




Prerequisites

NOTE: THIS WILL NOT WORK IN A NON-DOTS/ECS ENVIROMENT. YOU MUST BE WORKING IN UNITY DOTS & ECS.

Before you setup **DOTS Dynamic Bone** it is **important** that you import the following unity packages from the Package Manager (Note: links may not always send you to the latest version):

Packages
Unity.Entities v1.0.16+ 
Unity.Physics v1.0.16+ 
Unity.EntitiesGraphics v1.0.16+ 

Either the Universal Render Pipeline and/or High Definition Render Pipeline with Entities Graphics.

Project Setup

NOTE: Given that Unity still doesn't have an ECS Animation system in place the the newest version of DOT Dynamic Bone with Entities Graphics.

First lets add some Scripting Define Symbols in your Project Settings:

- *ENABLE_COMPUTE_DEFORMATIONS* - For Entities Graphics
- *ENABLE_DOTS_DEFORMATION_MOTION_VECTORS* - For using Entities Graphics Deformation **NOTE:** this may require extra setup, please read the Unity Doycmnetaiton regarding this.
- *UNITY_PIPELINE_URP* - if using Unity URP
- *DOTSDYNAMICBONE_UNITY_PHYSICS* - if using Unity.Physics

Then import DOTS Dynamic Bone from the Package Manager and your Done!

General Concepts

This sections contains information on the concepts of DOTSDynamic Bone

Topic	Description
DOTSDynamicBoneComponent	Goes over the DOTSDynamicBone and some of its features and quirks.
DOTSDynamicBone	Describes what a DOTSDynamicBone is and some of its features and quirks.
Particle	Describes what a Particle is and some of its features and quirks.
RigComponent	Gives some info regarding the RigComponent .
Systems	Goes overs the Systems in DOTSDynamicBone

Concepts/Keywords

- **Particle Simulation:** Refers to the default simulation behaviour without Physics Collisions.
- **Physics Simulation:** Refers to the simulation behaviour with Physics interactions/collisions also.
- **Core Entity/Rig Component Entity/Rig Root Entity:** Referes to the Entity that has a [DOTSDynamicBoneComponent](#) attached.

DOTS Dynamic Bone Component

The [DOTSDynamicBone](#) is a `Monobehaviour` is used to create a [DOTSDynamicBone](#). The data it contains is used in the editor and during runtime. In terms of the Editor there are 2 Lists called [eParticles](#) (short for editor particles) and `rParticles` (short for runtime particles). [eParticles](#) are responsible for containing information used in both the Editor and at runtime during the baking process. `rParticles` are mainly used for debugging in the editor and will be visible if `DOTSDYNAMICBONE_DEBUG` is defined in the Compiling Systems.

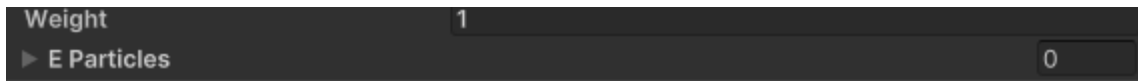
NOTE: The information below will go over some fields and thier quirks, if a field is not listed below then please the Api Documentation for more info.

Configurations

Default

To get DOTS Dynamic Bone working all newly created [DOTSDynamicBoneClasss](#) are initialized with a Default configuration:

