Runtime Text Edit Documentation

Release 1.0 - 26.05.2020

If you are reading the PDF documentation, here is the most up to date online version:

https://docs.google.com/document/d/16Km-eYJfPQjIAPnDcXKNJyUSSdmHS2zAcmyzs79KEJU/edit?usp=sharing

Gliding Squirrel - www.glidingsquirrel.com

Outline

Overview

Runtime Text Edit makes it possible to edit text directly in Unity's Scene view during Play mode. Changes can be persisted so that they are kept when restarting.

This asset targets Unity creators who want to perform many edits to text. The goal is to make the process more efficient and enjoyable. Seeing a text that you want to change and executing that change should be as frictionless as possible.

Supports **UI text components (Unity & TextMesh Pro)** and **3D text components (Unity & TextMesh Pro)**.

Rich text editing: While editing you can toggle if rich text tags are displayed or processed.

For an interactive demonstration, open Unity scene **InteractiveDemo.unity** at "RuntimeTextEdit/Examples/Scenes".

RuntimeTextEdit supports **rich text** editing.

RuntimeTextEdit is **not** capable of **persisting text formatting**.

Disclaimer: The intention of RuntimeTextEdit is to make it easy and fast to permanently change text during runtime. When a change is made there is no undo and restarting will not revert it. Please make sure that you have a backup of your data in case your text is accidentally overwritten.

Addressed Issues

When a Unity creator is in Play mode and notices a text he/she wants to change, the process could suffer from the following issues:

- The creator needs to locate the place where he/she can edit the text. This could be the inspector of a Unity text component, some place inside a file (JSON, XML, text), a member of a scriptable object, … He/she might search for a while.
- The creator stops Play mode, changes the text and wants to see it in Play mode again to verify the change. He/she might need to spend a long time bringing the application state back to where it was for the text to be displayed again.
- The text is edited outside of the Unity Editor but it is important that it fits into the context of the rendered scene. The creator has to go back and forth between the text editing environment and Unity multiple times to get it right.

RuntimeTextEdit offers a solution by letting the creator edit text directly in Play mode at the location where the text is displayed.

Does it fit me and my project?

Consider Runtime Text Edit when ...

You work on a text heavy project and expect that a lot of it will need to be edited.

The ability to directly perform edits during play mode might speed up your workflow when editing text. Small edits like fixing some wording or a spelling mistake can be done in a few seconds.

You enjoy using tools that make it fast when going from something you intend to do to executing it.

You want to integrate Runtime Text Edit into an existing project where text is saved in any kind of file.

Runtime Text Edit is designed in a way to be able to work with any kind of text file. Writing and reading to these files is not done by Runtime Text Edit, it will just tell you what to write and you implement it.

Consider other options when …

You need a solution that does not require writing a little C# code

Most of the coding is already done with RuntimeTextEdit. However, a small portion has to be implemented by the user. RuntimeTextEdit comes with example code that you can reference for your project.

You do not have the option to click on elements on the screen (Example: when the mouse is captured for camera movement)

RuntimeTextEdit requires that you click on the text you want to edit. If that is not possible then you cannot use RuntimeTextEdit or have to design your own workaround.

You need a solution that works outside the Unity Editor environment

RuntimeTextEdit is targeting Unity developers and not end users. While it also works in a deployed build, it was not designed to be used that way. You may still use it and modify it to fit your needs.

You want to integrate Runtime Text Edit into an existing project where text is saved in ScriptableObjects or inside text components.

If you do not save text in some form of file but instead in a ScriptableObject or in the text component itself, the effort for integrating Runtime Text Edit into your project is a bit higher. The reason for that is, any change to text inside a ScriptableObject or inside a text component during runtime is going to be reverted when Play mode is stopped. In order to use Runtime Text Edit you will have to save your text inside a file and load it at startup. Depending on the size of your project, changing everything to use files instead might take some effort.

Dependencies

TextMeshPro version 2.0.1

Unity version 2019.3
(earlier versions do not work and are not supported as of yet)

Getting started

Downloading and importing

Import TextMesh Pro Essential Resources in the Editor under “Window/TextMeshPro/Import TMP Essential Resources”. Do this before importing Runtime Text Edit.

Download and import Runtime Text Edit from the Unity Asset store.

The Folder *Examples* contains optional files, which are not necessary for RuntimeTextEdit to work but offer examples to get familiar with RuntimeTextEdit.

