Toast Me

API Doc.

Release 1.1.1

A notification library for Unity 3d

Getting Started

Components

How To

Overview

This API Doc. was designed to provide and overview of the features provided by the **Toast Me** API and how to use them. This document is a work-in-progress and will continuously be updated.

Installation

Toast Me requires Text Mesh Pro, so using the Unity Asset Store, download and import Text Mesh Pro. Once completed, use the Unity Asset Store to download and import Toast Me.

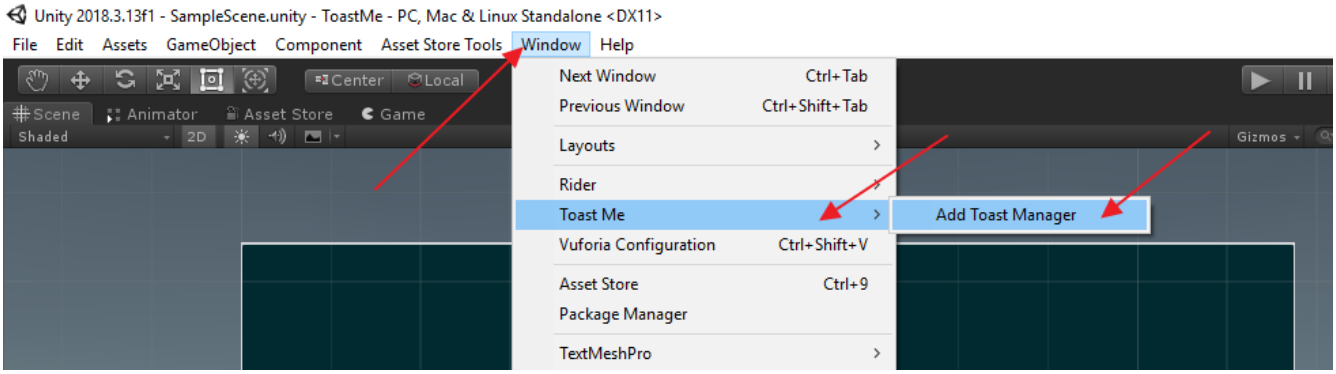
Note: When importing, make sure that all items to import are selected before clicking the import button.

Quick Start

After downloading and importing both **Text Mesh Pro** and **Toast Me**, navigate into the *Scene* directory and open the **SampleScene** and click *Play*. This scene provides the ability to display the six different premade toasts and control where on the screen the toasts will appear.

To use **Toast Me** in your own scene you can start by dragging the **ToastManager** prefab into your *Hierarchy*.

You may also use the new menu item to add the **ToastManager** to your scene.



Toast Me Window menu item.

Support

If any questions arise or you require assistance, please send an email to **toastme_support@over-one.studio**. Our FAQ is under development but is a living document and is certainly available at **https://overone.freshdesk.com.**

Components

Toast (static class)

C#

Queues a toast of the given type with the first-found Toast Manager in the scene.

```
public static ToastInfo Pop (
        string type,
        string message = null,
        string title = null
)
```

type *(string)*
The type of toast to display.

message *(string)*
The text content used to set the message body of the toast.
- Setting this to *null* will prevent the ToastManager from overriding the text content that already exists within the toast prefab.
- Setting this to *string.Empty* will however clear the text content that already exists on the toast prefab.

title *(string)*
The text content used to set the title of the toast.
- Setting this to *null* will prevent the ToastManager from overriding the text content that already exists within the toast prefab.
- Setting this to *string.Empty* will however clear the text content that already exists on the toast prefab.

return *(ToastInfo)*
A reference to the created ToastInfo object.

C#

Queues a toast of the given type using the given Toast Manager reference.

```
public static ToastInfo Pop (
        ToastManager toastManager,
        string type,
        string message = null,
        string title = null
)
```

toastManager *(ToastManager)*
The ToastManager reference to use when displaying a toast of the given **type.**

type *(string)*
The type of toast to display.

message *(string)*
The text content used to set the message body of the toast.
- Setting this to *null* will prevent the ToastManager from overriding the text content that already exists within the toast prefab.
- Setting this to *string.Empty* will however clear the text content that already exists on the toast prefab.

title *(string)*
The text content used to set the title of the toast.
- Setting this to *null* will prevent the ToastManager from overriding the text content that already exists within the toast prefab.
- Setting this to *string.Empty* will however clear the text content that already exists on the toast prefab.

return *(ToastInfo)*
A reference to the created ToastInfo object.