Name						
Disable	<input type="checkbox"/>					
Update Data From	Animated Local To World ▾					
Use Natural Colliders	<input type="checkbox"/>					
Update World Positions	<input checked="" type="checkbox"/>					
Update World Rotations	<input checked="" type="checkbox"/>					
Update Local Positions	<input type="checkbox"/>					
Update Local Rotations	<input type="checkbox"/>					
Update Transform But Flip Local And	<input type="checkbox"/>					
Update Transform As If No Parent	<input type="checkbox"/>					
Use Animated Local To Root	<input checked="" type="checkbox"/>					
Reset Options						
Reset All For One Frame	<input type="checkbox"/>					
Only Execute On First Particle	<input type="checkbox"/>					
Reset World Position	X	<input checked="" type="checkbox"/>	Y	<input checked="" type="checkbox"/>	Z	<input type="checkbox"/>
Reset World Rotation	X	<input checked="" type="checkbox"/>	Y	<input checked="" type="checkbox"/>	Z	<input type="checkbox"/>
Reset Local Position	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Reset Local Rotation	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Reset M Transform	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Reset Physics Velocity	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Reset ALTW	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Reset ALTR	X	<input type="checkbox"/>	Y	<input type="checkbox"/>	Z	<input type="checkbox"/>
Flip World And Local Transforms	<input type="checkbox"/>					
Rig Component	unitychan_DOTS_Mutiple_DDB (Rig Component) ⦿					
Root Bone	None (Transform) ⦿					
Update Rate	60					
Update Mode	Default ▾					
Damping	<input type="range"/>					0.1
Damping Distrib	<input type="range"/>					
Elasticity	<input type="range"/>					0.1
Elasticity Distrib	<input type="range"/>					
Stiffness	<input type="range"/>					0.1
Stiffness Distrib	<input type="range"/>					
Inert	<input type="range"/>					0
Inert Distrib	<input type="range"/>					
Friction	0					
Friction Distrib	<input type="range"/>					
Radius	0					
Radius Distrib	<input type="range"/>					
End Length	0					
End Offset	X	0	Y	0	Z	0
Global Force	X	0	Y	0	Z	0
Gravity	X	0	Y	0	Z	0
Force	X	0	Y	0	Z	0
Colliders						0
Exclusions						0
Freeze Axis	None ▾					
Culling Data						
Bone Total Length	0					



This should be enough to get you started however, if your not using physics it is highly recommended to not set `UpdateLocalPositions`, `UpdateLocationRotations`, `UseNaturalColliders`, `UpdateTransformsAsIfNoParent`, & `UpdateTransformButFlipLocalAndWorld` to true. We will go over those settings later in the doc.

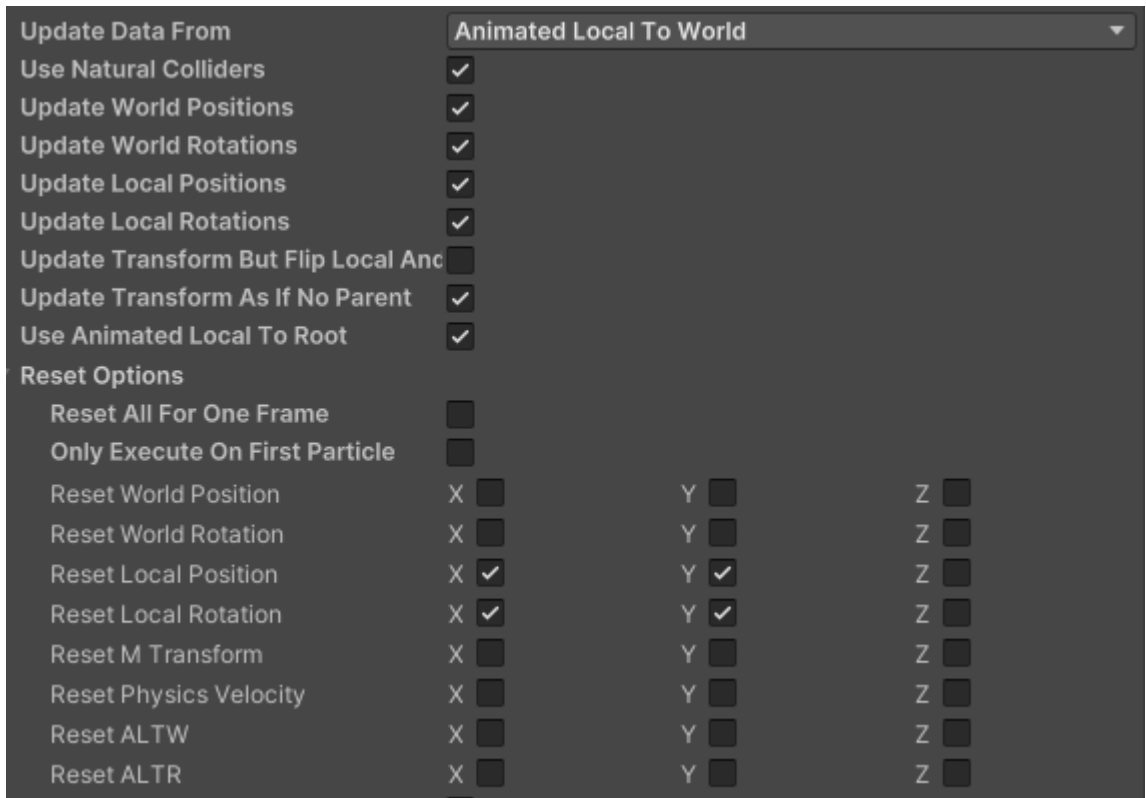
NOTE: For more information or if the image isn't loading please check the `InitializeDefault` function in [DOTSDynamicBoneClass](#)

