Cubism SDK for Cocos Creator alpha Tutorial

Import SDK

This is a tutorial on how to import a model file exported from Cubism Editor into a Cocos Creator project and display it on the screen.

What to prepare

Cocos Creator と Cubism SDK for Cocos Creator

Cubism SDK for Cocos Creator supports Cocos Creator version 3.6.2 or later. If your version is less than v3.6.2, the information serialized in the assets in the SDK may be discarded. For information on installing Cocos Creator, please see here.

Also, download Cubism SDK for Cocos Creator beforehand. This SDK is distributed as a zip file. This SDK is also available on Github, but it does not include Cubism Core libraries.

Node.js

When using Cubism SDK for Cocos Creator, you need to install the module using the npm command. Be sure to install the version of Node.js that is compatible with Cubism SDK for Cocos Creator.

Node.js

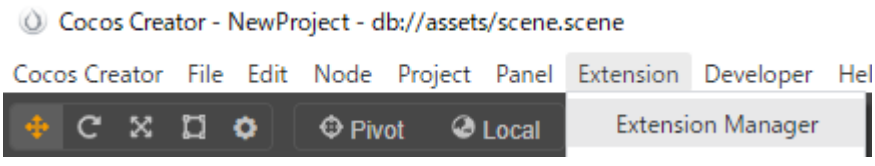
It can be installed from a Node.js management tool such as nvm. Please set an environment variable that allows npm to run at the same time as installation.

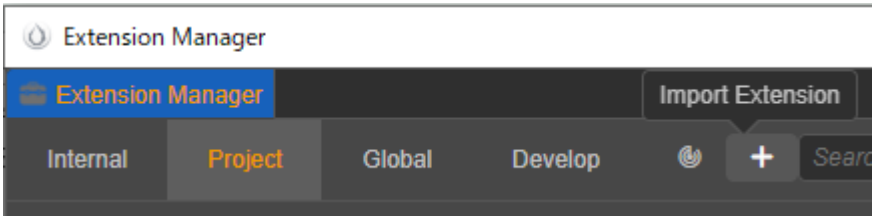
Cubism Model Data

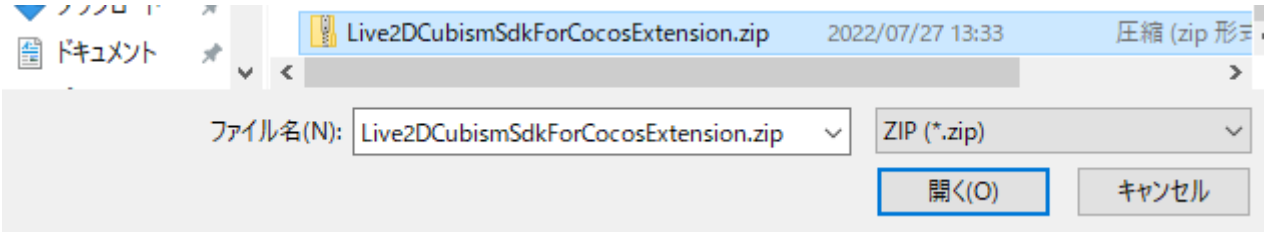
When working with Live2D models in SDK, you need to export them as built-in models, not as .cmo3 or .can3 models for editing. See here for information on how to export models for embedding. The data to be exported are the .moc3 file, the .model3.json file, and the folder containing the textures. Please put them in one folder.

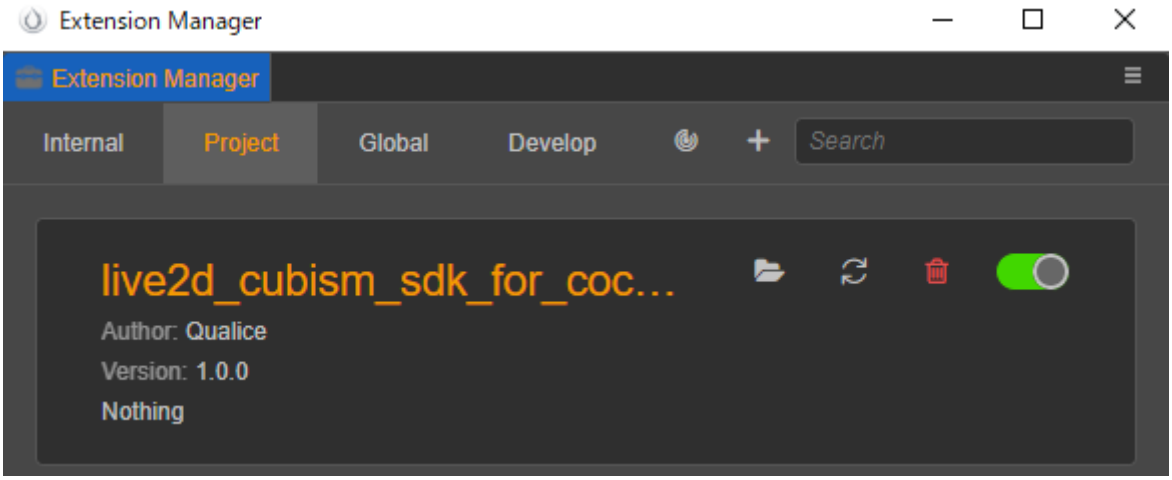
Import SDK into your project

- Create a new Cocos Creator project.
    - For more information on creating a new Cocos Creator project, see here for more information about creating a new Cocos Creator project.





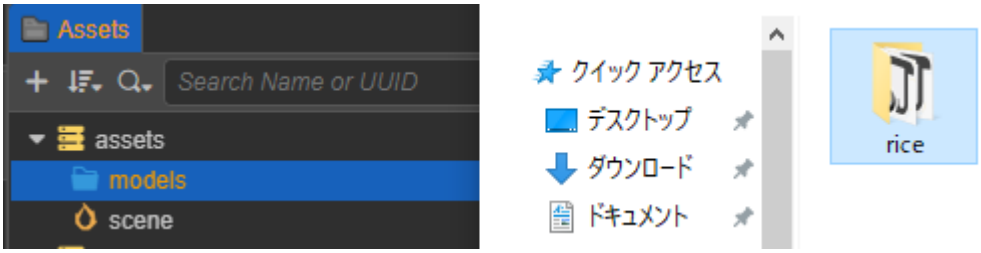




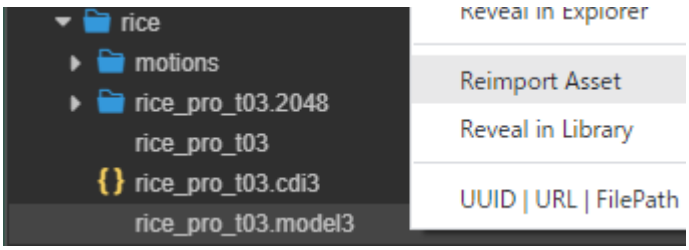
- Add **Cubism SDK for Cocos Creator** to **Extension**.
    - Open **Extension Manager**. (Menu->Extension->Extension Manager)
    - Select **Project** in the Extension Manager and click **Import Extension** ([+] button).
    - Select and open the **Cubism SDK for Cocos Creator (.zip file)**.
    - When the import is complete, close Cocos Creator.
    - You can also create an **extensions** folder in the project root in Explorer and extract the zip file there.
- Build the Extension project.
    - Go to the Extension ({project root}/extensions/{SDK folder}) added above and execute the following commands at the command prompt, etc.
        - npm install
        - npm run build
    - After the build is complete, reopen the project.

Importing Models

Drag and drop a set of built-in models exported from the Cubism Editor into the Asset panel, folder by folder.



After successful import, a Prefab will be automatically generated by Cubism's Importer included in the SDK. If the Prefab is not generated, right-click on the **model3.json** file and click on **Reimport Asset**.



This generated Prefab is a Live2D model converted to a state that can be handled by Cocos Creator. The display of parameters and parts can be manipulated from the Hierarchy panel at this stage.

The model can be placed by adding the generated Prefab to the Hierarchy panel or Scene panel and running the scene.

About the Original Workflow Method

When importing a model using the Original Workflow method, before dragging and dropping it into the Project window Check Live2D > Cubism > Original Workflow > Should Import As Original Workflow in the menu bar.

Cubism SDK for Cocos Creator is enabled to import using the Original Workflow method by default. If you do not want to use the Original Workflow method, uncheck Live2D > Cubism > Original Workflow > Should Import As Original Workflow in the menu bar.

If you import with this checked, an additional OW method component will be attached to the generated Prefab.

Animation Playback

This tutorial is about how to play an embedded animation file exported from the Cubism Editor on a model in a Cocos Creator.

This explanation is based on the assumption that the model will be added to the project in which［Import SDK-Placing Models］was performed.

Summary

To play Cubism animations on a Cocos Creator project, a motion file in .motion3.json format is required. See here for information on exporting motion files.

The SDK provides an Importer for motion3.json as well as models, and motion3.json is automatically converted to AnimationClip, Cocos Creator's animation format, upon import.

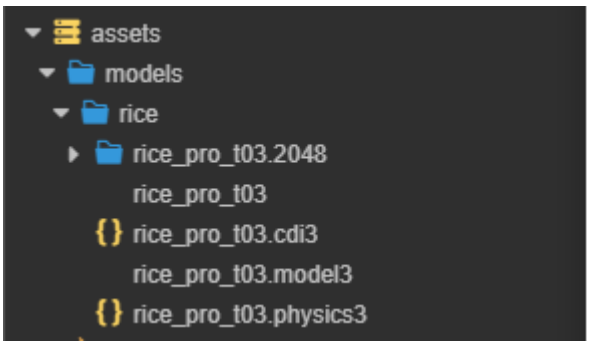
By using the converted AnimationClip, Cocos Creator can handle animations using only the built-in functions of Cocos Creator, without using Live2D functions in Cocos Creator.

