Entity Prototypes

Introduction

To facilitate data driven design, Quantum 2.0 introduced *Entity Prototypes*.

An *Entity Prototype* is a serialized version of an entity that includes:

- composition (i.e. which components it is made of); and,
- data (i.e. the components' properties and their initial value).

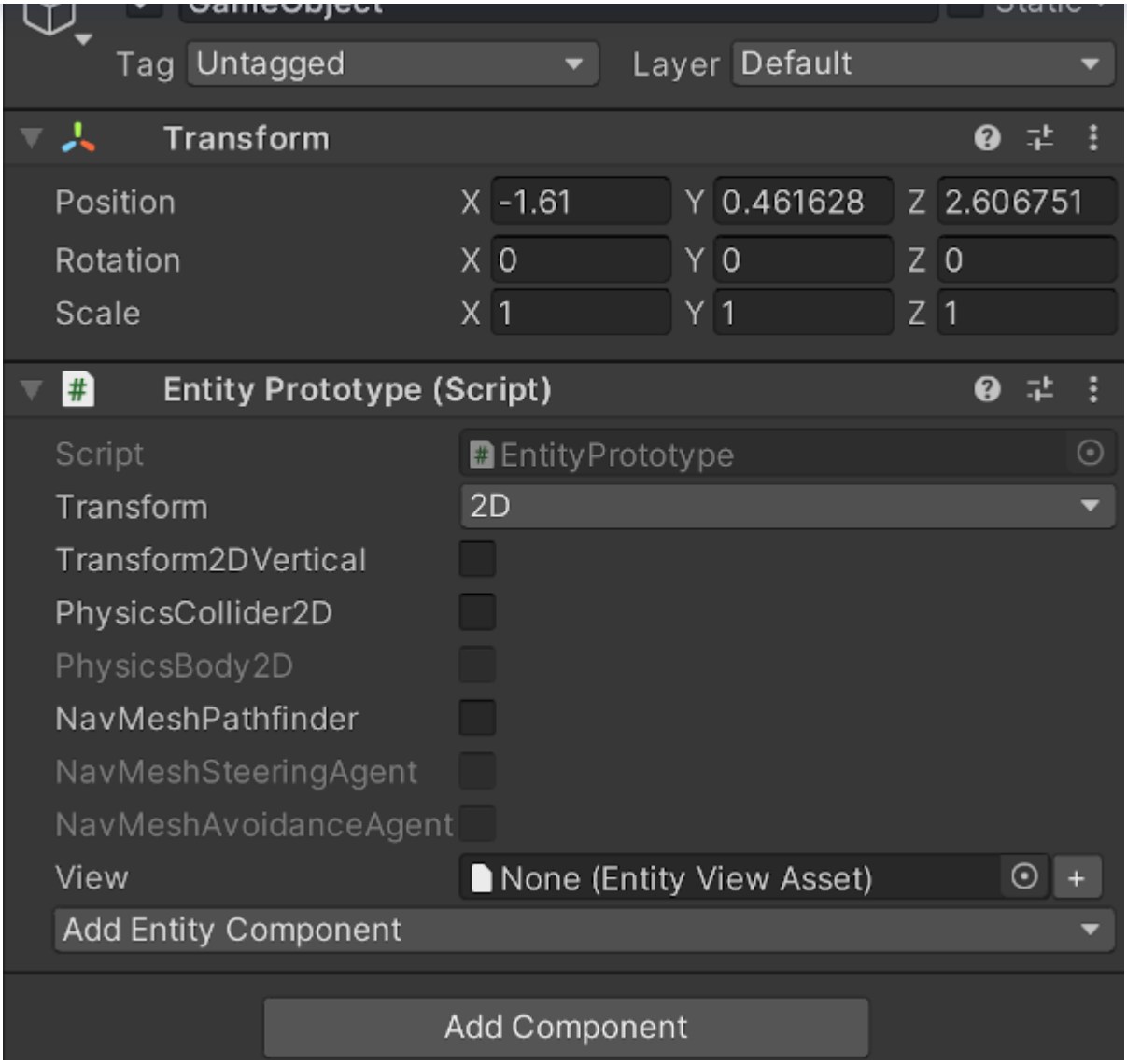
This allows for a clean separation of data and behaviour, while enabling designers to tweak the former without programmers having to constantly edit the latter.

Setting up a Prototype

Entity prototypes can be set up in Unity Editor.

Basic

To create an *Entity Prototype* simply add the *Entity Prototype* script to any GameObject.



Basic Entity Prototype (Empty GameObject + Entity Prototype Script).

The *Entity Prototype* script allows you to set up and define the parameters for the most commonly used components for both 2D and 3D.