Using Physics

if you plan on using physics then there are a few things you must know. First is how to setup the Transform update boolean parameters like below:

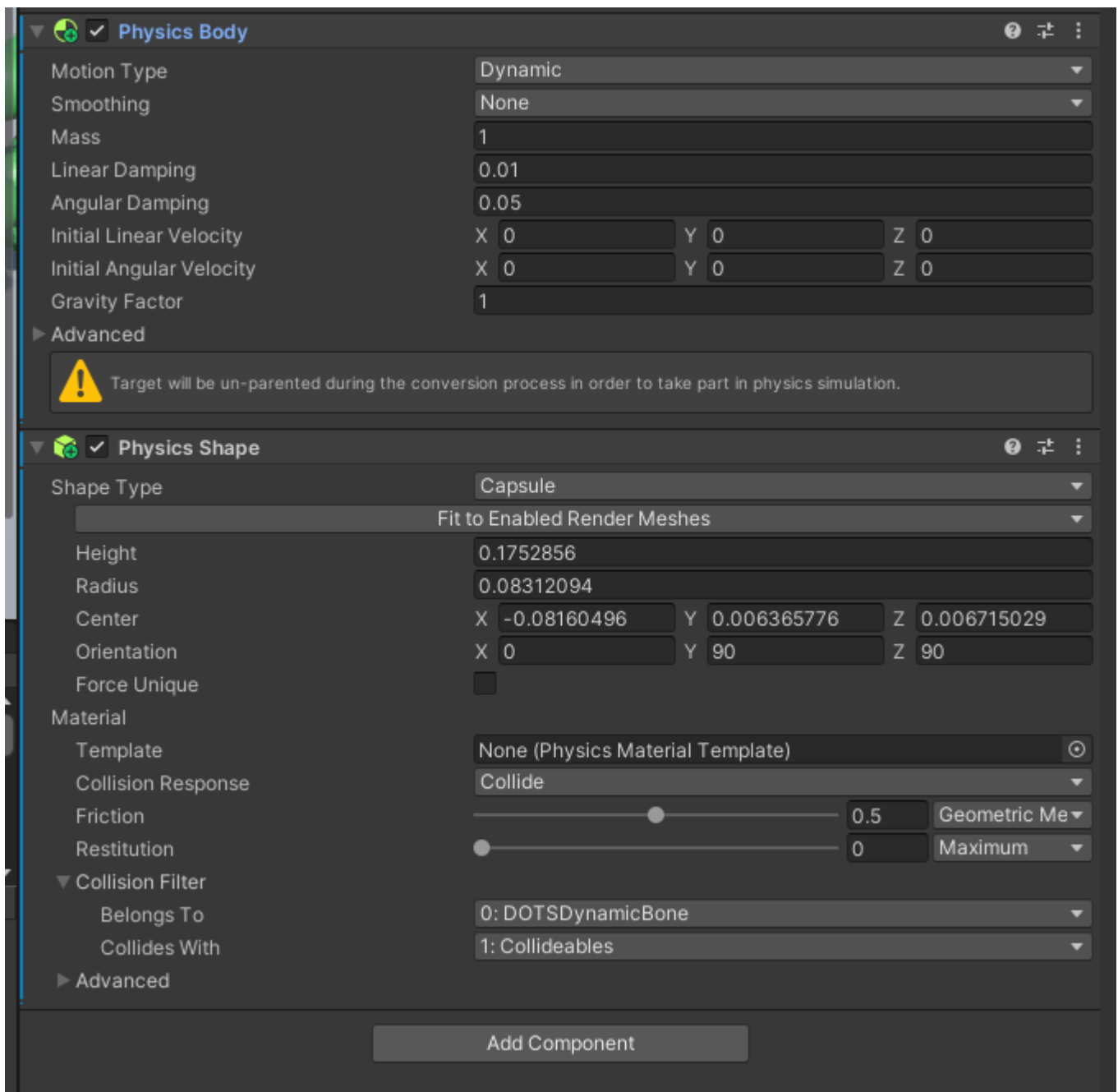
NOTE: Be sure to add the `DOTSDYNAMICBONE_UNITY_PHYSICS` compiler definition in your project.