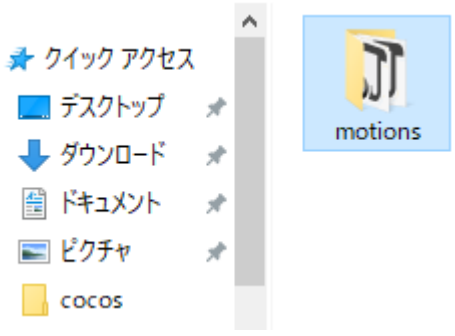
The steps to play a motion in a Cocos Creator project are as follows

- Import the motion file
- Play the AnimationClip

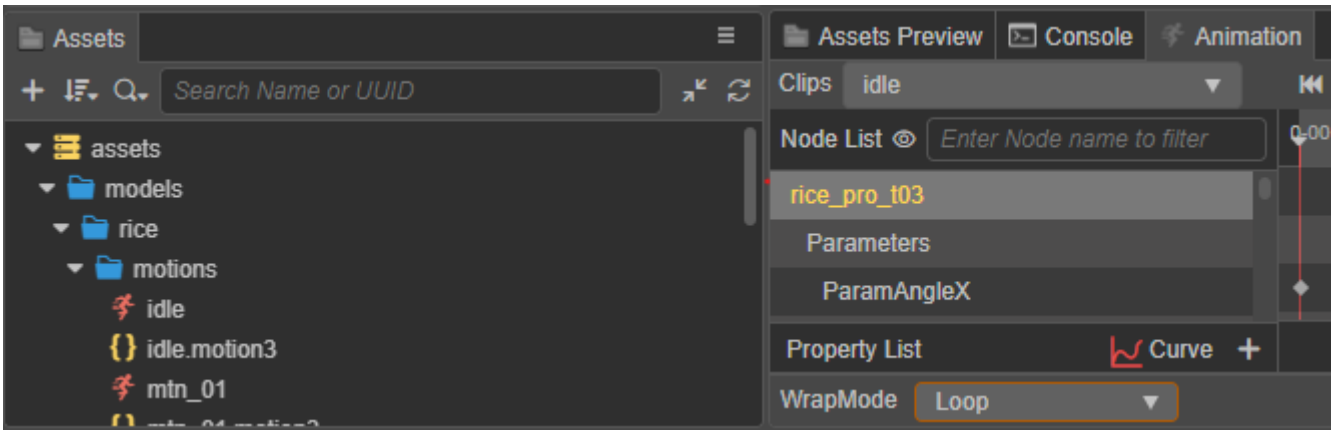
Import motion files

Drag and drop a motion file exported from the Cubism Editor into the Project view, along with the folder containing it.





Then, an AnimationClip is generated from motion3.json, as shown in the following image. Loop can also be set on the generated AnimationClip.

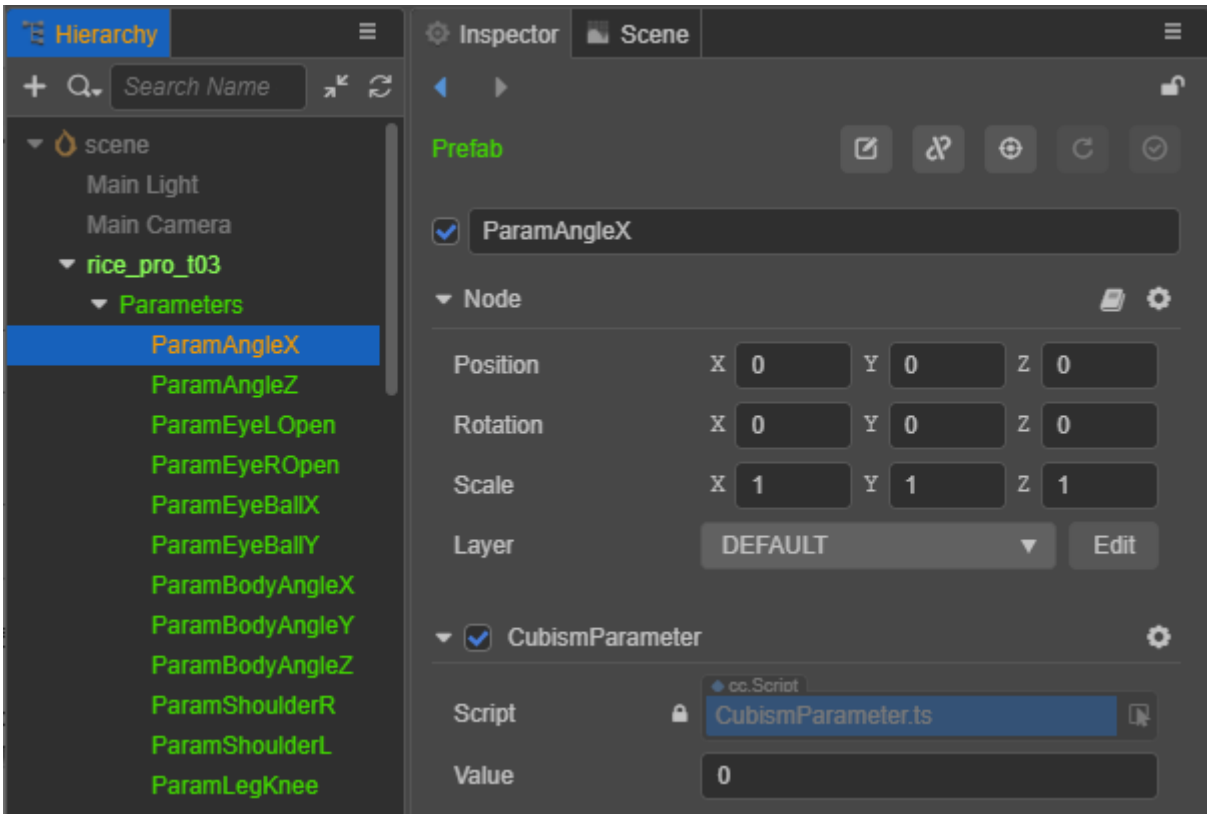


TIPS

This AnimationClip has a curve with values to set for each parameter of the model, and the properties for the parameters set by this curve are located in the following hierarchy of the model's Prefab. (Value is hidden on Inspector)

> [Root of Model]/Parameters/[parameter ID]/Cubism Parameter/Value

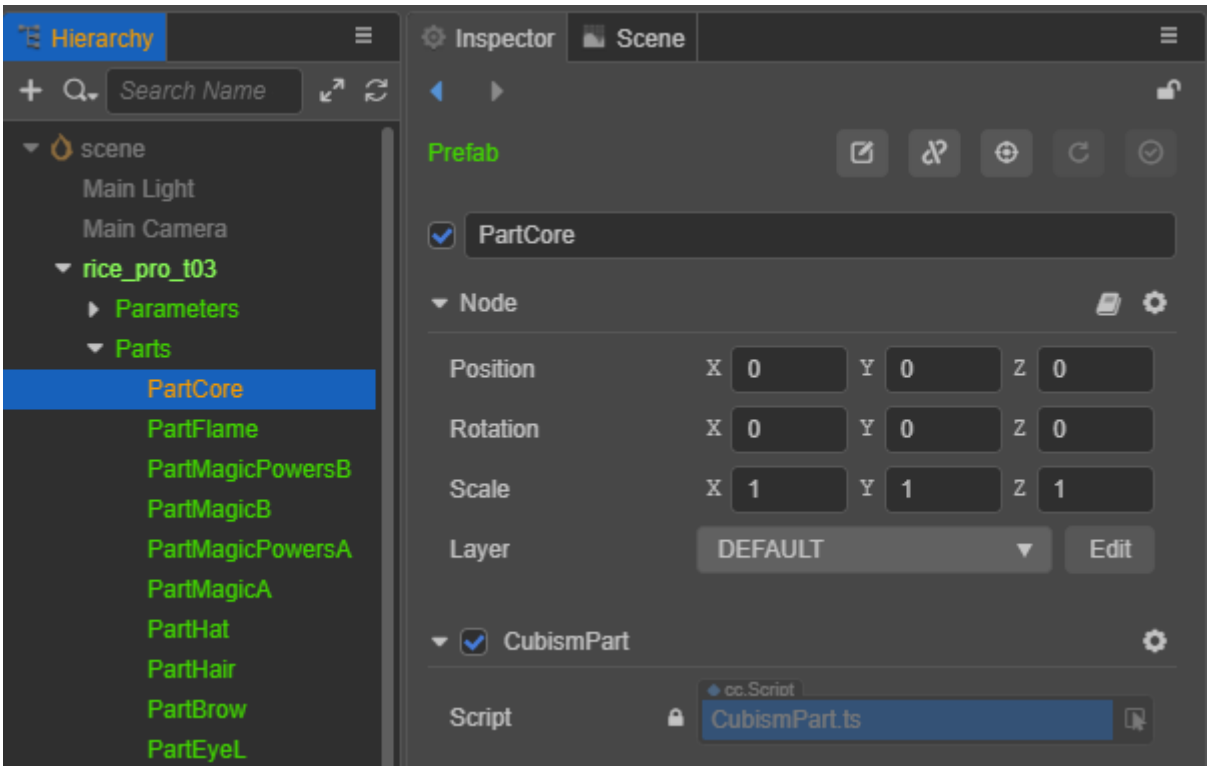
The maximum and minimum values to be set for Value are different for each parameter ID, but values outside that range are treated as maximum or minimum values.