Demo scene

If you are looking for an interactive demo that shows you how RuntimeTextEdit can be used, open the scene **InteractiveDemo.unity** under *RuntimeTextEdit/Examples/*. This is recommended if you are new to Runtime Text Edit.

If you wish to set up RuntimeTextEdit in your own project, proceed with the next chapter.

Setting up RuntimeTextEdit in your own project

Add a physics raycaster component to your main camera

Find your main camera GameObject and add a **PhysicsRaycaster** component to it. Make sure it is not a Physics**2D**Raycaster component.

This component is used to react to mouse click events and figure out which text was clicked.

Add a RuntimeTextEditManager

Add an empty GameObject (location in hierarchy not important) and add the Script **RuntimeTextEditManager.cs** to it.

RuntimeTextEdit requires a manager component being present in your scene. The manager keeps track of all RuntimeTextEdit instances, which we will add in the next step. There shall always be **exactly one** manager in the scene.

Add RuntimeTextEdit instance(s)

Locate a text component in your Unity Hierarchy that you want to edit with RuntimeTextEdit. This could be a UI or 3D text component, see section supported text component types. Add the script **RuntimeTextEdit.cs** to the same GameObject.

Do the same with all other text components that you wish to edit using RuntimeTextEdit.

Start editing

In Play mode, press and hold right CTRL on your keyboard and click on the text you want to edit. Try to select a word. You should now be able to edit your text. While editing, click outside the text area to confirm your changes or press ESC to discard your changes.

Your edits will change the text during runtime but these changes will not persist yet once you stop and restart the application. There should also be a warning about that in your console. This will be set up in the next section.

If you cannot edit the text, see chapter troubleshooting.

Define callback method for persisting text

RuntimeTextEdit is **not** taking care of saving your text somewhere so that it persists when restarting the application. This can be done in many ways: A text file, a JSON file, a XML file, a whole dialogue system and so on. Persisting text has to be implemented by you. That way you are free to choose a solution that suits your project best.

An important note: Changes made to the text inside the inspector of a text component during runtime will be reverted back when restarting the application. The same is true for Unity ScriptableObjects, which will revert back when restarting the Unity Editor. It is not possible to change these during runtime from a script and make the change persistent. Use files instead as your method for saving text by reading the file content at application startup and write it into the text component.

Create a script that loads text from your source of choice (file, dialog system, ...) into your text component at startup. See folder “Examples/Scripts/LoadSaveText_...” for example implementations. Attach this script to the same GameObject as RuntimeTextEdit. Check to make sure the text loads correctly.

Add **using RTE;** to the top of your script to import the namespace of RuntimeTextEdit.