Toast Info (class)

C#

Contains all information to set the content of a toast and to change the way a toast behaves when processed by a Toast Manager.

```
[RequireComponent(typeof(Button), typeof(CanvasRenderer)]
[RequireComponent(typeof(LayoutElement), typeof(EventTrigger))]
public class ToastInfo : MonoBehaviour
{
        …
}
```

Persist *(bool)*
Used out-of-process to prevent this toast from being dismissed by a ToastManager until explicitely set to false.

OnShow *(UnityEvent)*
Used by the ToastManager to get any callbacks to invoke when showing this toast.

OnHide *(UnityEvent)*
Used by the ToastManager to get any callbacks to invoke when destroying this toast.

PopSound *(AudioClip)*
Used by the ToastManager to get the sound to play when showing this toast.

CanDismiss *(bool)*
Used by the ToastManager to determine if this toast can be dismissed.

Fading *(bool)*
Used to determine if this toast is currently in the process of fading-in or fading-out.

Type *(string)*
Used by the ToastManager to Get or Set the type of this toast.

DateCreated *(DateTime?)*
Used to determine when this toast was created.

ToastEffectTime *(float)*
Indicates the time in seconds it takes this toast to show or hide from the ToastEffect.

Destroyed *(bool)*
Used by the ToastManager to check if this toast has already been destroyed.

TitleTextRef *(TextMeshProUGUI)*
The reference to the TextMeshPro text title component of this toast.

MessageTextRef *(TextMeshProUGUI)*
The reference to the TextMeshPro text message component of this toast.

IconRef *(Image)*
The reference to the image component used for the icon of this toast.

BackgroundRef *(Image)*
The reference to the image component used for the background of this toast.

TitleTextColor *(Color?)*
Used to Get or Set the title text color of this toast.

MessageTextColor *(Color?)*
Used to Get or Set the message text color of this toast.

BackgroundColor *(Color?)*
Used to Get or Set the background color of this toast.

Icon *(Sprite)*
Used to Get or Set the icon of this toast.

IconColor *(Color?)*

Used to Get or Set the icon color of this toast.
Title *(string)* Used to Get or Set the title text of this toast.
Message *(string)* Used to Get or Set the message text of this toast.
TapToDismiss *(bool)* Used to Get or Set the ability to manually dismiss this toast with a mouse click.
Lifetime *(float)* Used to Get or Set the lifetime this toast will remain visible.
PreventDismissal *(function)* Prevents this toast from being destroyed. This is called when the mouse *OnHover* event is triggered.
EnableDismissal *(function)* Allows this toast to be destroyed. This is called when the mouse *OnLeave* event is triggered.
FadeIn *(function)* Called by the ToastManager to begin a fade-in effect on this toast.
FadeOut *(function)* Called by the ToastManager to begin a fade-out effect on this toast.
Destroy *(function)* Used by the ToastManager to destroy this toast. This is also called when a user click this toast to manually dismiss this toast.