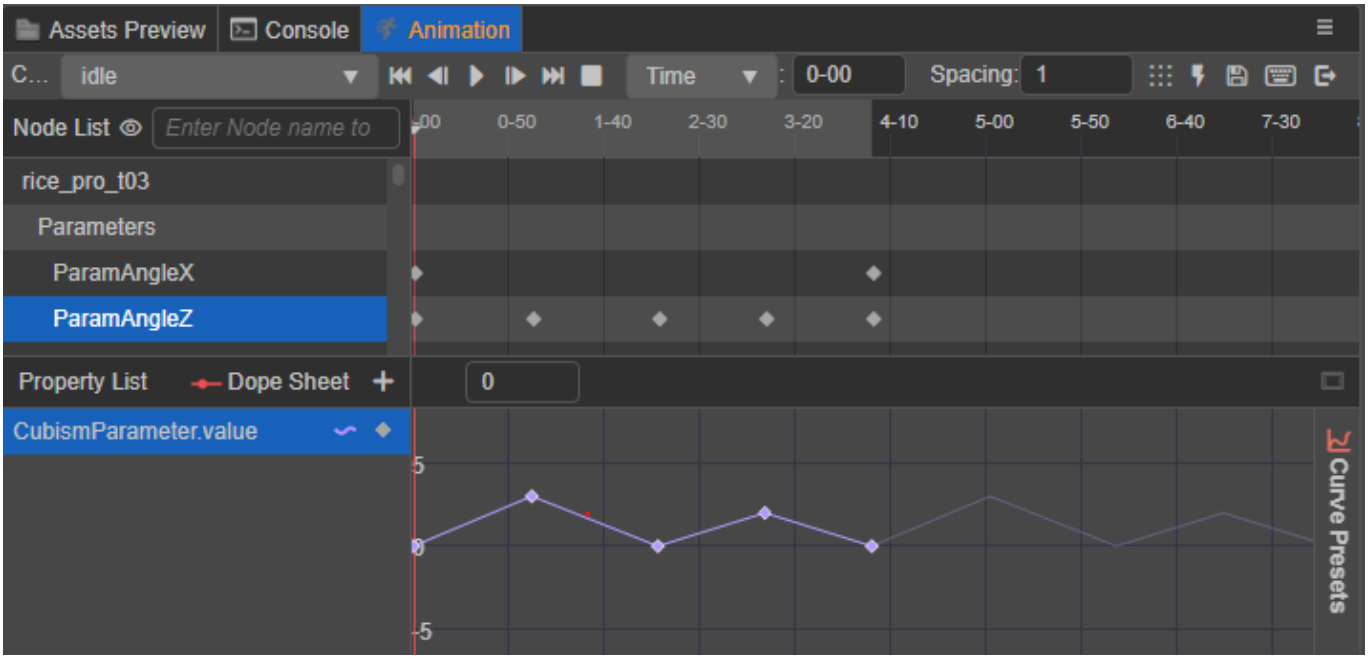
The opacity of the part is also located in the following hierarchy of Prefab. (Opacity is hidden on Inspector)

> [Root of Model]/Parameters/[Part ID]/Cubism Part/Opacity

The value set for Oacity ranges from 0 to 1. Values outside this range are treated as 0 or 1.



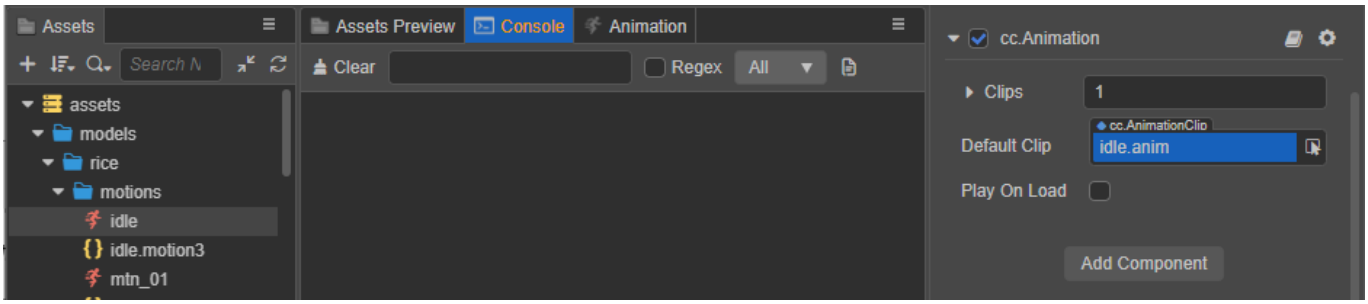
By specifying these properties, you can manipulate model parameters and part opacity from an AnimationClip or program created in Cocos Creator.



Play AnimationClip

To play an AnimationClip in Cocos Creator, use Animator.

- Drag and drop the Prefab of the model into the Hierarchy.
- In the Prefab Inspector of the model, add the **Animation** component.
- Drag and drop the AnimationClip generated by "Import Motion Files" onto the **Default Clip** of the **Animation** component.
- Check the **Play on load** checkbox in the **Animation** component.



When a Scene is executed in this state, the animation is played back.

Tips

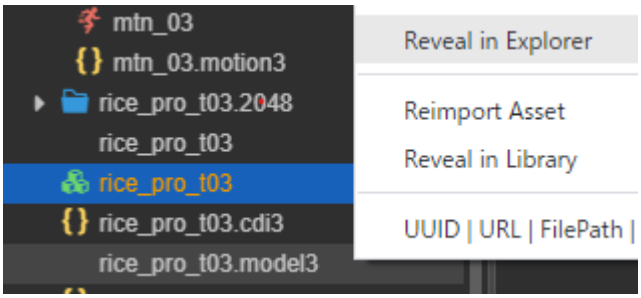
In Live2D Cubism SDK for Cocos Creator, the fade time set in motion3.json is disabled by default.

Update imported models

When updating a model that has been imported into a project, such as when modifying a model, if you drag and drop it back into Cocos Creator If you drag and drop the model into Cocos Creator again, it will be imported as a new model without being overwritten.

To update a model, please follow the steps below.

Right-click on the moc3 file, model3.json file, etc., and click [**Reveal in Explorer**] on Windows, or click on [**Reveal in Finder**] on Mac.



This will open Windows Explorer or Mac Finder in the hierarchy of the file you right-clicked. In the Explorer or Finder, overwrite the file to be updated.

Finally, right-click model3.json or the folder containing it in Cocos Creator's Asset Manager and click on[**Reimport Asset**] to update the Prefab of the model.

About the Original Workflow Method

Whether the updated Prefab is set using the conventional or the Original Workflow method will take precedence over the setting at the time of re-importation.

For example, if a Prefab is generated with the check box checked and then updated with the check box unchecked, the updated Prefab will not have any OW method components attached to it.

Updating Model Parameters

This section is an explanation of the methods and notes on updating parameters attached to the model.

Summary

In Cubism SDK for Cocos Creator, updates are automatically performed from scripts attached to nodes created by **toModel()**.

In addition, parameters can be updated from a separate script using **parameter.value = value;**. Note that parameter updates must be done using **lateUpdate()** from the Cocos Creator event function execution order. The same care should also be taken when the order of script invocation is set with **ScriptExecutionOrder**.

Detail

How to update with Cubism SDK for Cocos Creator

Cubism SDK for Cocos Creator automatically updates through `CubismRenderController.ts` attached to the generated Prefab. This can be used to update the drawing by applying values changed by the animation to parameters. To apply a value to a parameter, the description would look something like this:

```
lateUpdate(deltaTime: number)
{
  parameter = model.parameters[index];
  parameter.value = value;
}
```

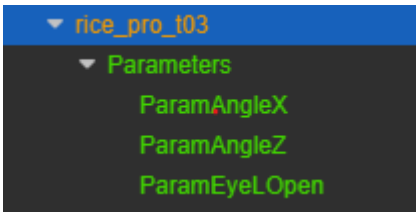
Examples of specific descriptions are:

```
import { _decorator, Component } from 'cc';
import ComponentExtensionMethods from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/ComponentExtension
Methods';
import CubismModel from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/CubismModel';
const { ccclass, property } = _decorator;

@ccclass('ParameterUpdateSample')
export class ParameterUpdateSample extends Component {
    _model: CubismModel | null = null;
    _t: number = 0.0;

    start() {
        this._model = ComponentExtensionMethods.findCubismModel(this, true);
    }

    lateUpdate(deltaTime: number) {
        this._t += deltaTime * 4;
```

```
            let value = Math.sin(this._t) * 30;

            let parameter = this._model.parameters[1];
            parameter.value = value;
        }
    }
```

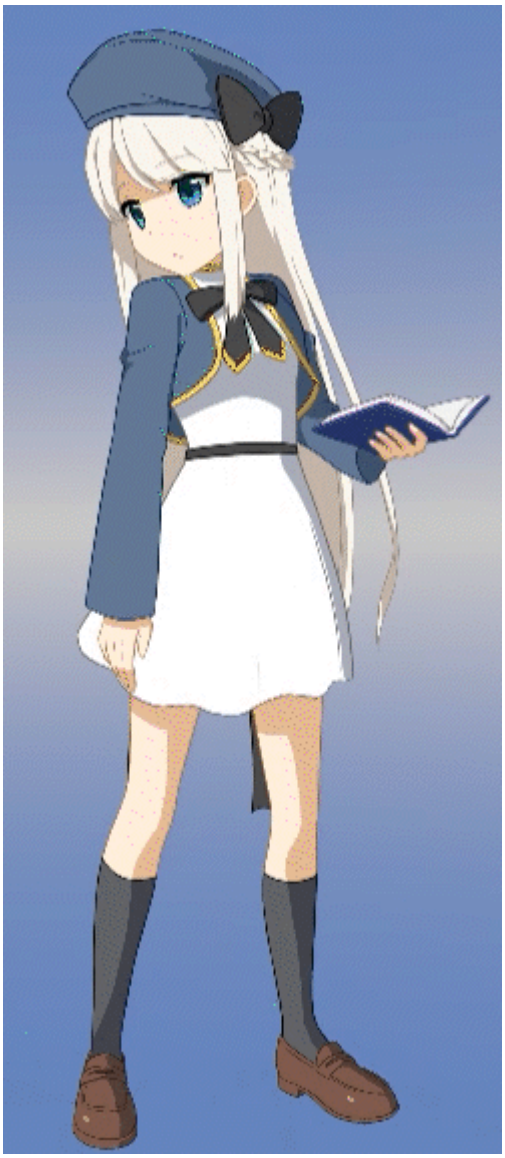
This is the form This is the code used in the GIF described below.

The value of model.parameters[index] is set to "1" because the ParamAngleZ parameter of rice is the first one (starting from 0).



Note that you must use **lateUpdate()** to update parameters in scripts using the Cubism SDK for Cocos Creator. Cocos Creator's animation is processed after `update()`. Therefore, if you change the value of a parameter in `update()`, it will be overwritten by the animation. To avoid this, you need to update the parameters using `lateUpdate()`, which is called after the animation has been processed.

The following GIFs are Scenes in which ParamAngleZ values are manipulated from a script on the model playing the animation. The left model is manipulating the parameter values from `update()` and the right from `lateUpdate()`, and you can see that the value changes have been overwritten in the left one.



TIPS1

The GIF shown in the example is set up by overwriting the animation parameters. Therefore, the parameters that are updated from the script are not applied to the animation.

