**General purpose components**

ComponentData in Unity is a struct that contains only the instance data for an [entity](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/entities.md). ComponentData should not contain methods beyond utility functions to access the data in the struct. You should implement all of your game logic and behavior in systems. To put this in terms of the object-oriented Unity system, this is somewhat similar to a Component class, but one that **only contains variables**.

The Unity ECS API provides an interface called [IComponentData](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.IComponentData.html) that you can implement in your code to declare a general-purpose component type.

**IComponentData**

Traditional Unity components (including MonoBehaviour) are [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) classes that contain data and methods for behavior. IComponentData is a pure ECS-style component, which means that it defines no behavior, only data. You should implement IComponentData as struct rather than a class, which means that it is copied [by value instead of by reference](https://stackoverflow.com/questions/373419/whats-the-difference-between-passing-by-reference-vs-passing-by-value?answertab=votes#tab-top) by default. You usually need to use the following pattern to modify data:

var transform = group.transform[index]; // Read

transform.heading = playerInput.move; // Modify

transform.position += deltaTime \* playerInput.move \* settings.playerMoveSpeed;

group.transform[index] = transform; // Write

IComponentData structs must not contain references to managed objects. This is because ComponentData lives in simple non-garbage-collected tracked [Chunk memory](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/chunk_iteration.html), which has many performance advantages.

**Managed IComponentData**

It is helpful to use a managed IComponentData (that is, IComponentData declared using a class rather than struct) to help port existing code over to ECS in a piecemeal fashion, interoperate with managed data not suitable in ISharedComponentData, or to prototype a data layout.

These components are used the same way as value type IComponentData. However, ECS handles them internally in a much different (and slower) way. If you don't need managed component support, define UNITY\_DISABLE\_MANAGED\_COMPONENTS in your application's **Player Settings** (menu: **Edit > Project Settings > Player > Scripting Define Symbols**) to prevent accidental usage.

Because managed IComponentData is a managed type, it has the following performance drawbacks compared to valuetype IComponentData:

* It cannot be used with the Burst Compiler
* It cannot be used in job structs
* It cannot use [Chunk memory](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/chunk_iteration.html)
* It requires garbage collection

You should try to limit the number of managed components, and use blittable types as much as possible.

Managed IComponentData must implement the IEquatable<T> interface and override for Object.GetHashCode(). Additionally, for serialization purposes, managed components must be default constructible.

You must set the value of the component on the main thread. To do this, use either the EntityManager or EntityCommandBuffer. Because a component is a reference type, you can change the value of the component without moving entities across Chunks, unlike [ISharedComponentData](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.ISharedComponentData.html). This does not create a sync-point.

However, while managed components are logically stored separate from value-type components, they still contribute to an entity's EntityArchetype definition. As such, adding a new managed component to an entity still causes ECS to create a new archetype (if a matching archetype doesn't exist already) and it moves the entity to a new Chunk.

For an example, see the file: /Packages/com.unity.entities/Unity.Entities/IComponentData.cs.