NOTE: When using the `PhysicsDebugDisplay`, since `DOTSDynamicBones` executes *after* that system the drawn shapes will look like they are a frame behind.



NOTE: For more information or if the image isn't loading please check the `InitializeDefault` function in [DOTSDynamicBoneClass](#)

Then you have to make sure your Entity has a `PhysicsShape|UnityEngine.Collider` and a `PhysicsBody|UnityEngine.Rigidbody`:



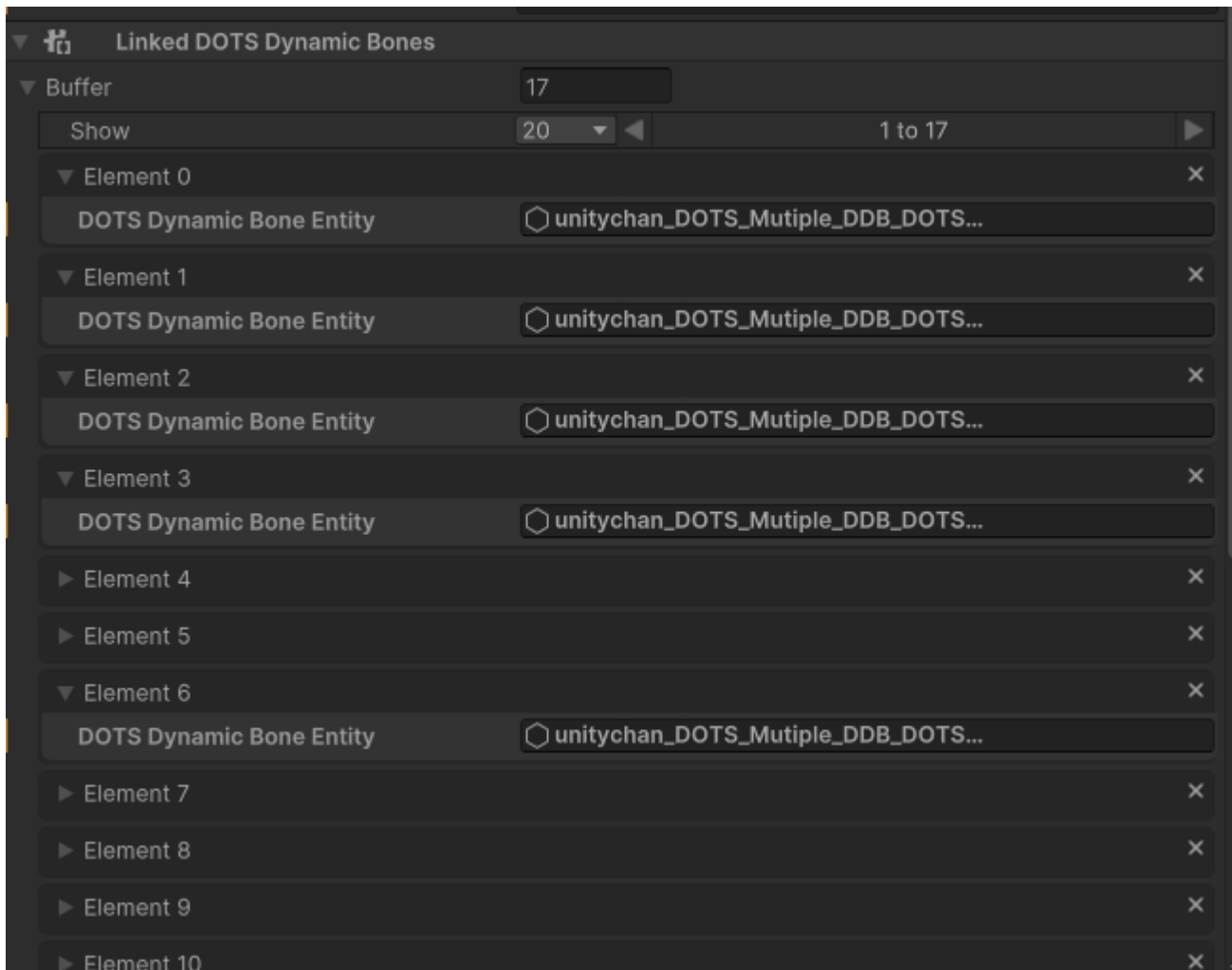
NOTE: besure your collision filter is setup so the colliders don't interact with each (or you can setup better colliders then me :))