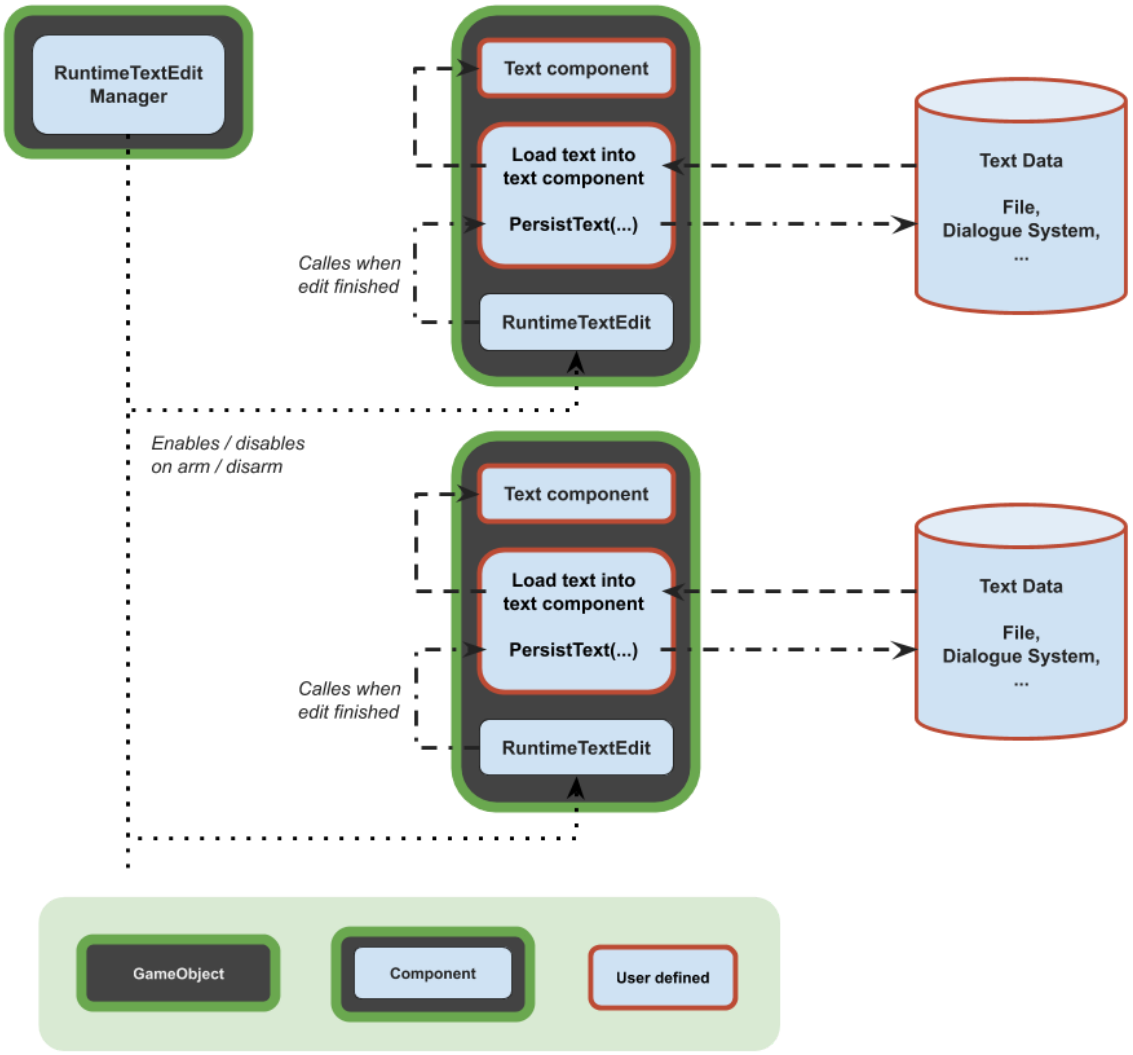
Make the class inside your script implement the interface *RuntimeTextEdit.TextEditCallback* and its method **public void PersistText(string newText, int textCompID)***. PersistText(...)* will be called by RuntimeTextEdit when this text component was edited. Inside this method, make sure to save *newText* to where the text originated from.

Side note: Parameter **textCompID** tells you the ID of the text component and can be used to figure out which text component was edited.

Check if edits are persisted correctly by making an edit and restarting Play mode.

These are the basics of how to get RuntimeTextEdit to run in your project. Once this works you can dive further into configuration scenarios and customizations.

How RuntimeTextEdit is structured



Configuration scenarios

Implementing interface TextEditCallback on a different GameObject than the text component

Per default RuntimeTextEdit will search for the implementation of interface TextEditCallback on the same GameObject. In some scenarios the user might want to implement this interface on another GameObject.

This can be done but RuntimeTextEdit needs to know in which GameObject to search for the implementation. In the RuntimeTextEdit inspector, set the property TextEditCallback to link to that GameObject.

When this configuration is used it is also possible to let multiple text components call the same TextEditCallback implementation. This is useful when there is a script that handles multiple text components at the same time. When the method *PersistText(string newText, int textCompID)* is called RuntimeTextEdit will pass the text component ID with *textCompID*. That way, you can differentiate inside PersistText(...) which text component was changed.

Editing text that reacts on a mouse click & preventing an action on a mouse click

Clicking on a text in order to initiate RuntimeTextEdit might cause something to happen in your code. This could be unwanted.

Here are some more concrete examples:

- Imagine you have a button in your project and you want to edit its text without triggering the button action.
- Imagine you have a GameObject that you set up to react to mouse clicks and it should not be triggered when trying to edit text.
- Imagine you set up your input so that a mouse click triggers some event in general which should not happen when trying to edit text.

For these scenarios RuntimeTextEdit offers two custom events: **OnArm** & **OnDisarm.** You can add callback methods when these events happen in the inspector of RuntimeTextEdit.

RuntimeTextEdit is in the **Armed** state while the ArmButton (default right CTRL) is pressed. And **Disarmed** if not.

You can deactivate components or parts of your code that should not happen while trying to edit text in **OnArm** and reactivate them in **OnDisarm**.

