Simple Ordering

v 1.3

KUPIO

Table of Contents

Important!

Once you have viewed the examples in this asset, you will need to delete the Examples folder and its contents.

This is because Unity will see the example code as part of your app when they add themselves to your script execution order.

Introduction

Simple ordering allows you to control your script ordering automatically by declaring dependencies directly in your code.

By adding simple attributes to your `MonoBehaviour` scripts, the ordering will be automatically calculated. This carries the advantage that any circular dependencies in your architecture will be exposed, and makes script ordering easy to change.

How to declare dependencies

Simply set up dependencies using the `RunAfter` and `RunBefore` attributes.

```
using com.kupio.declarativeorder;

public class ScriptA : MonoBehaviour {
    void Update() {
        /* ... */
    }
}

[RunAfter(typeof(ScriptC))] ❶
public class ScriptB : MonoBehaviour {
    void Update() {
        /* ... */
    }
}

[RunBefore(typeof(ScriptA))] ❷
public class ScriptC : MonoBehaviour {
    void Update() {
        /* ... */
    }
}

[RunAfter(typeof(ScriptA), typeof(ScriptB), typeof(ScriptC))] ❸
public class ScriptD : MonoBehaviour {
    void Update() {
        /* ... */
    }
}
```

The script execution order will be generated automatically from the `RunAfter`/`RunBefore` attributes.

In the above example, the script order will be `ScriptC`, `ScriptA`, `ScriptB`, `ScriptD`.

`ScriptD` will be last because the line at ❸ tells it to run after the other three.

`ScriptB` will be run after `ScriptC` due to line ❶.

`ScriptC` will run before `ScriptA` due to line ❷.

> *Note*
>
> You can combine `RunBefore` and `RunAfter` attributes on the same class.

RunFirst and RunLast

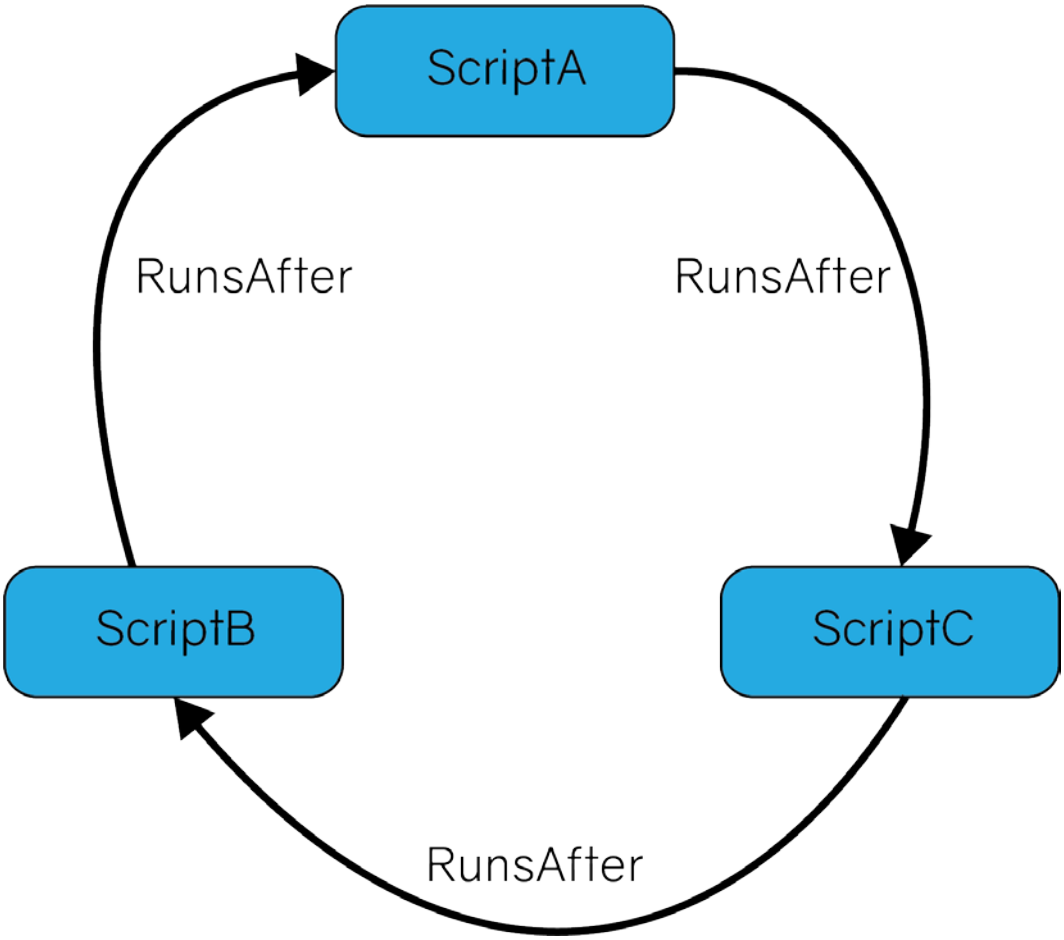
Additionally, you can use the `RunFirst` and `RunLast` attributes to make sure a component gets to run before or after everything else.

```
using com.kupio.declarativeorder;

[RunFirst]
public class ScriptA : MonoBehaviour {
    void Update() {
        /* ... */
    }
}

[RunLast]
public class ScriptB : MonoBehaviour {
    void Update() {
        /* ... */
    }
}
```