If there are any Entities that do not have Physics Components but is a child of the **RootBone** in [DOTSDynamicBoneClass](#) then you must add that Entity in the Exclusions List with **ExcludeFromCollision** enabled (otherwise things may break a bit). Please go to the Exclusions section for more information

Fields & Parameters

DOTS Dynamic Bones (List)

This list will contain all the [DOTSDynamicBone](#) data for the associated Entity. For each element in this list a new Entity with all the [DOTSDynamicBone](#) data will be recreated and linked to the associated Entity.



Name

The name field is used in both the editor and runtime. In the Editor it is used to differentiate between [DOTSDynamicBoneClass](#) data within a [DOTSDynamicBoneComponent](#). During runtime the name of the entity is constructed in the folloing way:

```
authoring.name + "_DOTSDynamicBone_" + m_DOTSDynamicBone.Name
```

This was done to help ensure the uniqueness of names without using a uuid (though technically a entity with a duplicate name can occur if you name everything the same)

Update Data From

This field will almost always be set to `AnimatedLocalToWorld` but feel free to play around with it or add your own!

Transform Update Settings

This include the fields and descriptions:

- Use Natural Colliders - This is used when you want physics calculations to be enabled on each [Particle](#)
- Update World Positions - This should always be true. It tells the [DOTSDynamicBoneVisualPhysicalOverrideSystem](#) to update `LocalToWorld` position data.
- Update World Rotations - This should always be true. It tells the [DOTSDynamicBoneVisualPhysicalOverrideSystem](#) to update `LocalToWorld` rotation data.
- Update Local Positions - This should be true if `UseNaturalColliders` is true. It tells the [DOTSDynamicBoneVisualPhysicalOverrideSystem](#) to update `LocalTransform` position data.
- Update Local Rotations - This should be true if `UseNaturalColliders` is true. It tells the [DOTSDynamicBoneVisualPhysicalOverrideSystem](#) to update `LocalTransform` rotation data.
- Update Transform But Flip Local And World - In practice this should never be enabled. feel free to enable it and see what happens :)
- Update Transform As If No Parent - This should be true if `UseNaturalColliders` is true. This notifies systems that some [Particles](#) may be unparented and must adjust how they process thier data.
- Used AnimatedLocalLocalToRoot - this is always true (see code for more info)

Reset Options

each bool3 is used in the [DOTSDynamicBoneParticleResetSystem](#) and each boolean value is setup as:

- x: true = reset.
- y: true = reset on every frame if true else only in one frame.
- z: true = execute only on root particle.

In pratice you if physics isn't enabled then you have:

```
ResetWorldPosition = new bool3(true,true,false);
```

```
ResetWorldRotation = new bool3(true,true,false);
```

otherwise you have:

```
ResetLocalPosition = new bool3(true,true,false);
```

```
ResetLocalRotation = new bool3(true,true,false);
```

Rig Component

The RigComponent field is associated with SkinnedMeshRenderers and such.

Root Bone

The Root Bone refers to Entity that will be used to create Particles from. The Entity will be the `RootBone` and the children of that Entity will be the [Particles](#). Depending on your [DOTSDynamicBoneClass](#) settings you may create Particles with *RootBoneLike* values.