If you want to use parameters in combination with animation, you need to change the blend mode using **CubismParameterBlendMode** included in Live2D.Cubism. There are three blend modes

- Override... Override and update parameters
- Additive... Additive... Update parameters by adding them
- Multiply... Updates parameters by multiplying them

We encourage you to use the blend mode that best suits your application. The following is an example code.

```
{
  parameter = model.paramters[index];

  // Override
  CubismParameterExtensionMethods.blendToValue(parameter,
CubismParameterBlendMode.Override, 値);
```

```
    // Additive
    CubismParameterExtensionMethods.blendToValue(parameter,
  CubismParameterBlendMode.Additive, 値);

    // Multiply
    CubismParameterExtensionMethods.blendToValue(parameter,
  CubismParameterBlendMode.Multiply, 値);
  }
```

TIPS2

Although model parameters can be obtained by the method described in the update method in Cubism SDK for Cocos Creator, specific parameters can also be obtained by the following method.

This is the code to retrieve CubismParameter by using **findByIdFromParameters()** included in Live2D.Cubism.Core.

```
import { _decorator, Component } from 'cc';
import ArrayExtensionMethods from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/ArrayExtensionMeth
ods';
import ComponentExtensionMethods from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/ComponentExtension
Methods';
import CubismModel from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/CubismModel';
import CubismParameter from
'../extensions/live2d_cubismsdk_cocoscreator/static/assets/Core/CubismParameter';
const { ccclass, property } = _decorator;

@ccclass('ParameterUpdateSample')
export class ParameterUpdateSample extends Component {
  private _model: CubismModel | null = null;
  private _paramAngleZ: CubismParameter | null = null;

  @property({serializable: true})
  public parameterID: string = "ParamAngleZ";

  protected start() {
    this._model = ComponentExtensionMethods.findCubismModel(this, true);

    this._paramAngleZ ??= ArrayExtensionMethods.findByIdFromParameters(
        this._model?.parameters,
        this.parameterID ?? ''
      );
  }
}
```

By attaching a script with this code to the root of the model, the specified parameter (ParameterID) can be obtained.

TIPS3

Another point to note is that you should also be careful if you are changing the order of script execution with **CubismUpdateExecutionOrder**. Depending on the order in which the scripts are executed, the timing of parameter updates may change, and parameter updates may not work properly.

Update Cubism Components

This section describes how to replace the project that was created by［Import SDK～Placing Models］with the latest package.

Summary

There are two ways to obtain the Cubism SDK: from a package or from GitHub.

The package contains the latest Core libraries and CubismComponents at the time the package was released, as well as examples that use those libraries and components.

To apply Cubism SDK updates from GitHub to your Cocos Creator project, follow the instructions below.

If you are not yet familiar with Cocos Creator, please make a backup of your project before updating.

Tips

The SDK is updated by opening the latest zip and overwriting it with the location where the SDK is installed, e.g., in Explorer. The SDK installation location for your project is `[Cocos Creator project path]/extensions/live2d_cubismsdk_cocoscreator/`. After overwriting the SDK, be sure to open the `Extension Manager` from the `Extension` menu of Cocos Creator and perform `Reload`.

Get the latest packages

Obtain the latest package from the Cubism SDK download page.

Apply the latest packages to your Cocos Creator project

When overwriting an existing Cocos Creator project, Cocos Creator may continue to use Cubism Core and may not be able to update the project.

In this case, follow the steps below to update. 1.

1. save all the scenes currently in use. 2. (2) After saving is completed, create a new scene by pressing "Ctrl+N" or other keys. (3) After the new scene is created, press "Ctrl+N" or other keys. Restart Cocos Creator with the new scene created. 4. After Cocos Creator starts, overwrite the latest package.

If there are no compile errors after overwriting, the Cubism SDK update for the Cocos Creator project is complete.

If a compile error occurs at this point, the possible causes are as follows

- The classes included in the package itself are customized.
- A class with the same name as the class added to the package already exists in the project.

Setting up a hit decision

This section describes how to obtain the model's pertinence from the input coordinates. This explanation is based on the assumption that the model will be added to the project in which ［Import SDK～モデルを配置］was performed.

Summary

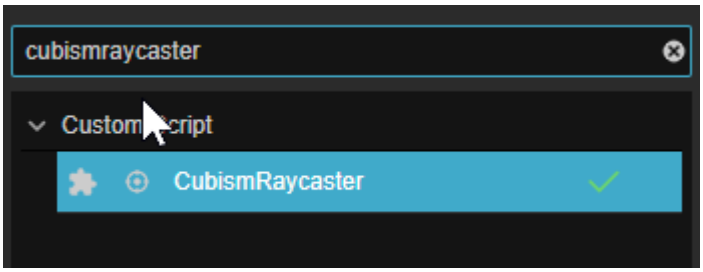
To obtain a hit decision, the Cubism SDK uses a component called Raycast.

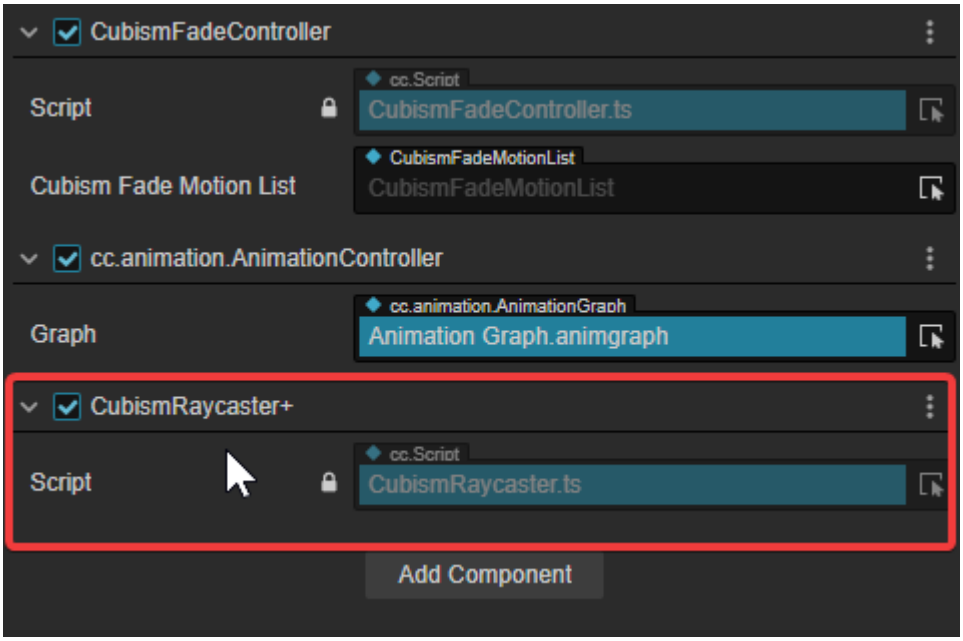
To set up Raycast, do the following three things.

1. Attachment of components to perform Raycast
2. Specify the ArtMesh to be used for the hit detection
3. Get the result of the judgment from CubismRaycaster.raycast

Attachment of components to perform Raycast

Attach a component called [CubismRaycaster], which handles hit detection, to the GameObject that will be the root of the model.