How To

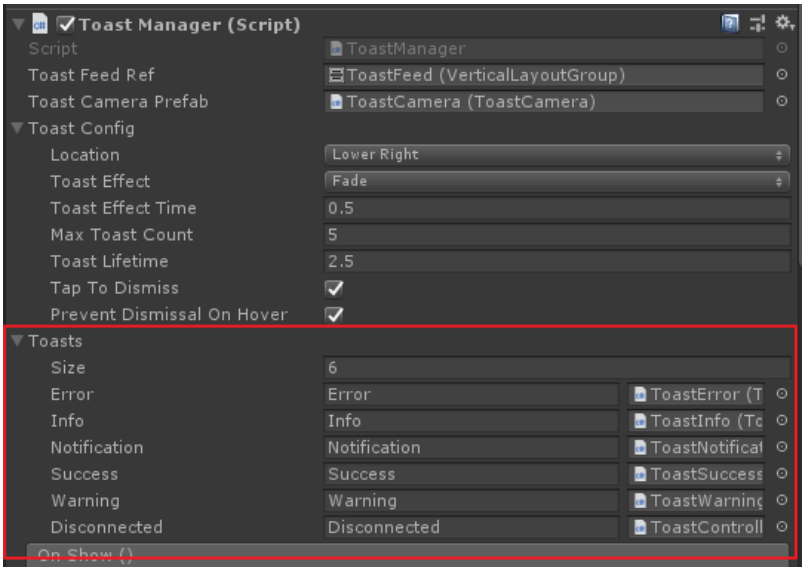
Display Toast

Requirements
1. **Toast Me** installation has been completed. 2. **Toast Me** setup has been completed.

The most common approach to display toasts using the Toast Me API is by leveraging the static class Toast. This class exists within the ToastMe namespace and provides two different public static methods Pop (…).

The Pop(…) method(s) are used to queue a toast of the given type with the Toast Manager to be processed and displayed.

In order to display a toast, your ToastManager GameObject must first have a reference to the toast you want to display. There exists a Toast collection that can be expanded/shrunk from within the Inspector window. The default Toast Manager provided should contain a reference to the six pre-made assets.



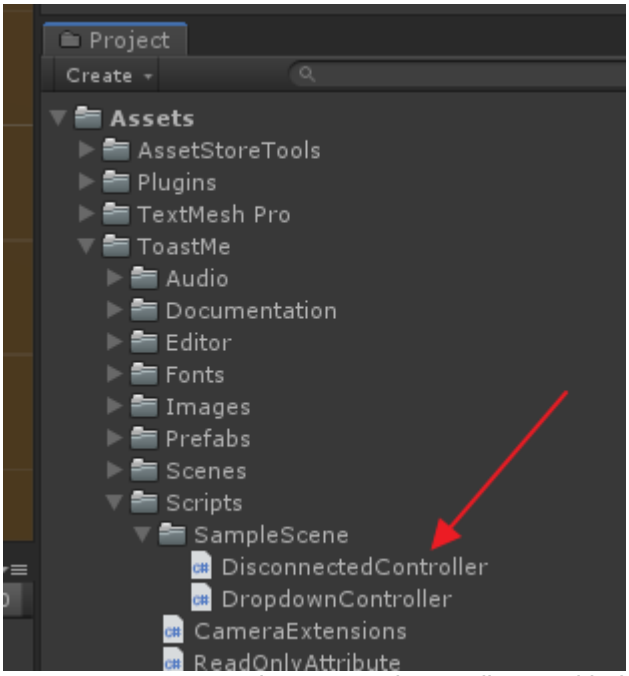
Toast Manager script highlighting Toasts collection.

After storing a reference to your toast prefab, you must supply it a *key*. The value you use is a unique identifier that will be used by the Toast Manager to instantiate your toast prefab during run-time. The pre-made toasts use the key(s) *Error, Info, Notification, Success, Warning* and *Disconnected*.

Once a toast prefab has been registered and provided a unique key, we can write some code to leverage one of the static Pop(…) functions in our application.

An example can be seen within the *DisconnectedController.cs* script that can be found at the location below from within your Project window. Please see the section under **Components** to see a details explanation of the two different Pop(…) functions available and how the behave.

Assets/ToastMe/Scripts/SampleScene/DisconnectedController.cs.

