**System State Components**

You can use [SystemStateComponentData](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.ISystemStateComponentData.html) to track resources internal to a system and create and destroy those resources as needed without relying on individual callbacks.

SystemStateComponentData and [SystemStateSharedComponentData](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.ISystemStateSharedComponentData.html) are similar to ComponentData and SharedComponentData, but ECS does not delete SystemStateComponentData when an entity is destroyed.

When an entity is destroyed, ECS usually:

1. Finds all components which reference the particular entity's ID.
2. Deletes those components.
3. Recycles the entity ID for reuse.

However, if SystemStateComponentData is present, ECS does not recycle the ID. This gives the system the opportunity to clean up any resources or states associated with the entity ID. ECS only reuses the entity ID once SystemStateComponentData is removed.

**When to use system state components**

Systems might need to keep an internal state based on ComponentData. For instance, resources might be allocated.

Systems also need to be able to manage the state as values, and other systems might make state changes. For example, when values in components change, or when relevant components are added or deleted.

"No callbacks" is an important element of the ECS design rules.

The general use of SystemStateComponentData is expected to mirror a user component, providing the internal state.

For instance, given:

* FooComponent (ComponentData, user assigned)
* FooStateComponent (SystemComponentData, system assigned)

**Detecting when a component is added**

When you create a component, a system state component does not exist. The system updates queries for components without a system state component, and can infer that they have been added. At that point, the system adds a system state component and any needed internal state.

**Detecting when a component is removed**

When you remove a component, the system state component still exists. The system updates the queries for the system state component without a component, and can infer that they have been removed. At that point, the system removes the system state component and fixes any needed internal state.

**Detecting when an entity is destroyed**

DestroyEntity is a shorthand utility for:

* Find components which reference given entity ID.
* Delete components found.
* Recycle entity ID.

However, SystemStateComponentData are not removed on DestroyEntity and the entity ID is not recycled until the last component is deleted. This gives the system the opportunity to clean up the internal state in the exact same way as with component removal.

**SystemStateComponent**

A SystemStateComponentData is similar to a ComponentData.

struct FooStateComponent : ISystemStateComponentData

{

}

Visibility of a SystemStateComponentData is also controlled in the same way as a component (using private, public, internal) However, it's expected, as a general rule, that a SystemStateComponentData will be ReadOnly outside the system that creates it.

**SystemStateSharedComponent**

A SystemStateSharedComponentData is similar to a SharedComponentData.

struct FooStateSharedComponent : ISystemStateSharedComponentData

{

public int Value;

}

**Example system using state components**

The following example shows a simplified system that illustrates how to manage entities with system state components. The example defines a general-purpose IComponentData instance and a system state, ISystemStateComponentData instance. It also defines three queries based on those entities:

* m\_newEntities selects entities that have the general-purpose, but not the system state component. This query finds new entities that the system has not seen before. The system runs a job using the new entities query that adds the system state component.
* m\_activeEntities selects entities that have both the general-purpose and the system state component. In a real application, other systems might be the ones that process or destroy the entities.
* m\_destroyedEntities selects entities that have the system state, but not the general-purpose component. Since the system state component is never added to an entity by itself, the entities that this query selects must have been deleted, either by this system or another system. The system reuses the destroyed entities query to run a job and remove the system state component from the entities, which allows the ECS code to recycle the entity identifier.

**Note:** This simplified example does not maintain any state within the system. One purpose for system state components is to track when persistent resources need to be allocated or cleaned up.

例子见文档