Here, `ScriptA` will be at the top of the execution order list, and `ScriptB` will be at the bottom.

Circular dependencies

Watch out for circular dependencies. This is where you create a chain of dependency that cannot be resolved.

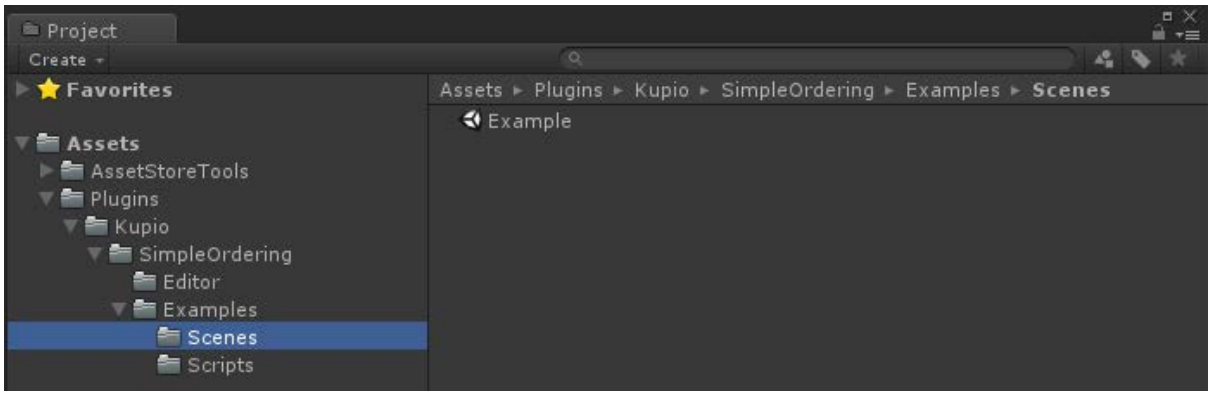


If this happens, then a warning will be printed to the console after Unity rebuilds your code. Your code will still run, but the script execution order may not be what you expected.

> *Note*
>
> Take care with subclasses. The annotation must be placed on the actual component class that is added to your `GameObject`s. Superclasses with the annotation will be ignored because Unity checks only the name of the class in the script order, not its hierarchy.

Examples

A simple example is provided and can be found by loading the scenes in this location:



Instructions for exploring the example and its code are presented on-screen in the scene itself.

Feedback

All feedback is welcome. Please allow 24 hours for email response. We try to respond very quickly, but bear in mind that we may not be in your time zone.

If you have problems with the asset, then please attempt to contact us first before leaving any feedback on the Unity asset store. We are happy to resolve user issues, and your issues may be very quick to resolve.

To contact us about anything at all, simply email us at:

support@kupio.com

Thank you ☺

Changelog

1.0

Initial release

1.1

Added RunFirst/RunLast

Documentation improvements

1.2

Project renamed

1.3

Revised documentation