- Transform (including Transform2DVertical for 2D)
- PhysicsCollider
- PhysicsBody
- NavMeshPathFinder
- NavMeshSteeringAgent
- NavMeshAvoidanceAgent

The dependencies for the Physics and NavMesh related agents are respected. For more information, please read their respective documentation.

Custom Components

- the *Add Entity Component* drop-down; or,
- the regular Unity *Add Component* button by searching for the right *Entity Component*.

Note on Collections

Dynamic collections in components are only automatically allocated **IF** there is at least one item in them. Otherwise, the collection will have to be allocated manually. For more information on the subject, refer to the Dynamics Collection entry on the DSL page.

Hierarchy

In ECS the concept of entity/GameObject hierarchy does not exist. As such entity prototypes do not support hierarchies or nesting.

Although child prototypes are not supported directly, you can:

1. Create separate prototypes in the scene and bake them.
2. Link them by keeping a reference in a component.
3. Update the position of the "child" manually.

Note: Prototypes that are not baked in scene will have to follow a different workflow where the entities are created and linked in code.

You can have hierarchies in objects (View), however hierarchies in entities (Simulation) will have to be handled by you.

Creating/Instantiating a Prototype

Once a *Entity Prototype* has been defined in Unity, there are various ways to include it in the simulation.

Baked in the Scene/Map

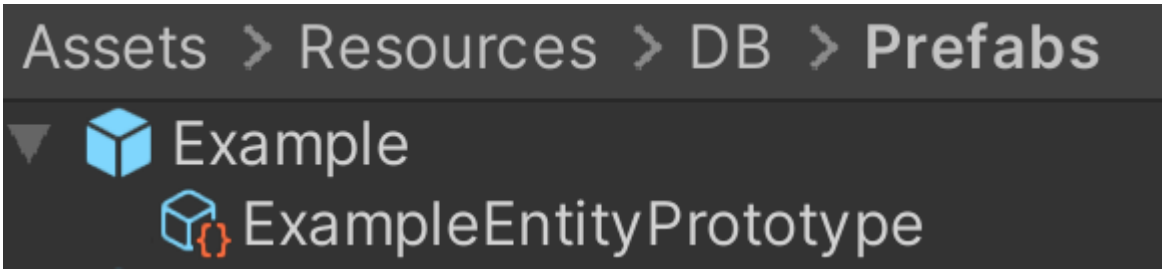
If the *Entity Prototype* is created as part of a Unity Scene, it will be baked into the corresponding Map Asset. The baked *Entity Prototype* will be loaded when the Map is and initialized with the values it

N.B.: If a Scene's *Entity Prototype* is edited or has its values changed, the Map Data has to be re-baked.

In Code

To create a new entity from an *Entity Prototype*, you need to follow these steps:

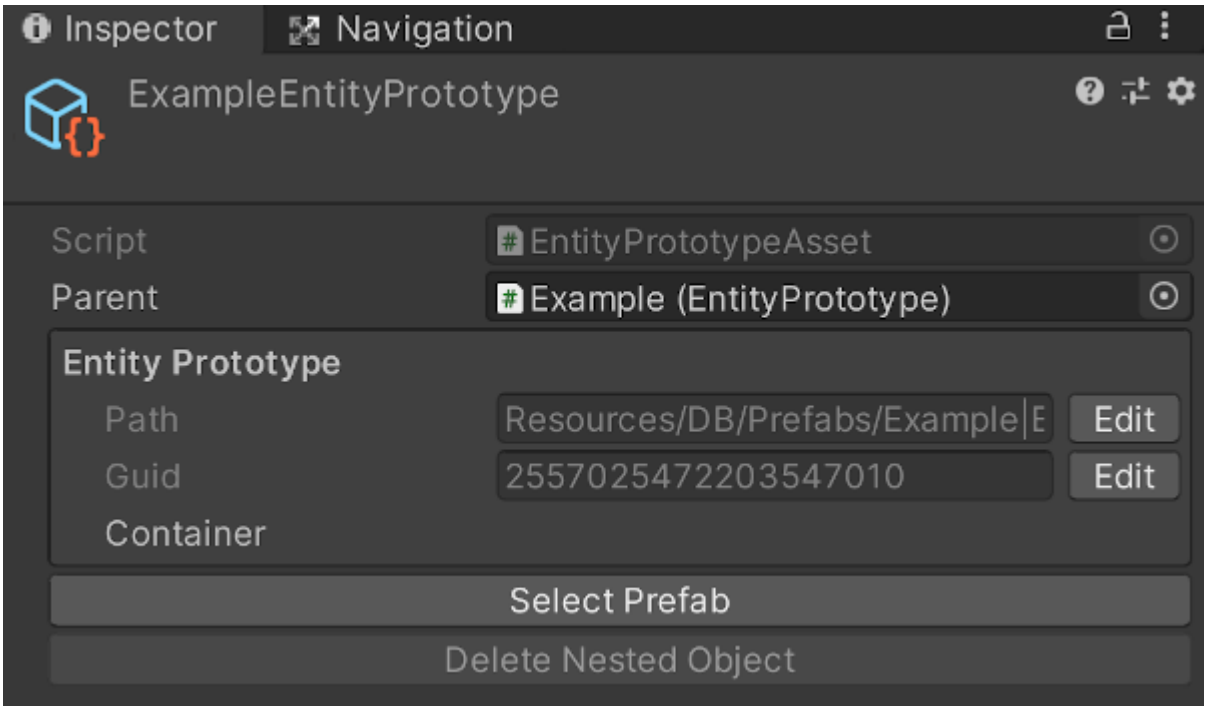
1. Create a Unity Prefab of the GameObject carrying the *EntityPrototype* script.
2. Place the Prefab in `Resources\DB`.



