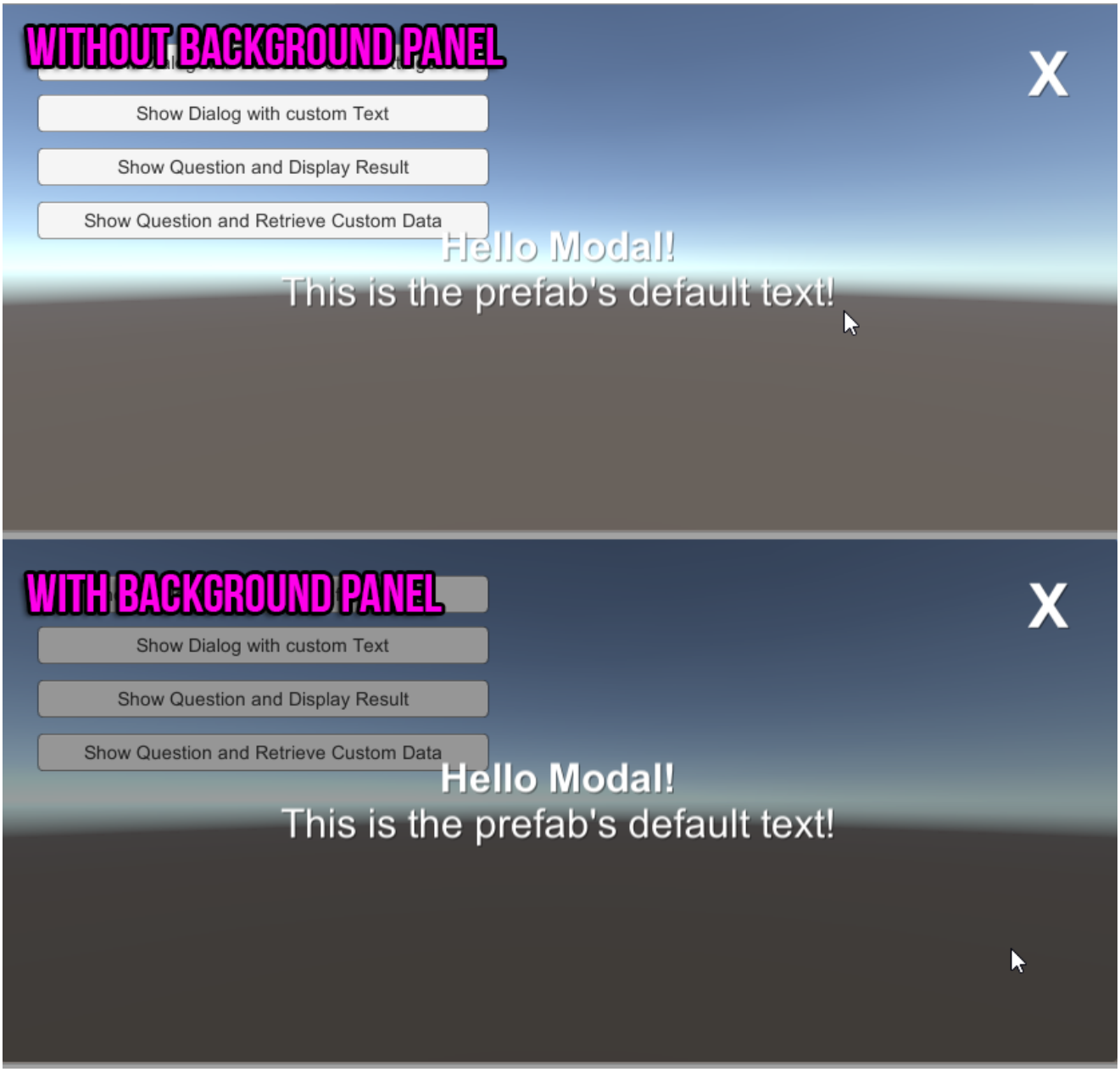


The red box shows where you can set the sort order of the canvas when "Force Modal" is disabled.
The yellow box shows the options where you can toggle the Force Modal option.
The magenta box shows the components that you might want to access and modify later when invoking the dialog (more on this later).

NOTE: You might want to remove the object from your hierarchy after you finish editing your prefab. You don't need it there since it'll be instantiated dynamically later via script. Always remember to apply changes to your prefab after you make any changes.