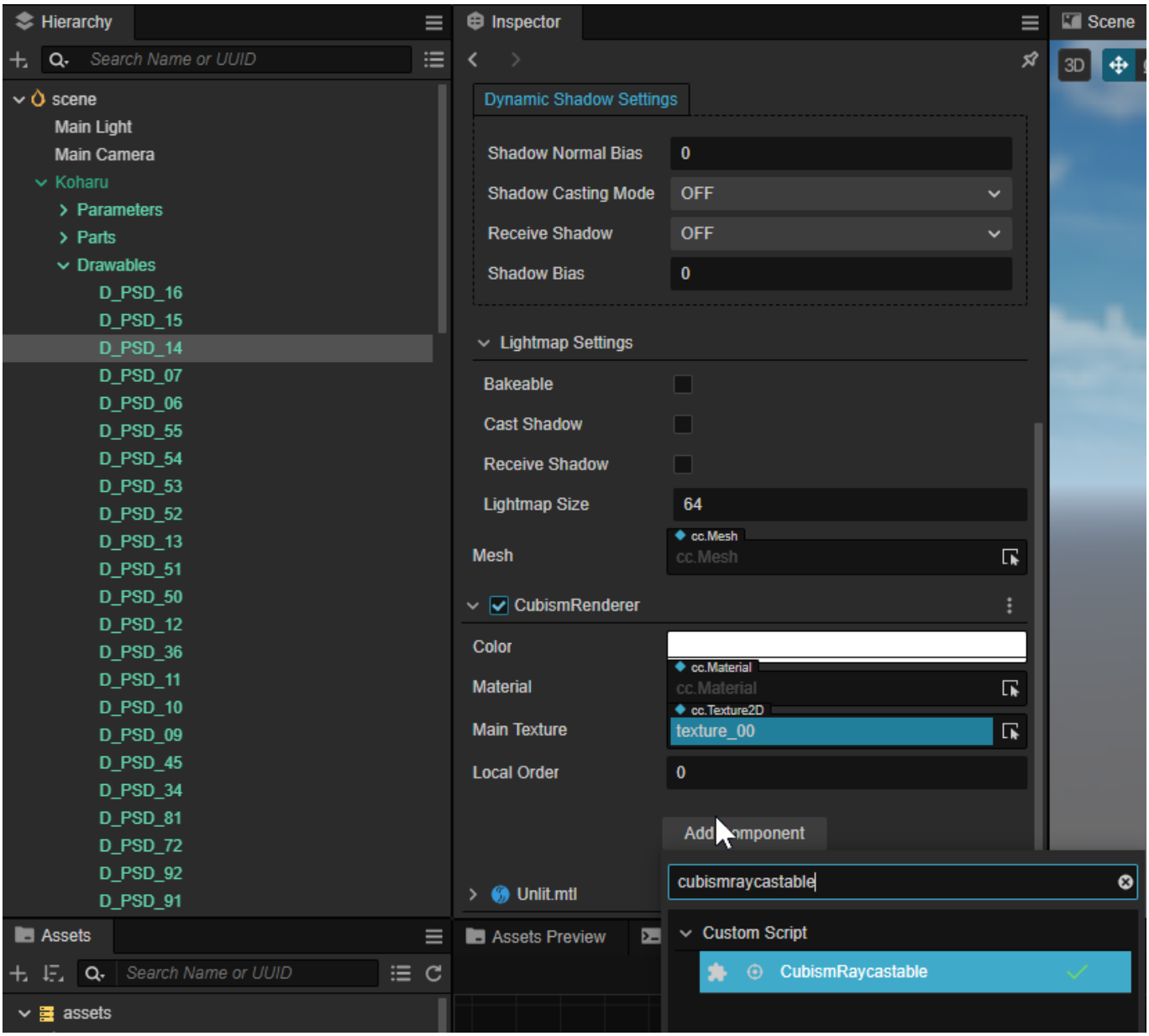




Specify the ArtMesh to be used for the hit detection

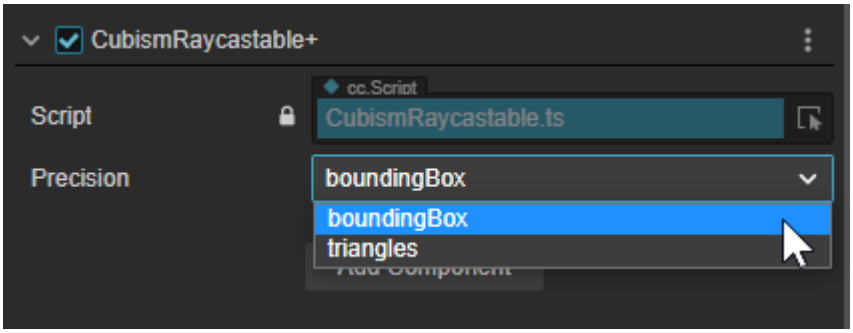
Under [Model]/Drawables/ are located the Nodes that manage each art mesh to be drawn. The name of the Node is the ID of its parameter.

Attach [CubismRaycastable] to the Node that is to be used as the range for the hit judgment.



From CubismRaycastable, you can select the range of the hit detection for the attached mesh.

- Bounding Box : The rectangle surrounding the mesh is used as the hit detection area, which is lighter than Triangles.
- Triangles : The shape of the mesh is used as the hit detection. If you want to judge the range accurately, set this option.



Get the result of the judgment from CubismRaycaster.raycast

Finally, use Raycast() of the CubismRaycaster attached to the root of the model to get the result of the hit decision.

Create a TypeScript script called "CubismHitTest", add the following code and attach it to the root of the model.

```
import { _decorator, Component, Node, input, Input, EventTouch, Camera } from
'cc';
import CubismRaycaster from '../../Framework/Raycasting/CubismRaycaster';
import CubismRaycastHit from '../../Framework/Raycasting/CubismRaycastHit';
const { ccclass, property } = _decorator;

@ccclass('CubismHitTest')
export class CubismHitTest extends Component {
  @property({ type: Camera, visible: true })
  public _camera: Camera = new Camera;

  protected start() {
    input.on(Input.EventType.TOUCH_START,this.onTouchStart,this);
  }

  public onTouchStart(event: EventTouch) {
    const raycaster = this.getComponent(CubismRaycaster);

    if (raycaster == null) {
      return;
    }

    // Get up to 4 results of collision detection.
    const results = new Array<CubismRaycastHit>(4);

    // Cast ray from pointer position.
    const ray = this._camera.screenPointToRay(event.getLocationX(),
event.getLocationY());
    const hitCount = raycaster.raycast2(ray, results);

    // Show results.
    let resultsText = hitCount.toString();
    for (var i = 0; i < hitCount; i++)
    {
      resultsText += "\n" + results[i].drawable?.name;
    }

    console.log(resultsText);
  }
}
```

This completes the setup.

In this state, when a mesh to which CubismRaycastable is attached is clicked in the Scene View during execution, the result of the hit detection is output to the Console View.

Execution example

```
D_PSD_16<CubismDrawable>
D_PSD_15<CubismDrawable>
D_PSD_14<CubismDrawable>
D_PSD_07<CubismDrawable>
```

Eye Tracking Settings

This section explains how to make the model's line of sight, etc., follow the mouse cursor.

Summary

To set up eye tracking, the Cubism SDK uses a component called Lookat.

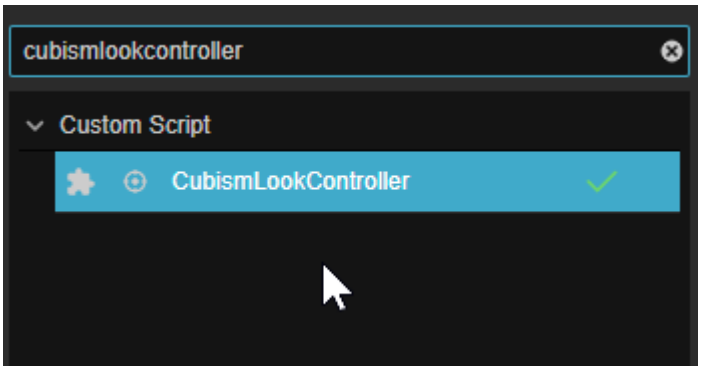
The SDK includes a sample scene called [LookAt] that uses this component. Here is a sample that tracks the position of a Node that rotates above the model's head with the line of sight.
/assets/resources/Samples/Lookat

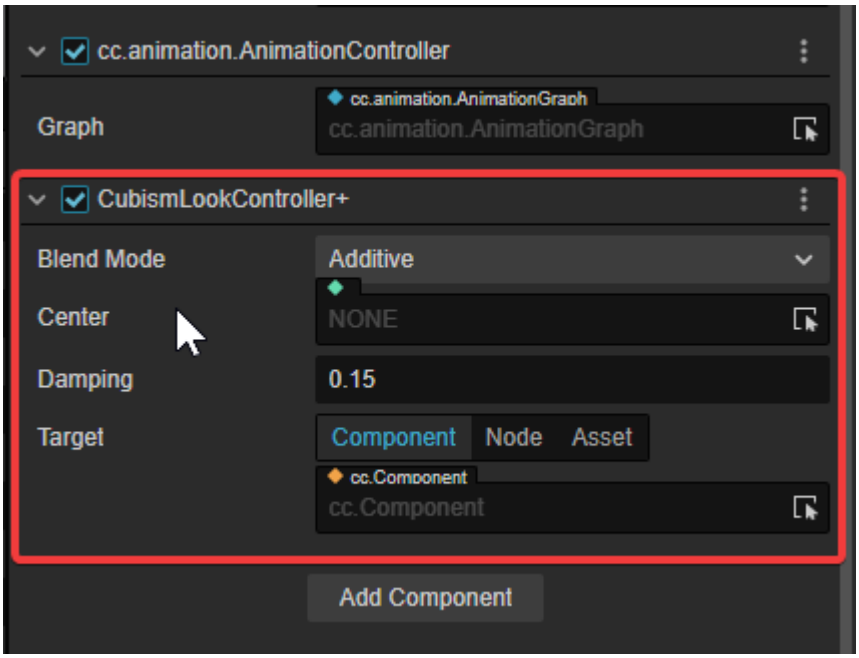
To set up Lookat, do the following three things.

1. Attach a component to manage the Lookat
2. Specify parameters to be followed
3. Setting the object to be followed

Attachment of components to manage Lookat

Attach a component called CubismLookController, which manages eye tracking, to the Node that is the root of the model.





CubismLookController has four setting items.

- Blend Mode: This is the setting for how the parameters reflect the values that vary when tracking the line of sight. The following three settings are available.
  <br
  - Multiply: Multiply the currently set value.
  - Additive: Add to the currently set value.
  - Override: Override the currently set value.
- Center: Set the Node to be treated as the center of the coordinates to be tracked here. The center is the center of the Bounds of the set Node. The Center will be set to the Node under [Model]/Drawables/. If nothing is set for this item, CubismLookController will use the center of the attached Node.
- Damping: The time it takes to follow the target. The smaller the value, the faster the tracking speed.
- Target: Sets the target to be followed. Details are described below.

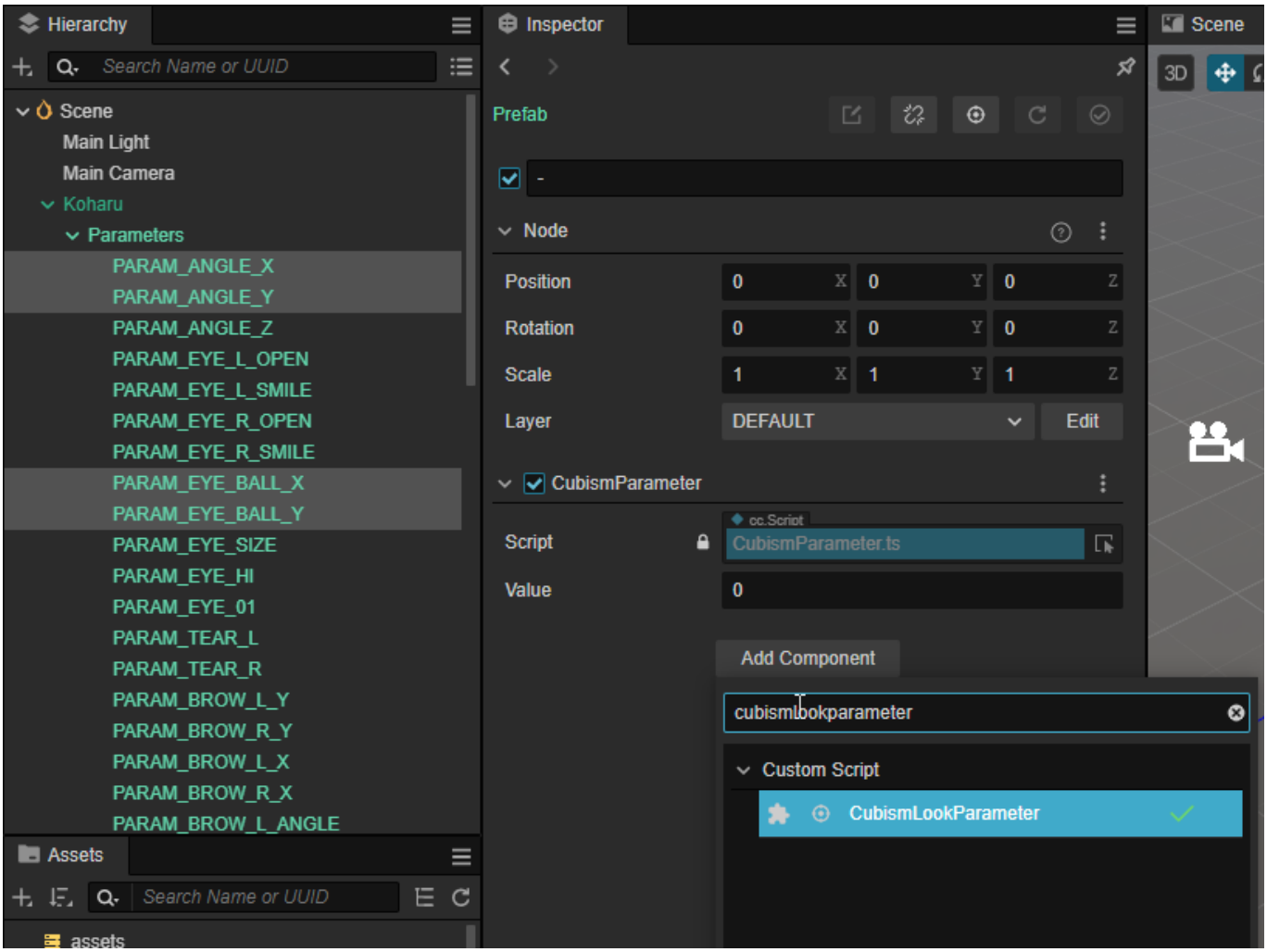
In this case, set Blend Mode to Override.