Entity Prototype Prefab + Nested Entity Prototype Asset.

=> This will generate a nested *Entity Prototy* **Asset**.

3. Refresh the Quantum Database `Quantum -> Generate Asset Resources`.
4. Make the *Entity Prototy* **Asset** Path or GUID available to your simulation.



Entity Prototype Asset Window.

5. Call `Create()` via the frame and pass, for example, the EntityPrototype reference, or an instance of it:

C#

```
void CreateExampleEntity(Frame f){
    // using a reference
    var exampleEntity = f.Create(myPrototypeReference);

    // OR, getting an instance before, using the asset's path as
    var entityPrototype = f.FindAsset<EntityPrototype>("Resources
    var exampleEntity = f.Create(entityPrototype);
}
```

Note

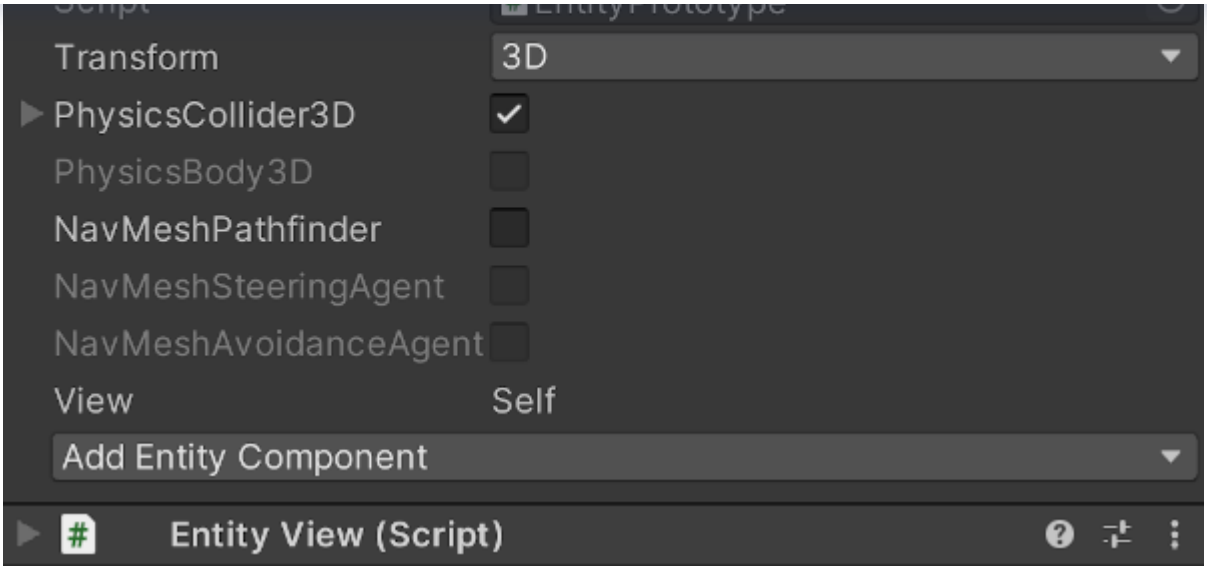
Entity Prototypes present in the Scene are baked into the **Map Asset**, while prefabed *Entity Prototypes* are individual **Assets** that are part of the Quantum Asset DataBase.

Entity View

The *Entity View* corresponds to the visual representation of an entity in Unity. In the spirit of data driven design, an *Entity Prototype* can either incorporate its *View* component or point to a separate *EntityView* Asset.

Self

To set an *Entity Prototype*'s view to itself, simply add the *Entity View* component to it.



Entity Prototype with "Self" View.