Background

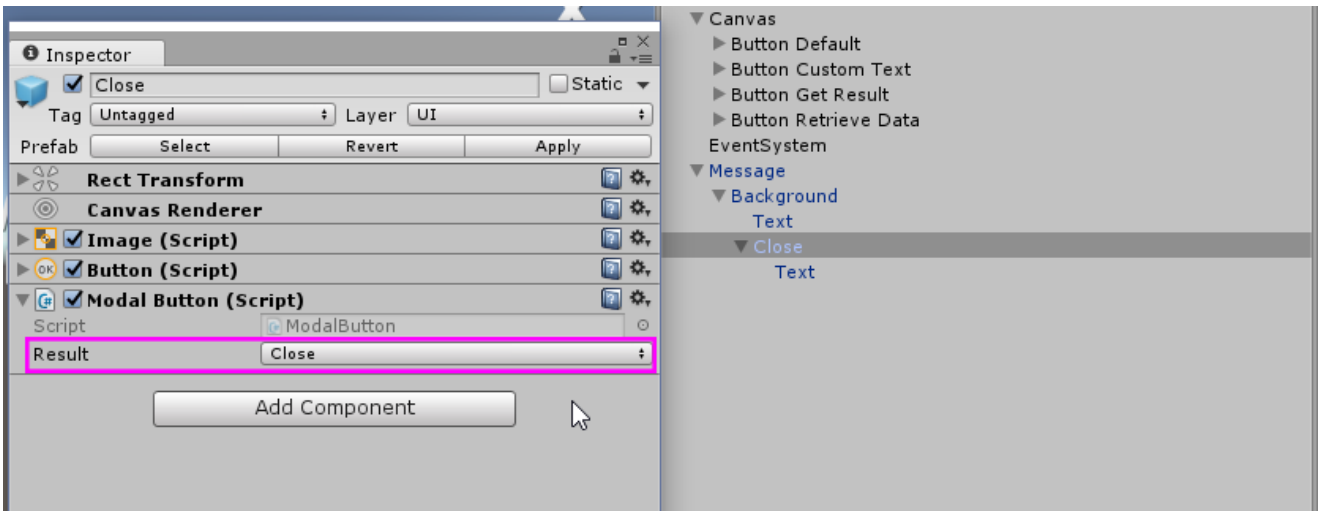
You might want to add a background (panel) to your modal dialog, to block some visibility of the UI elements below the modal dialog. It'll help it appear more like a modal dialog too and not just some floating UI elements.



Buttons

Modal dialogs needs some way to dismiss itself, right? There's a component called **ModalButton** that can be added to Unity's UI Button. The component allows you to set the dialog result when user presses the button.

For example the "X" button on the upper right corner from the last screenshot. That button contains a **ModalButton** with a result set.