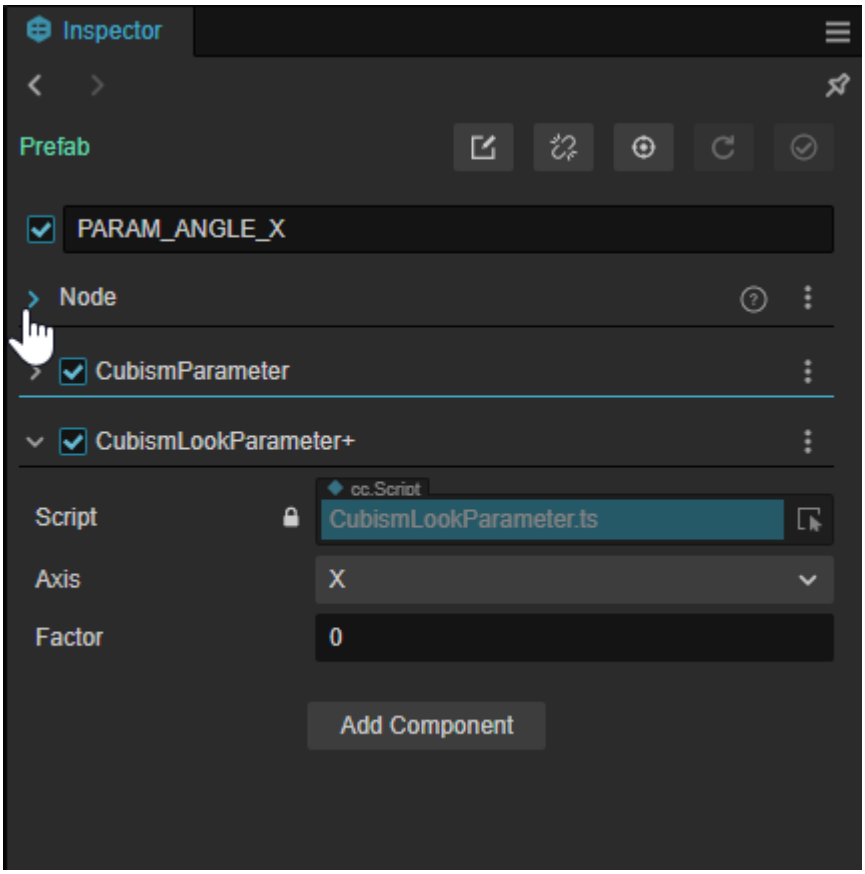
Specifying parameters to be tracked

Under [Model]/Parameters/, there is a Node that manages the parameters of the model. The name set in this Node is the ID of the parameter. These are the same as those that can be obtained from CubismModel.parameters.

Attach a component named CubismLookParameter to the Node for the parameter whose ID you want to track.

If CubismLookParameter is attached to the Node for the parameter, the aforementioned CubismLookController will refer to it and follow it.





CubismLookParameter has two setting items.

- Axis: Specifies how many axes of deformation to treat the set parameters as. For example, if X is specified, it is calculated and set based on the value of Target's X axis.
- Factor: Sets the number to multiply the calculated value by. Since the resulting value ranges from -1 to +1, depending on the parameter, it may be more natural to increase or decrease its range, or invert the + and - values.

Setting the target to follow your line of sight

Finally, prepare the target to be followed.

The [Target] of the CubismLookController component should be set to a component that implements the [ICubismLookTarget] interface.

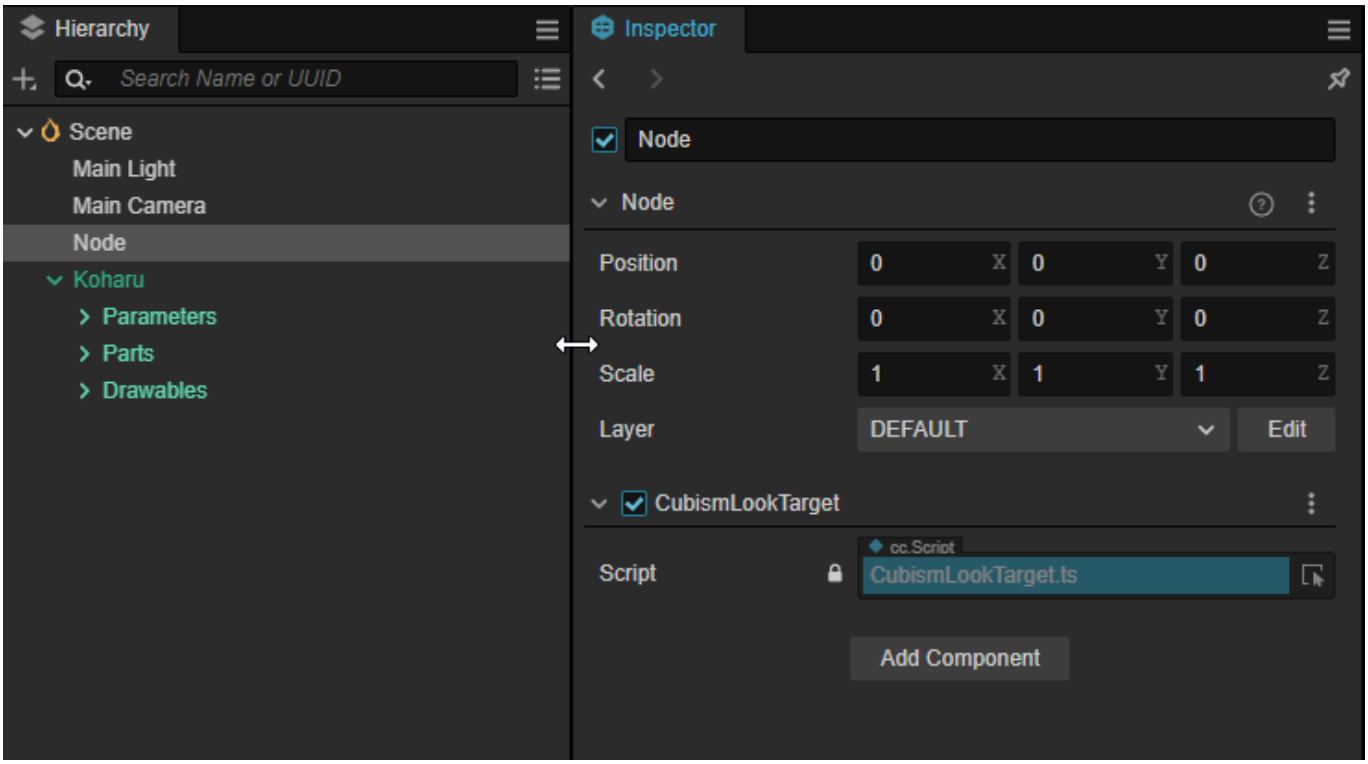
Depending on the target settings, it is possible to set specific conditions, such as the mouse cursor or Node position as the target for eye tracking, or to track only while dragging.

Create a TypeScript script called “CubismLookTarget” and rewrite the code as follows. Here, we are trying to track the coordinates of the mouse while dragging.