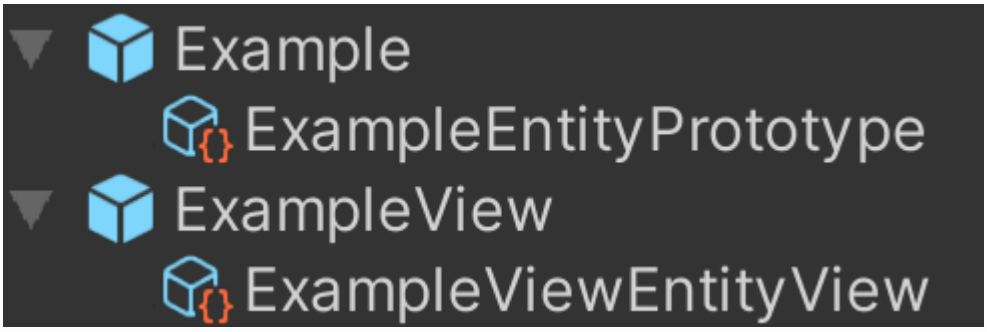
Once the component has been added, the *Entity Prototype* script will list **Self** as the value for the *View* parameter. This will also create a nested *Entity View* **Asset** in the same prefab.

Entity Prototype Asset and "Self" View Asset.

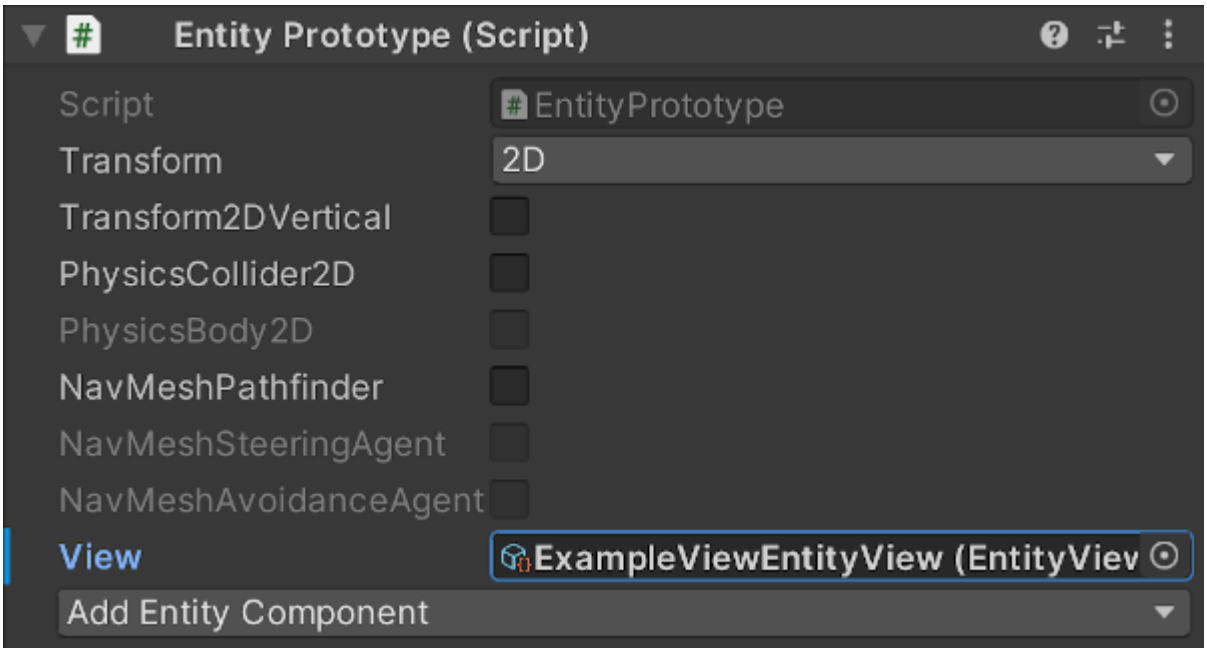
Separate from Prototype

To set up and link a view separate from the *Entity Prototype* asset:

1. Add the *Entity View* to the GameObject you would like to represent the view.
2. Prefab the GameObject carrying the *Entity View*.
3. Place the prefab in `Resources\DB`, this will create an *Entity View* **Asset** nested in the prefab.



4. Refresh the database `Quantum -> Generate Asset Resources`.
5. Link the *View* field from the *Entity Prototype* with the newly created *Entity View Asset*. This can be done via drag-and-drop or the Unity context search menu.



Linking an Entity Prototype with a separate Entity View Asset.

Important

For an *Entity View* to be visible in Unity, the scene has to have an `EntityViewUpdater` script.

Back to top

photon

Memberships

Gaming Circle
Industries Circle

Support

Gaming Circle
Industries Circle
Circle Discord
Circle Stack Overflow

Connect

Public Discord
YouTube
Facebook
Twitter
Blog
Contact Us

Bolt
Server
VR | AR | MR

Resources

Dashboard
Samples
SDK Downloads
Cloud Status

Languages

English
日本語
한국어
简体中文
繁体中文