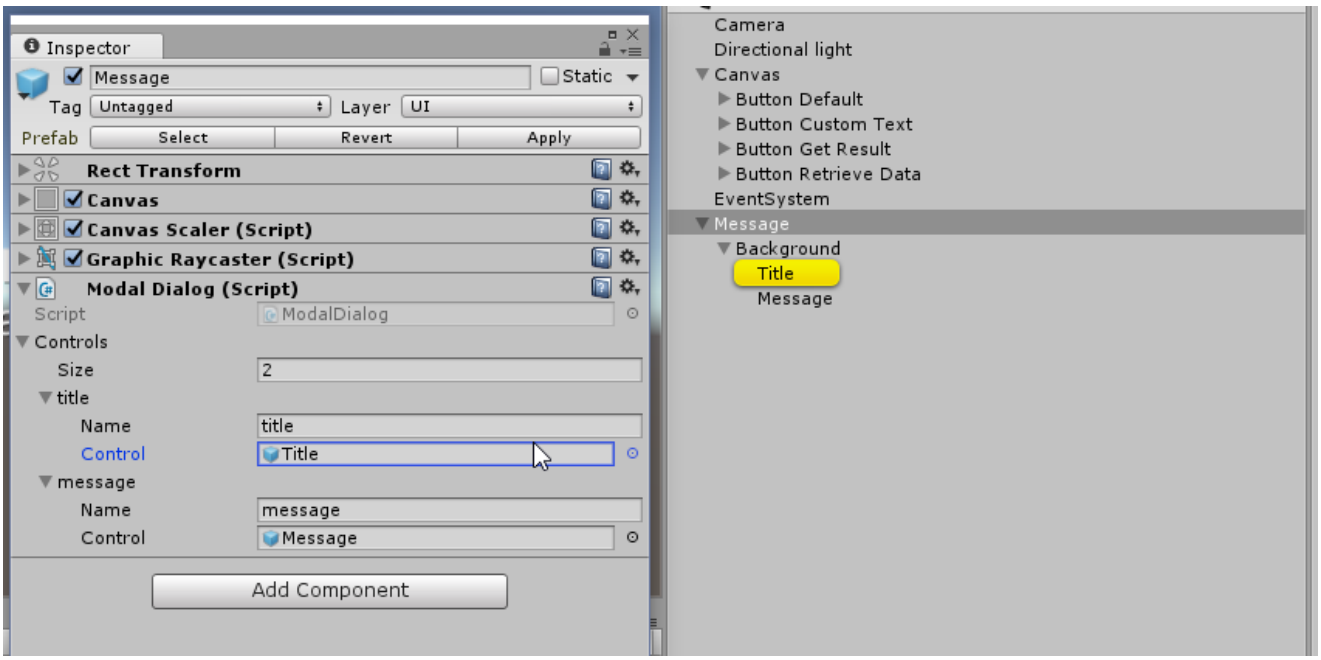


NOTE: The "Close" result is just a logical result which you can verify on script callback (as a user feedback). Any result value you set will actually dismiss the dialog and fire the script callback with that specific result value. The "Close" value is set on the screenshot because it makes more sense since the button is a "close" button on the example.

Control references

If you want dynamic content on your modal dialog (such as custom text), you'll need to set named references on your **ModalDialog** component. Those references will allow a fast access to the objects inside your prefab, and using the fluent API, you'll be able to create and configure the dialog fast and easy.

For example, if you have two **Text** objects inside your dialog, one that serves as the title and one as the message, you'll find yourself setting two references:



You don't need a reference to every object inside your prefab. Just set those for the elements that you might want to change/update when invoking a new instance of the dialog.

NOTE: reference names must be UNIQUE, any duplicated name will cause runtime errors.

API Examples

Invoking a new dialog is easy, and you'll just need a reference to your dialog's prefab and all the dynamic content of the dialog can be set via script using the control reference names you had on your **ModalDialog** component on the prefab's canvas object.

Create

```
ModalDialog.Create(prefab);
```

This will just invoke a new instance of the dialog.

Create and set text content

```
ModalDialog
  // Invoke
  .Create(prefab)
  // Sets the Text.text of the object "title"
  .SetTextText("title", "Buy Item")
  // Sets the Text.text of the object "message"
  .SetTextText("message", "Are you sure you want to buy 500 credits?");
```

This will invoke a new instance of the dialog, and use the reference names to set its control contents.

This example also uses control helpers/extensions:
SetTextText is an extension function that makes it easier to set **Text.text** value.

Without using the extension, this is what you'll find yourself writing:

```
ModalDialog
  // Invoke
  .Create(prefab)
  // Gets a reference to the Text component of the "title" object
  .Control<Text>("title", (Text text) => {
    // Sets the component value
    text.text = "Buy Item";
  })
  // Gets a reference to the Text component of the "message" object
  .Control<Text>("message", (Text text) => {
    // Sets the component value
    text.text = "Are you sure you want to buy 500 credits?";
  })
```

There are extensions to serve as a guide/example on the **ModalDialog.Controls.cs** file.

Create and retrieve dialog result

```
ModalDialog
  // Invoke
  .Create(questionPrefab)
  // Set text
  .SetTextText("text", "Are you a Unity developer?")
  // Callback
  .Result((result) => {
    ModalDialog
      // Invoke another
      .Create(messagePrefab)
      // Set text
      .SetTextText("text", "The answer to the question was: " + result);
    // Dissmis first dialog
    return true;
  });
```

This will invoke a new instance of the dialog, set the message's text and wait for user feedback on the dialog.

You're able to access the result on the callback function. Returning **true** on the callback function will allow the dialog to dismiss itself, otherwise, returning **false** will abort the dialog dismiss process and it will remain open and working.

Create and retrieve dialog data

```
// Stores a reference to the dialog
var dialog = ModalDialog.Create(namePrefab);
// Callback
dialog.Result((result) => {
  // Name variable
  var name = "";
  // Gets a reference to an InputField by its name "name"
  dialog.Control<InputField>("name", (input) => {
    // Sets the name variable value with the input value
    name = input.text;
  });
  // Greetings with another dialog
  ModalDialog.Create(greetingsPrefab)
    // Set the greetings text
    .SetTextText("text", "Hello there, " + modal.GetInputFieldText("name") + "!");
  // Dismiss the first dialog
  return true;
});
```

This will invoke a new instance of the dialog (it has an input field, editable by the user), when user dismiss the dialog, the callback will access the InputField, get the value on the field, and send a hello message with the user's name.

That's basically all you need to know to get it up and running. It's pretty straightforward and easily extensible using class extensions (if you don't want to touch the original scripts)

Contact

If you have any questions, drop me an email at wolfulus@gmail.com.

Thank you!

http://wolfulus.com
https://twitter.com/wolfulus