For a button it could be done like this: Disable the button component in **OnArm** and enable it in **OnDisarm.**

Editing text that moves, disappears or changes in some other manner over time

Editing text could be difficult or impossible if the text moves, disappears or changes while you are editing.

Here are some more concrete examples:

- Imagine you have a dialog in a video game where each line is displayed for some time and then gets replaced by the next line.

- Imagine a text that is not on a fixed position on the screen. Either because it moves or the camera moves.

For these scenarios RuntimeTextEdit offers two sets of custom events: **OnArm/OnDisarm** and **OnEnterEditMode/OnLeaveEditMode.** Either of those two could be used. You can add callback methods when these events happen in the inspector of RuntimeTextEdit.

RuntimeTextEdit is in the **Armed** state while the ArmKey (default right CTRL) is pressed. And **Disarmed** if not.

RuntimeTextEdit is in **EditMode** state while the user can edit text.

You can stop time, or stop physics simulation or stop advancing dialog text in **OnEnterEditMode** and resume it in **OnLeaveEditMode**. That way, the text you edit stays where it is while you edit it.

Or you can use **OnArm & OnDisarm** instead to make this happen as soon as the ArmButton (default is right CTRL) is pressed.

Instantiate components during runtime

Instantiating RuntimeTextEdit & RuntimeTextEditManager components during runtime is possible. Please make sure to instantiate the RuntimeTextEditManager before instantiating RuntimeTextEdit components.

Customization

Changing the ArmKey

RuntimeTextEdit is activated while holding a key on your keyboard. The default is right CTRL. You can change the key to something else in the inspector of the RuntimeTextEditManager component.

Rich text

Rich text editing is supported. The ArmKey (default is right CTRL) has a double function while editing text and will toggle if rich text tags are shown or processed.

Supported text components

- Unity UI text (UnityEngine.UI.Text)
- TextMesh Pro UI text (TMPro.TextMeshProUGUI)
- Unity Text Mesh (UnityEngine.TextMesh)
- TextMesh Pro text (TMPro.TextMeshPro)

Troubleshooting

Clicking on text does nothing

First, make sure to check the console for warnings. If there are none, proceed.

Make sure that you set up RuntimeTextEdit in the correct way. Refer to Setting up RuntimeTextEdit in your own project.

RuntimeTextEdit uses raycasts to check which text was clicked. A collider might be in the way between the camera and your text when Unity performs a raycast. Go to Unity's Scene view and search for colliders that might block a raycast reaching the texts collider.

Rendered text does not change after an edit

First, check the console for warnings. If there are none, proceed.

Pressing ESC while editing will revert changes back to the state before editing. In order to commit your changes you have to click outside the text component.

It could be that Runtime Text Edit changed the text component text successfully but it was overwritten by some other behaviour afterwards. This can happen if you set your text component text inside an Update() function. Check if this could be the case.

A mouse click in order to initiate Runtime TextEdit causes something to happen in my code which should not happen.

See section Editing text that reacts on a mouse click & preventing an action on a mouse click on how to set up RuntimeTextEdit in a way to prevent that.

I do not have a right CTRL key on my keyboard or it is used for some other function already.

You can select a different trigger key inside the inspector of RuntimeTextEditManager.cs.

RuntimeTextEdit console messages

Text component [...] is not a raycast target. Clicking on this text component will not trigger RuntimeTextEdit.

A text component can be configured in its inspector if it should be a target during raytracing or not. If not, clicking on the text will not trigger an event for RuntimeTextEdit. Therefore, every text has to be configured as a raycasting target.

Go to the text component and enable the option **Raycasting Target**.

Could not find RuntimeTextEditManager. Please add a RuntimeTextEditManager component to your scene.

A single instance of RuntimeTextEditManager is required in every scene, where you wish to use RuntimeTextEdit.

Solution: Add a GameObject to your scene at any position in the Hierarchy and add a RuntimeTextEditManager component to it.

Could not find text component in GameObject: [...]

A RuntimeTextEdit component will search for a text component in the same GameObject. This log is displayed when RuntimeTextEdit could not find one.

RuntimeTextEdit could be on the wrong GameObject.

The text component might not be supported. See section supported text components.

There appears to be no PhysicsRaycaster component on the main camera [...]. Please add one!

A PhysicsRaycaster component is needed by RuntimeTextEdit to determine which text was clicked. Determine your main camera and add a PhysicsRaycaster component to it.

Text components text does not match input fields text. Maybe some other component changed the text component in its Update() method.