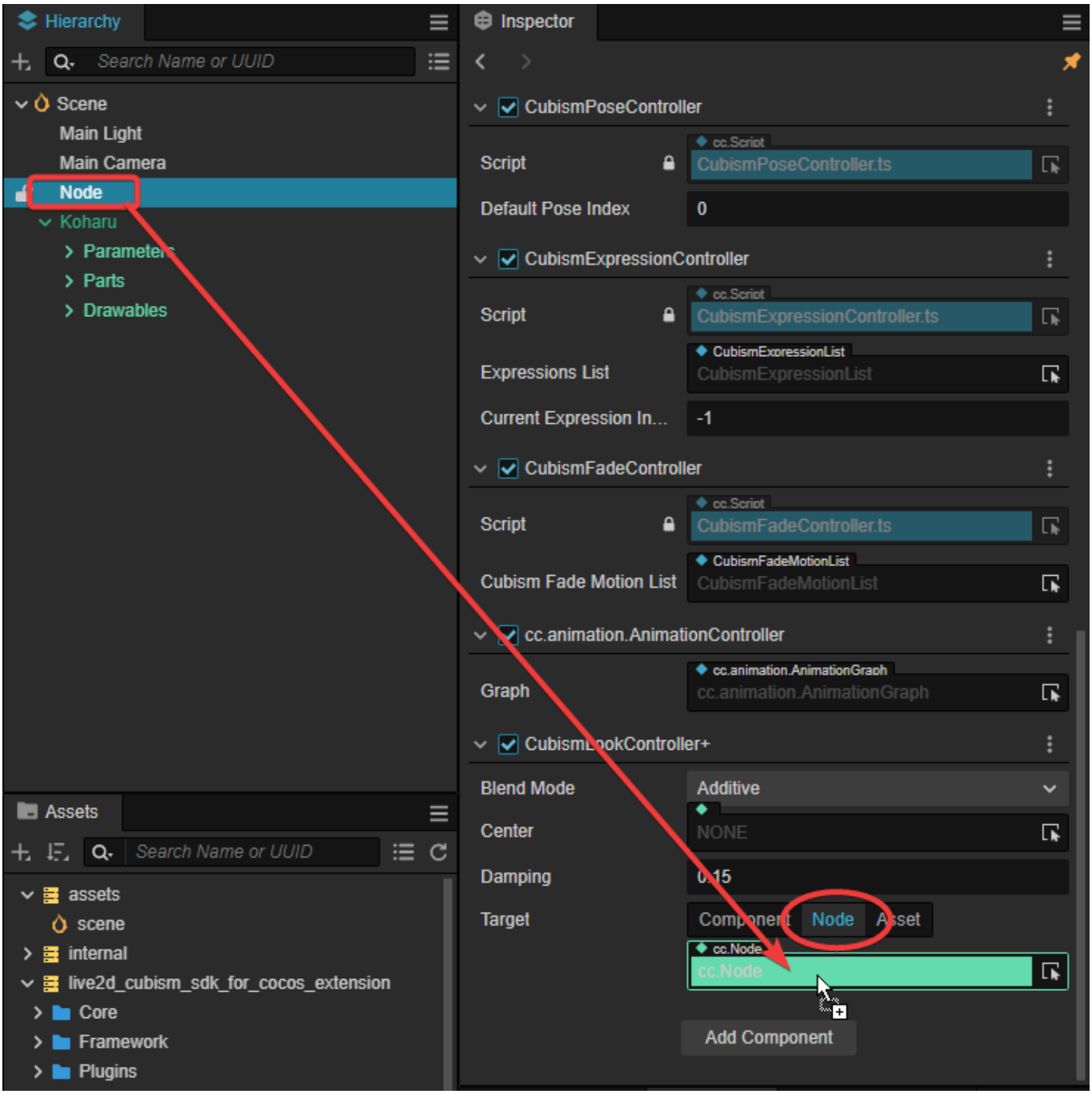
```
import { _decorator, Component, Node, Input, EventTouch, input, math, Vec3,
__private, Camera, Vec2, screen } from 'cc';
import ICubismLookTarget from '../../Framework/LookAt/ICubismLookTarget';
const { ccclass, property } = _decorator;

@ccclass('CubismLookTarget')
export class CubismLookTarget extends Component implements ICubismLookTarget {
  readonly [ICubismLookTarget.SYMBOL]: typeof ICubismLookTarget.SYMBOL =
ICubismLookTarget.SYMBOL;

  private _position: math.Vec3 = Vec3.ZERO;

  public getPosition(): math.Vec3 {
    return this._position;
  }

  public isActive(): boolean {
    return true;
  }

  protected start() {
    input.on(Input.EventType.TOUCH_MOVE,this.onTouchMove,this);
    input.on(Input.EventType.TOUCH_END,this.onTouchEnd,this);
  }

  public onTouchMove(event: EventTouch) {
    let targetPosition = event.getLocationInView();
    targetPosition.x = targetPosition.x / screen.resolution.width;
    targetPosition.y = targetPosition.y / screen.resolution.height;

    targetPosition = targetPosition.multiplyScalar(2).subtract(Vec2.ONE);
```

```
      this._position = new Vec3(targetPosition.x, targetPosition.y,
  this._position.z);
    }

    public onTouchEnd(event: EventTouch) {
      this._position = Vec3.ZERO;
    }
  }
```

Create an empty Node and attach the above CubismLookTarget to it.



Select the model and drag and drop the Node created above from the Inspector view into the [Target] of the CubismLookController.



This completes the setup.

Running the scene and dragging the Scene view with the left mouse button causes the model's line of sight to follow.

Automatic blink setting

This section explains how to make the model blink automatically. The following explanation is based on the assumption that the model will be added to the project in which the ［Import SDK-Placing Models］ step was performed.

Summary

If the model has the standard parameters [ParamEyeLOpen (left eye open/closed)] and [ParamEyeROpen (right eye open/closed)], the Prefab of the model generated by the import will automatically be set to blink.

If you wish to have models that have not been configured above to blink automatically, you can do so by following the procedure described in this article.

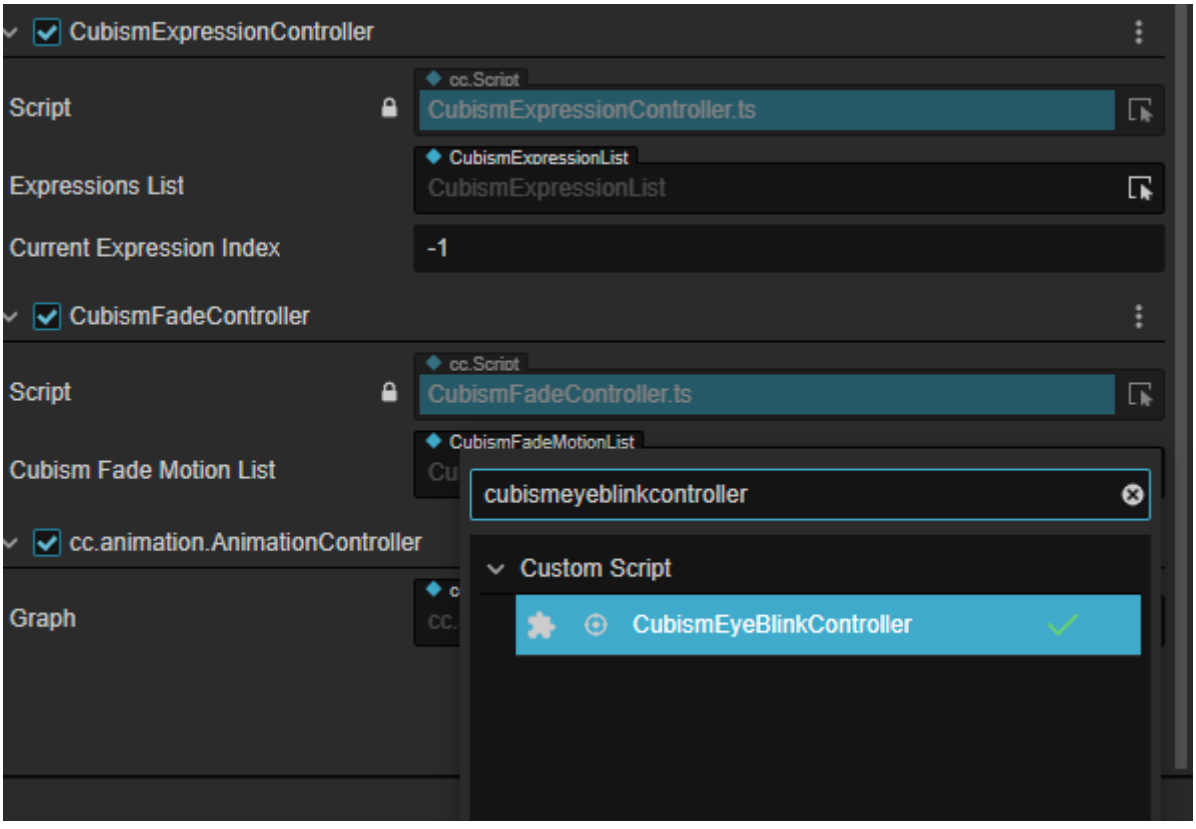
For blink settings, the Cubism SDK uses a component called EyeBlink.

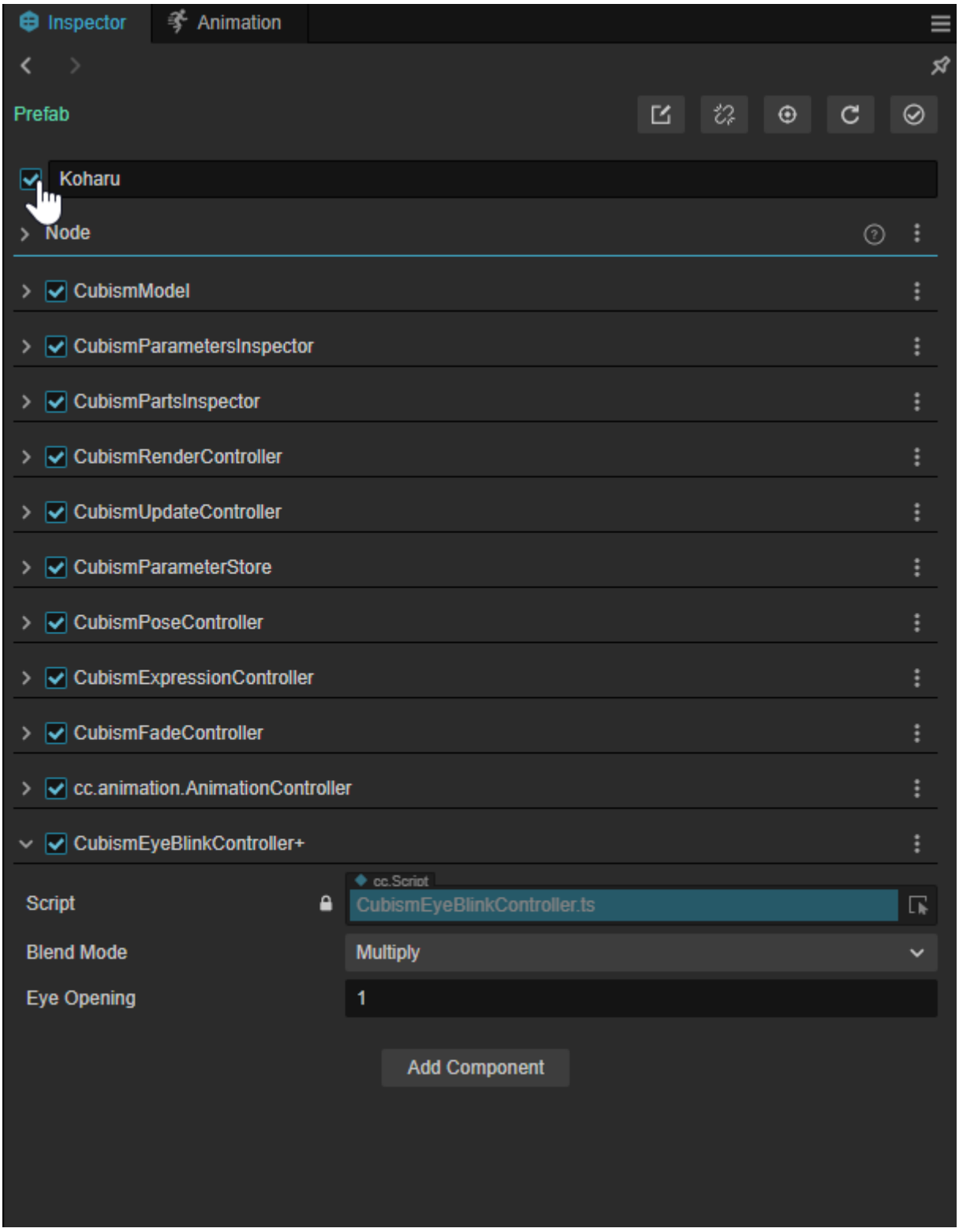
To set up EyeBlink on a Cubism model, do the following three things.