RootBoneLike = a Particle with a `m_ParentParticleIndex < 0 && m_LastParentParticleIndex < 0`

Colliders

If this list is not empty then the Particles generated from the DOTSDynamicBoneClass will only collide with Entities that contains the [DOTSDynamicBoneCollider](#) within this list.

Exclusions

This list holds information regarding which Entities will be excluded and how they will be excluded from calculations.

Exclude From Particle Creation

If this is set to true then the Entity will not have a particle created for it. You should set this to true if you don't plan on having the Entity being a part of the Particle Simulation later.

Exclude Children From Particle Creation

If this is set to true then the Entity's children will be excluded from Particle creation. You should set this to true if you don't plan on having the Entity being a part of the Particle Simulation later.

Disable Particle Physics

If this is set to true then After Initialization the Particle will be excluded from Particle Simulation however, it can be reenabled by setting `Particle.m_ExcludeFromParticlePhysics` to true.

Disable Children From Particle Physics

If this is set to true then After Initialization the Particle's children will be excluded from Particle Simulation however, it can be reenabled by setting `Particle.m_ExcludeFromParticlePhysics` to true.

Exclude From Collision

If this is set to true and UseNaturalColliders is set to true then Physics Collisions calculations will be skipped for the associated Entity.

Exclude Children From Collision

If this is set to true and UseNaturalColliders is set to true then Physics Collisions calculations will be skipped for the associated Entity's children.

For more information check out the API reference for [DOTSDynamicBoneComponent](#)

What is a DOTSDynamicBone?

A [DOTSDynamicBone](#) is a data structure that holds all data relevant to their associated [Particle](#)s. The Bone data structure is associated with 1 **Entity** which is Created during baking. The new Entity will contain the [DOTSDynamicBone](#) and other associated datastructures. Along with A DOTSDynamicBone a DOTSDynamicBone Entity is expected to be accompanied with:

- *DynamicBuffer*<[AnimatedLocalToWorld](#)>
- *DynamicBuffer*<[Particle](#)>
- *DynamicBuffer*<[AnimatedLocalToRoot](#)>
- [DOTSDynamicBoneEntity](#)

And is linked to the core entity via a *DynamicBuffer*<[LinkedDOTSDynamicBones](#)>.

For more information take a look at the API reference for [DOTSDynamicBone](#)

What is a Particle?

[Particles](#) in **DOTS Dynamic Bone** is a data structure that represents a Bone within a model i.e: a Hip, Finger, Hair, or Tail bone. The [Particles](#) are updated and modified during the simulation based on the Particle and [DOTSDynamicBone](#) parameters. [Particles](#) also can represent different things like segments, or divisions within a model. During each update, data from the associated **DOTSDynamicBone** is used to update the Particles in an initialization-like step. Afterward, the Particles is put through different methods that update its position and rotation based on the input data.

Where are Particles Created?

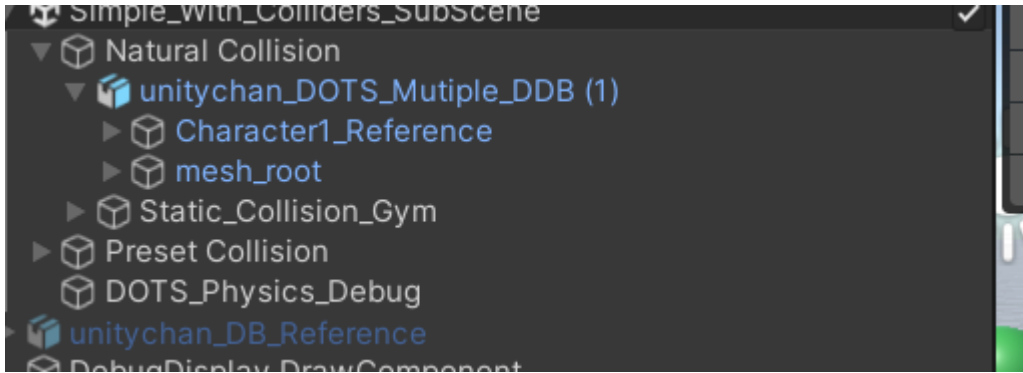
Particles are created during baking using the **DOTSDynamicBone.CreateParticles**.

