# Unity Data oriented Technical Stack

## Introduction to ECS

### Why is ECS useful?

- Extremely performant code (default)
- Easier to read
- Easier to reuse code
- Take advantage of Burst Compiler for High Performance C#
- C# Job System

Allow us to write extremely performing code. It's a much better way to structure our data.

### What is ECS?

ECS is basically a new way of writing code in Unity. With ECS we are moving from traditional object-oriented programming to something called data oriented design.

So far when using Unity pretty much everything has been based around GameObjects and MonoBehaviours. Say, for example, we wanted to create a player, we would make a game object and name it **Player**. And  on this object we would place MonoBehaviour components to give the player functionality. These components would take care of stuff like rendering, physics and movement.

With **ECS** we are taking a different approach by splitting our game into three parts: Entities, Components and Systems.

**Entities** are used to group together components. They're much like traditional GameObjects but way lighter.

** Components** are just containers for data. Unlike traditional amount of behaviors, these components don't have any logic in them. Instead, we use **systems**.

**Systems** define component based behavior. What this means is that a system is responsible for operating on all entities with a specific set of components. These are the only objects that actually contain any logic.

So now if we wanted to create a player, we would make an entity. And on this entity we would place components. However, the only job of these components is to store data about a player. So currently it woldn't do anything. To change this, we create systems. We could, for example, create a render system to render all entities with the render component and a physics system to apply

physics to all entities with the physics component. Of course, the ECS package already has several of these basic systems ready to use. So that is the general explanation, let's see how we can apply this in practice.

First we need to install ECS. ECS is currently available as a preview package. To get it, we first need to install Unity 2018.1 or later. We can then create a new project. Once it's up and running, we can go **Edit > Project Settings > Player > Other Settings**. Under **configuration**, we change the scripting runtime version to **.Net 4x Equivalent**. This will require a quick restart. After that, we can open our project folder. Navigate to Packages folder and open the manifest.json file. Here we neeed to add a few lines.  We can then save the file and Unity should start loading in the ECS package. When it's done, we can go to **Window** (menu) and open up the package manager. And we should now see the entities package installed here.

Now, say, we want to rotate an object over time. First let's create a cube and let's create a new scipt on this cube called rotator. If we open up the script we can see that as expected this automatically creates a rotated class that derives from MonoBehaviour. With the old system, we could then use an update method to change the rotation of the transform component and maybe create a variable to control the speed. By running this code, all objects with this script attached will rotate according to our speed variable . And if we take a second to analyze the code, we can actually see that the MonoBehaviour contains both the data which in this case is our speed variable and the behaviour where we update the rotation. Now this might look fine on the surface, but it's actually not a very performant way of doing things and it also makes a bit hard to distinguish what is going on.

To avoid this, with ECS, we separate the data from the behavior. And there are currently two ways to do this. The first is using pure ECS. This system is completely separate from the old way of coding. With pure ECS, we stopped using GameObjects and MonoBehaviours all together. The second way is using hybrid ECS. To make it easier to transition from the old system to ECS, Unity has created a way to combine the two. Now this won't unlock the full performance benefits of pure ECS. But it's a great way to start converting your scripts to ECS and learning this new way of thinking. The cool thing is that by using hybid ECS, we can continue using MonoBehaviours for storing data. So currently we have some scripts sitting on different objects. Each script has some data and some behavior. Now with hybrid ECS, we still keep a script on each object that has the data we need, but we use a system to handle the behavior. In other words, we can use a class called the ComponentSystem to update the rotation of all our objects. Now if this all sounds a bit overwhelming, that's fine. But let's try to take it from the top like before we first create a class deriving from MonoBehaviour. But this time we only put our variable here. Then for our behaviour, we create a new class and derive it from ComponentSystem

参考:


1. Unity Dots
2. DOTS Documents
3. . DOTS Documents readme
4. [ECS Video Tutorial](https://learn.unity.com/tutorial/entity-component-system

5. UniteAustinTechnicalPresentation
6. ECS Manual
7. ECS API
8. MegaCity
9. Entity Component System Wiki
10. Unity at GDC - C# to Machine Code
11. Unity-Technologies/EntityComponentSystemSamples
12. What is an Entity Component System architecture for game development?
13. Parsec Patrol Diaries: Entity Component Systems
14. Unity DOD(ECS) 学习总结
15. Data Oriented Technology Stack
16. On DOTS: Entity Component System
17. Overwatch Gameplay Architecture and Netcode
18. 浅谈Unity ECS（一）Uniy ECS基础概念介绍：面向未来的ECS
19. Unity DOTS(一） Job System 介绍
20. 详解实体组件系统ECS
21. Unity ECS框架Entities源码解析
22. gametorrahod.com 有关于ECS一些技术细节的文章