An instance of RuntimeTextEdit modified a text component but the changes were overwritten by another component while editing the text. This could mean that the new text will not show up.

A possible cause for this could be a script that sets the text of the text component in its Update() method every frame. You might want to consider setting the text of the text component once at the start using method Start().

Found more than one folder named RuntimeTextEdit in the project. Could not locate package path. Please rename all folders called RuntimeTextEdit that do not contain the RuntimeTextEdit package.

Text component is missing a collider. Needed for RuntimeTextEdit to work. Added a box collider.

Text components with a 3D position in the world need a collider so that RuntimeTextEdit can detect which text component was clicked.

Searched for implementation of interface TextEditCallback on GameObject [...] but failed to find it. Could not call PersistText(...). Set inspector property TextEditCallback in RuntimeTextEdit.

Runtime Text Edit calls a method called PersistText(...) when the user finished editing a text. It searched on the specified GameObject for an implementation and found none.

While in Play mode check the field TextEditCallback inside the inspector of the RuntimeTextEdit component. The GameObject there will be the location where Runtime Text Edit searches for a script that implements TextEditCallback.PersistText(...). Make sure that an implementation exists there.

FAQ

What is meant by persisting text?

Your text will most likely exist in two places: The first is **in runtime memory.** A change will stay while your application runs but will not persist when you restart your application. The second is **on your hard drive.** This text gets loaded during runtime, sometimes only once at the start. “Persisting text” means writing a change to your hard drive so that a change will persist when you restart your application.

Why are TextEdit components disabled when entering Play mode?

TextEdit components are disabled per default at startup because otherwise they would interfere with mouse events that you react to in your code. The manager will activate all TextEdit components while the ArmKey is pressed. That way, RuntimeTextEdit is basically turned off as long as the ArmKey is not held down.

Why does a green text input field appear for some text components?

The green text input field is used for 3D text components, meaning text that is not UI text and has a 3D position. There is no in-place text input for 3D text yet.

Is rich text supported?

Yes! During editing the ArmKey serves a dual purpose and can be pressed to toggle if rich text tags should be shown in text or applied.

Will there be support for Unity versions below 2019.3 ?

As of right now, earlier versions are not supported and will not work for the most part.

Please let me know if you would be interested in this.

Scripting Reference

RuntimeTextEdit.cs

Namespace: RTE

Inspector properties

Text Edit Callback: Reference to a GameObject with a component/script that implements interface *TextEditCallback.* RuntimeTextEdit will call *PersistText(string newText, ...)* in that component when an edit was made with *newText* being the new text.
Default value: If no GameObject is linked here, RuntimeTextEdit will search in its own GameObject for a component that implements *TextEditCallback.*

OnArm: Callback methods that are triggered when RuntimeTextEdit goes from state Disarmed to Armed. RuntimeTextEdit is in the Armed state while the ArmButton (default right CTRL) is pressed. And Disarmed if not.

OnDisarm: Callback methods that are triggered when RuntimeTextEdit goes from state Armed to Disarmed. RuntimeTextEdit is in the Armed state while the ArmButton (default right CTRL) is pressed. And Disarmed if not.

OnEnterEditMode: Callback methods that are triggered when RuntimeTextEdit goes into edit mode. RuntimeTextEdit is in edit mode while there is a text input field spawned by RuntimeTextEdit.

OnLeaveEditMode: Callback methods that are triggered when RuntimeTextEdit comes out of edit mode. This happens when the user stops editing.

TextEditCallback interface

Namespace: RTE
Defined inside: RuntimeTextEdit

Method: void PersistText(string newText, int textCompID)
This method is called by RuntimeTextEdit when the user has finished editing text. Inside this method the user should save *newText* to wherever it is stored (file, dialogue system, ...) in order for the change to be persistent. When multiple instances of RuntimeTextEdit.cs could have called this method, check parameter *textCompID* to differentiate which text component was modified.

RuntimeTextEditManager.cs

Namespace: RTE

Inspector properties

Arm Key: This is the key on your keyboard that when held is setting RuntimeTextEdit into the Armed state. It also serves as the toggle key for showing rich text tags while editing text.

Feedback & Questions & Support

Hopefully, this documentation and the examples answer all of your questions.

If you have used RuntimeTextEdit in your project I would love to hear about that. :)

You can get in contact with me via: LarsTheGlidingSquirrel@gmail.com

For support requests please add FREE or PAID to the subject of your email depending on your version. Thank you!