For more info take a look at [Particle](#) page.

What is the RigComponent for?

The RigComponent is used to find required data in the SkinnedMeshRenderer of your GameObject.

NOTE: It's import that your `SkinnedMeshRenderers` are properly setup by making sure `Root Bone` is not null



In the above example `unitychan_DOTS_Multiple_DDB (1)` would be the `m_RigComponentEntity` since it contains the `RigComponent`

DOTS Dynamic Bone Contain 8 Systems 5 of which and which run within the DOTS Dynamic Bone Update Group and are described below:

System Name	Description
DOTSDynamicBone PhysicsVelocityOverride System	This System is responsible for handling some physics/transform velocity glitchyness. This system may not be needed on most projects so you should be to disable it with no issues.
DOTSDynamicBone PresetPhysicsCollision System	This system is responsible for running Particle simulations with Preset Collisions.
DOTSDynamicBone AnimatedLocalToXPre UpdateSystem	This system is responsible for updating the AnimatedLocalToRoot & AnimatedLocalToWorld data for each DOTSDynamicBone Entity.
DOTSDynamicBone CullingSystem	This system is responsible for updating the reference transform matrix in each DOTSDynamicBone
DOTSDynamicBone ParticleResetSystem	This system is responsible for resetting DOTSDynamicBone and Particle data based on the DOTSDynamicBone's Reset Options.
DOTSDynamicBone UpdateSystem	This system is responsible for running Particle simulations with and without Natural Physics Collisions.
DOTSDynamicBone VisualPhysicalOverride System	This system is responsible for updating the DOTSDynamicBone and Particle Entity's LocalToWorld and LocalTransform based on your Transform Update Settings
DOTSDynamicBone InitializationSystem	A DOTSDynamicBone Baking system.

What Demos are there

This update comes with 2 demos using Free Assets. The Demo are there to show some of the features of DOTS Dynamic Bone and some applications you can use it for. Below are the Links to the Demos

Demo	Demo Documentation
Simple Demo	Simple Demo
Simple Physics Demo	Simple Physics Demo
Simple Animation Demo	Simple Animation Demo
Simple Physics Animation Demo	Simple Physics Animation Demo

Simple Demo

This page is to describe the content within the Simple Demo Scene Located within Assets/Resources/unity-chan!/Unity-chan! Model/Scenes/DDB_Examples/Simple.unity

In this Demo we will go over some aspects

General Components

- Main Camera - Unity Camera.
- Directional Light - Unity Directional Light.
- Simple_SubScene - SubScene.
- unitychan_DOTS_Single_DDB - An example of DOTS Dynamic Bone using Unity Chan (Free Ver.) using a single DOTSDynamicBone.
- unitychan_DOTS_Multiple_DDB - An example of DOTS Dynamic Bone using Unity Chan (Free Ver.) using multiple DOTSDynamicBones.

unitychan_DOTS_Single_DDB

This Example's goal is to show how you would setup a complex model using a single DOTSDynamicBone. In this case it is not recommended to go about this due to the model's complexity however, it's your choice. Default parameters are used here however the **RootBone** is the *Hip*. So in order to apply the Particle Physics to the bones we want we essentially have to exclude every other bone (which is why this isn't recommended to very complex models). In the demo we have done this for you so feel free to take a look.

unitychan_DOTS_Multiple_DDB

In this example we show how to use the DOTS Dynamic Bone Component to create multiple DOTSDynamicBones. Since we are creating multiple DOTSDynamicBones we can setup them up differently. For example, *Character_RightForeArm* is setup to exclude the Forearm transform and the Right Hand.

Simple With Colliders Demo

This page is to describe the content within the Simple With Colliders Demo Scene Located within Assets/Resources/unity-chan!/Unity-chan! Model/Scenes/DDB_Examples/Simple_With_Colliders.unity

In this Demo we will go over how to setup DOTSDynamicBone with Physics Colliders.

NOTE: in order to use Unity.Physics make sure you have the scripting definition `DOTSDYNMAICBONE_UNITY_PHYSICS` added to your project.