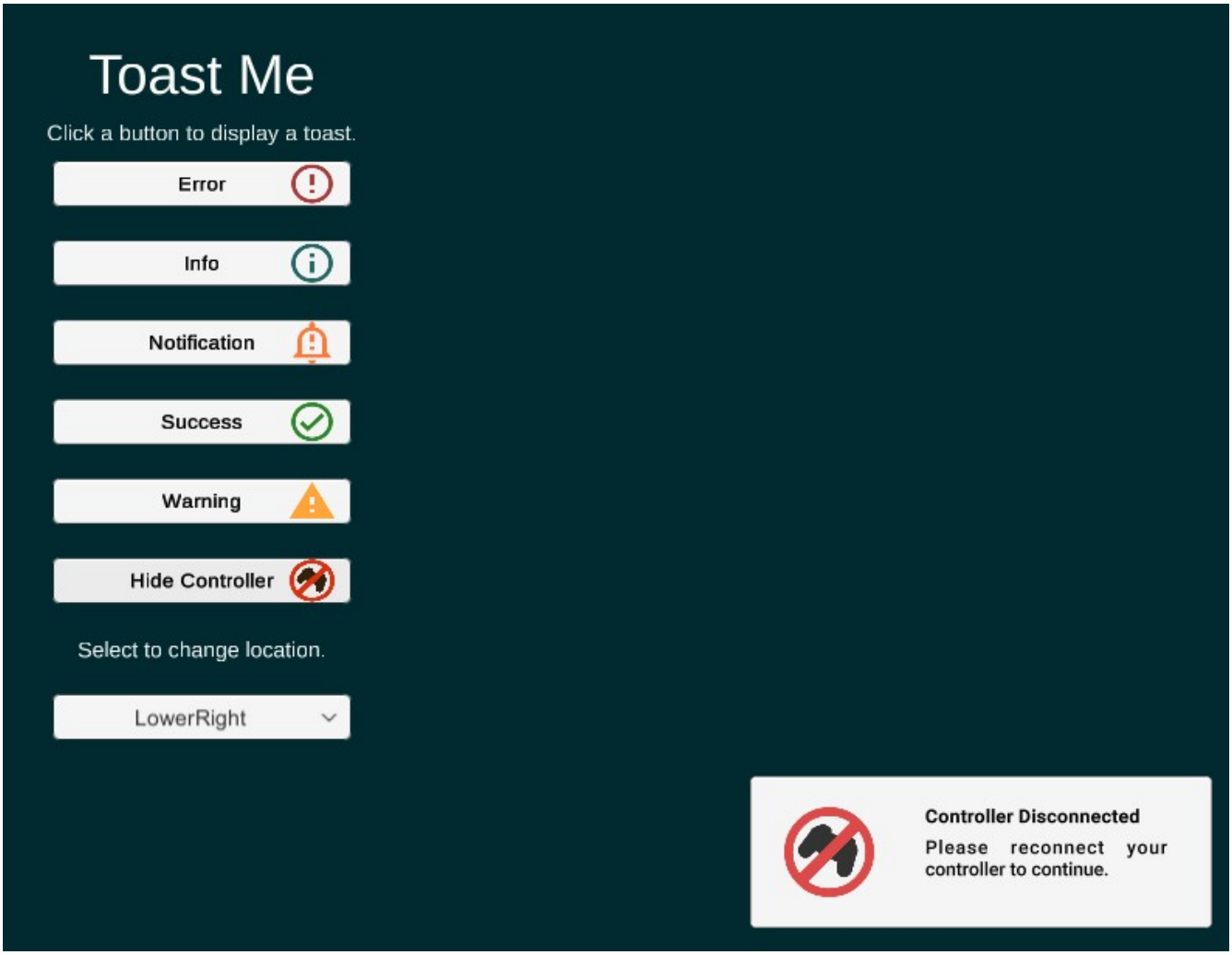



```
Joshua J Allen
public void Toggle()
{
    var prevVisible = _isVisible;

    _isVisible = !_isVisible;

    if (!prevVisible && _toastInstance == null)
    {
        _toastInstance = Toast.Pop(ToastType);
        _toastInstance.Persist = true;
    }
    else
    {
        _toastInstance.Persist = false;
        _toastInstance = null;
    }
}
```

DisconnectedController.cs with the Toggle() method calling Toast.Pop(...)

When calling the Pop(…) method during run-time, the minimum parameters required is the *type* parameter. This string parameter should be the *key* you gave your toast when you added the toast to the Toast Manager. Doing so will search the current scene for a Toast Manager and will request it to instantiate and display a toast of the given type.