General Components

- Main Camera - Unity Camera
- Directional Light - Unity Directional Light
- Simple_SubScene - SubScene
 - Natural Collision - GameObject that contains components related to using Natural Unity.Physics collision.
 - unitychan_DOTS_Mutiple_DDB - GameObject containing DOTSDynamicBone and Colliders.
 - Static_Collision_Gym - Group of GameObjects with Colliders and Collision Filters.
 - Preset Collision - GameObject that contains component related to using Preset Collisions.
 - unitychan_DOTS_Mutiple_DDB - GameObject containing DOTSDynamicBone and Colliders for Preset Collision Settings.
 - Static_Collision_Gym - Group of GameObjects with Colliders and Collision Filters used for Preset Collision.
 - DOTS_Physics_Debug - ECS Phsics Debugging.
 - Culling - GameObject that contains a DOTSDynamicBone culling setup.
 - Natural Collision - GameObject that contains components related to using Natural Unity.Physics collision.
 - unitychan_DOTS_Mutiple_DDB - GameObject containing DOTSDynamicBone and Colliders.
 - Static_Collision_Gym - Group of GameObjects with Colliders and Collision Filters.

Natural Collision

In the Natural Collision Example there are few extra things that had to be done aside to what has been done in the **Simple Demo.scene**. First we have to setup all the colliders on the Model. (You can view that by checking *Draw Colliders* in *Simple_With_Colliders_SubScene* > *DOTS_Physics_Debug* > *Physics Debug Display* > *Draw Colliders*). For our Demos we setup the Model's Colliders to only be apart of the DOTSDynamicBone Filter Group and only collider with the Collidables Filter Group. After the Model is setup we followed the instructions in [DOTSDynamicBoneComponent](#) under the *Using Physics* section.

Preset Collision

In the Preset Collision Example the same steps as the Natural Collision setup was followed initially. Then we setup **J_R_HairTail_00** to use Preset Collisions which was done by adding specific Transforms to that DOTSDynamicBone Colliders List. The transforms in that list in turn had DOTSDynamicBoneColliders attached to them. In the Demo you can see the in *Simple_With_Colliders_SubScene* > *Preset_Collision* > *Static_Collision_Gym* contains Green, Red, And Yellow Spheres. The **J_R_HairTail_00** is setup to Only Collide With the Yellow Shapes.

Culling

In the Culling Example we reuse the Natural Collision Setup with the only difference being adding Culling to the DOTSDynamicBones. This was done by setting the *ATA Culling Data* and checking the *Apply To All* checkbox. The 2 GameObjects named *Non-Culling_Area* and *Culling_Area* are just to show when Culling should and should not be enabled since it's setup to be distance based.

Simple With Animations Demo

This page is to describe the content within the Simple Demo Scene Located within Assets/Resources/unity-chan!/Unity-chan! Model/Scenes/DDB_Examples/Simple_With_Animations.unity

The setup for this is the same as Simple_Demo but with Animation provided using Latios Kinemation.

To do this we added Materials, Shaders, and a Bootstrap Scrips associated with the Latios Framework for ECS.

The Shaders are Located at *Assets > Materials* and the unitychan Materials are located in *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Materials_Latios_Kinemachine*

A Copy of the Original unitychan Model was Copied, Modified, and named Unitychan_DOTS_Latios.fbx located at *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Models > Unitychan_DOTS_Latios.fbx*

Some Mixamo animations for your use were imported and is located at *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Animations > Movements*

Simple With Collider And Animations Demo

This page is to describe the content within the Simple Demo Scene Located within
Assets/Resources/unity-chan!/Unity-chan!
Model/Scenes/DDB_Examples/Simple_With_Colliders_And_Animations.unity

The setup for this is the same as Simple_With_Physics_Demo but with Animation provided using Latios Kinemation.

To do this we added Materials, Shaders, and a Bootstrap Scrips associated with the Latios Framework for ECS.

The Shaders are Located at *Assets > Materials* and the unitychan Materials are located in *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Materials_Latios_Kinemachine*

A Copy of the Original unitychan Model was Copied, Modified, and named Unitychan_DOTS_Latios.fbx located at *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Models > Unitychan_DOTS_Latios.fbx*

Some Mixamo animations for your use were imported and is located at *Asset > Resources > unity-chan! > Unity-chan! Model > Art > Animations > Movements*