Toast being displayed through the Toast.Pop(…) API.



Create Custom Toast

Requirements
1. **Toast Me** installation has been completed. 2. **Toast Me** setup has been completed.

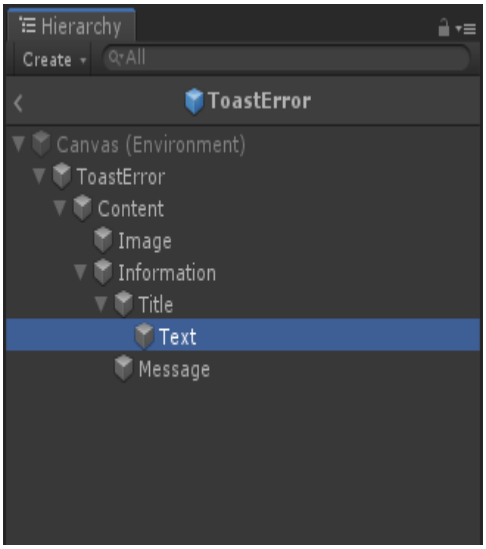
A toast in its basic form is a Unity button with a **ToastInfo** component attached to it. The Toast Info component maintains a reference to the **TextMeshProUGUI** components used for the *Title,* and *Message* with additional reference to the **Image**, used for the toast *Icon*.

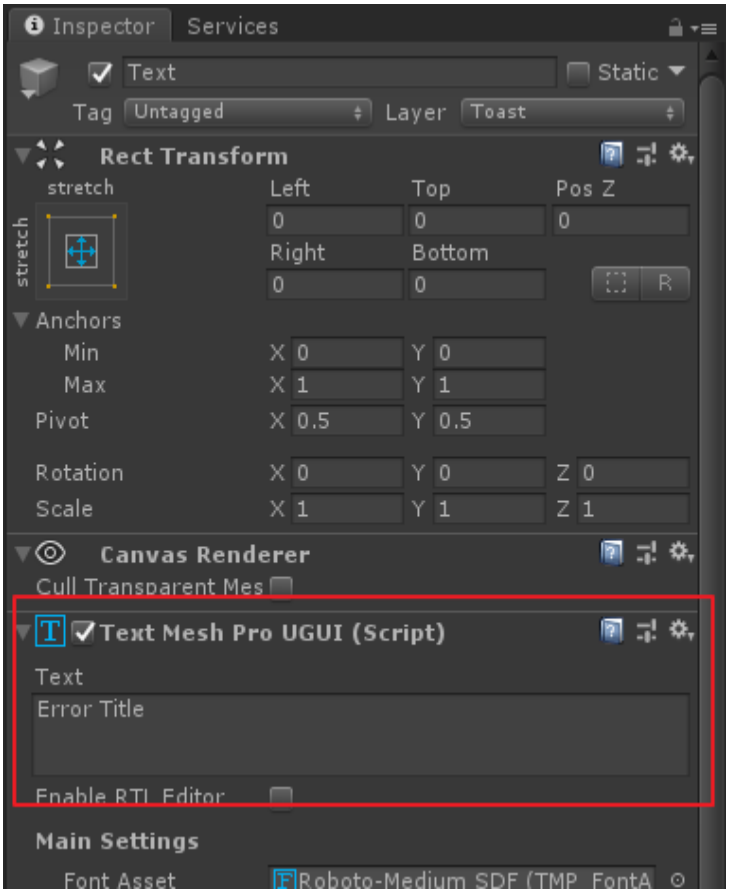
Toast prefab

1. Toast Title
2. Toast Message
3. Toast Icon

The best way to begin creating a custom toast is to leverage one of the pre-made prefabs. Using the Unity Inspector, you can edit the text content for the *Title,* and *Message* just as you normally would when editing text. You can even use the Unity Inspector to swap out the image used for the toast *Icon*.

Keep in mind that the **Toast Info** contains the references to the toast components you are editing within the inspector and provides an intuitive API that allows you to also manipulate the toast content during run-time. See the **Components** section for more information.





Hierarchy and Inspector screenshots from Unity