1. Attach a component to manage blinking
2. Specify parameters to make them blink
3. Set up a component to automatically manipulate the value of the parameter for blinking

Attach a component to manage blinking

Attach a component called CubismEyeBlinkController that manages blinking.





CubismEyeBlinkController has two setting items.

- Blend Mode: Specifies how the Eye Opening value is calculated for the value currently set for the specified parameter.
    - Mutiply: Multiply the currently set value by the Eye Opening value.
    - Additive: Add the Eye Opening value to the currently set value.
    - Override: Overwrites the currently set value with the Eye Opening value.
- Eye Opening: The value of eye opening and closing. It is treated as open at 1 and closed at 0. When this value is manipulated from the outside, the value of the specified parameter is also linked.

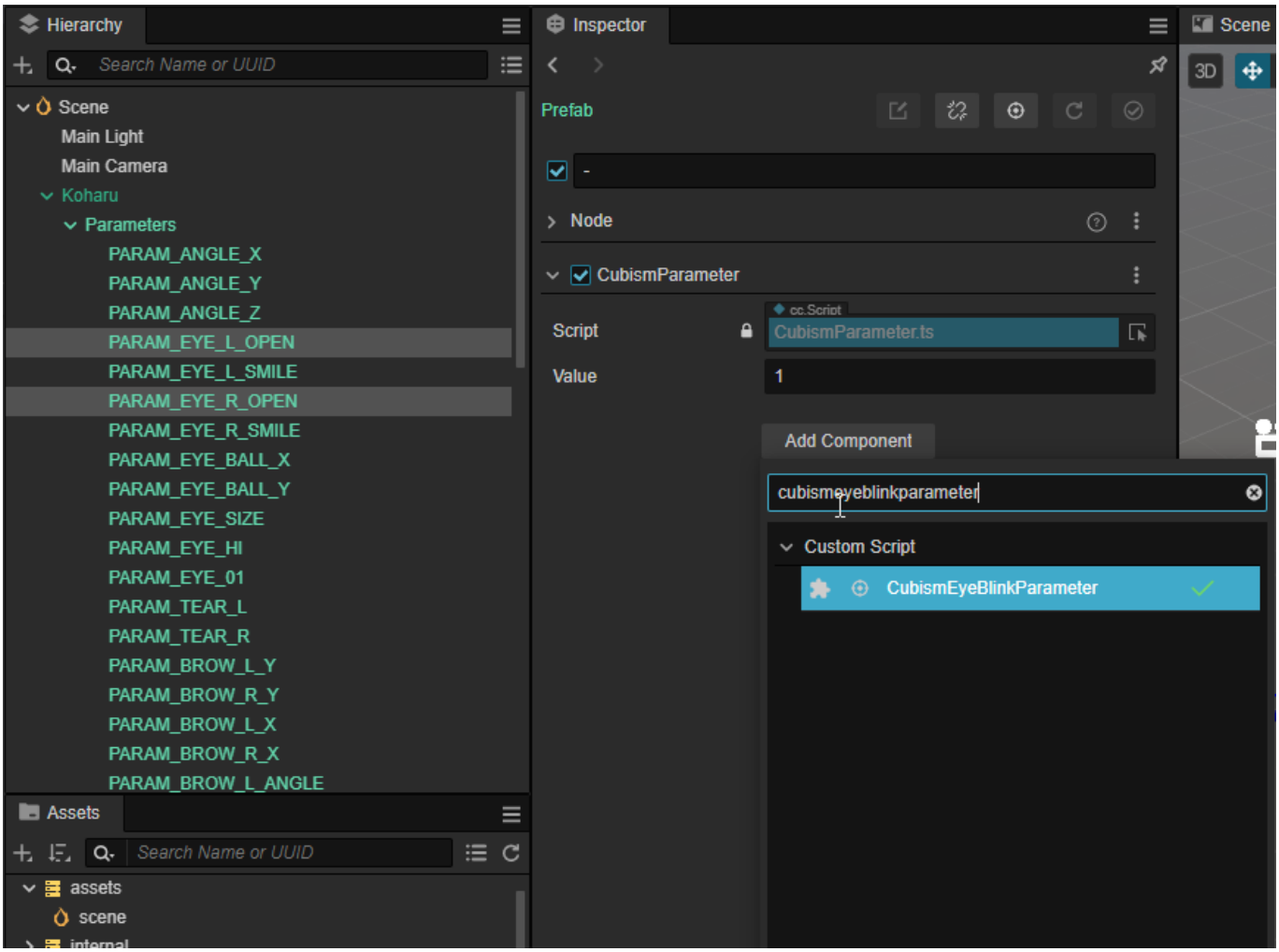
This time, set Blend Mode to [Override].

まばたきをさせるパラメータを指定

Under [Model]/Parameters/ is located the Node that manages the parameters for that model. The name set in this Node is the ID of the parameter. These are the same as those obtained by CubismModel.parameters().

Attach a component named CubismEyeBlinkParameter to the Node for this parameter whose ID is treated as a blink.

If CubismEyeBlinkParameter is attached to the Node for the parameter, CubismEyeBlinkController will refer to it when executing the scene to set the opening and closing of the eyes.

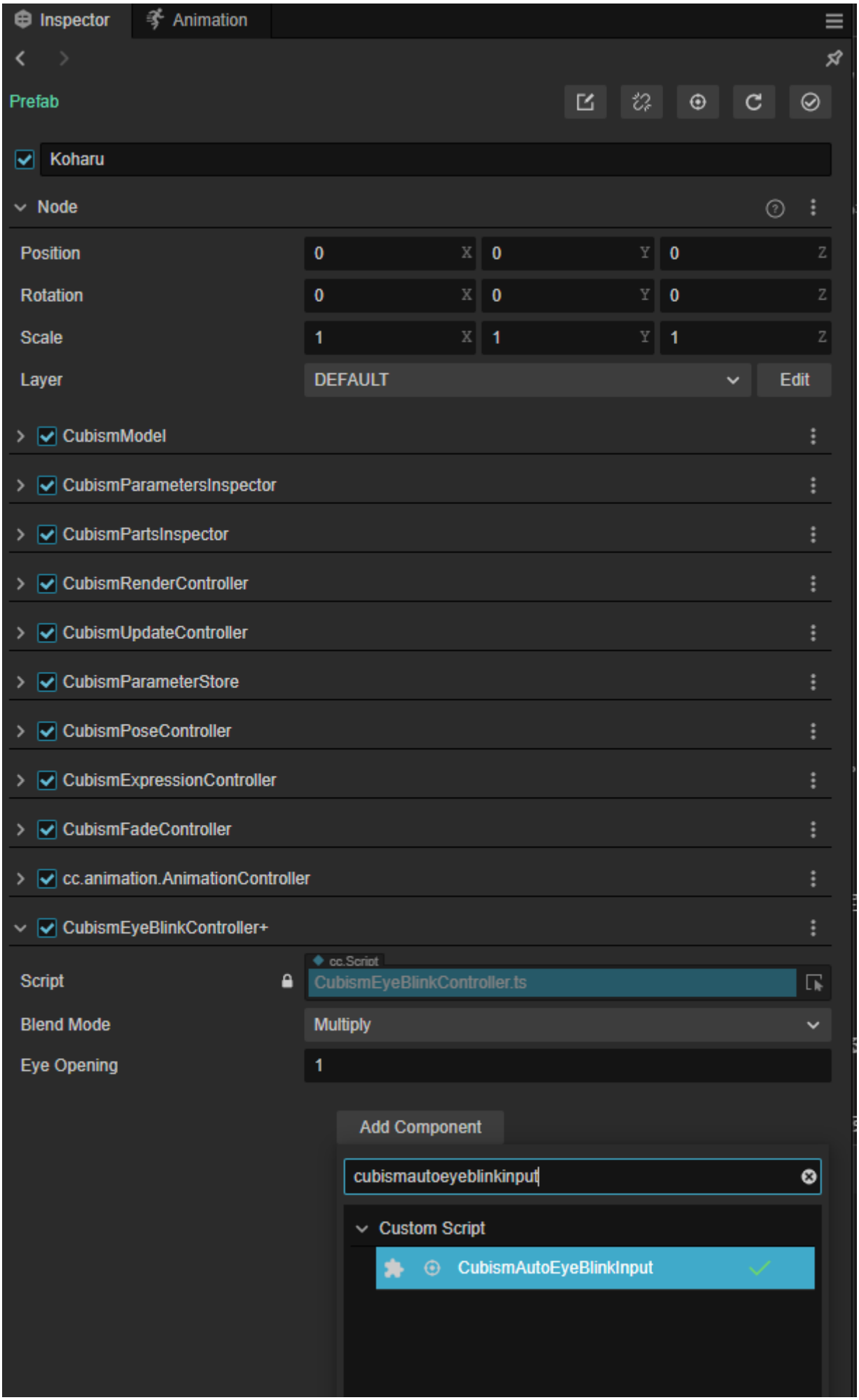


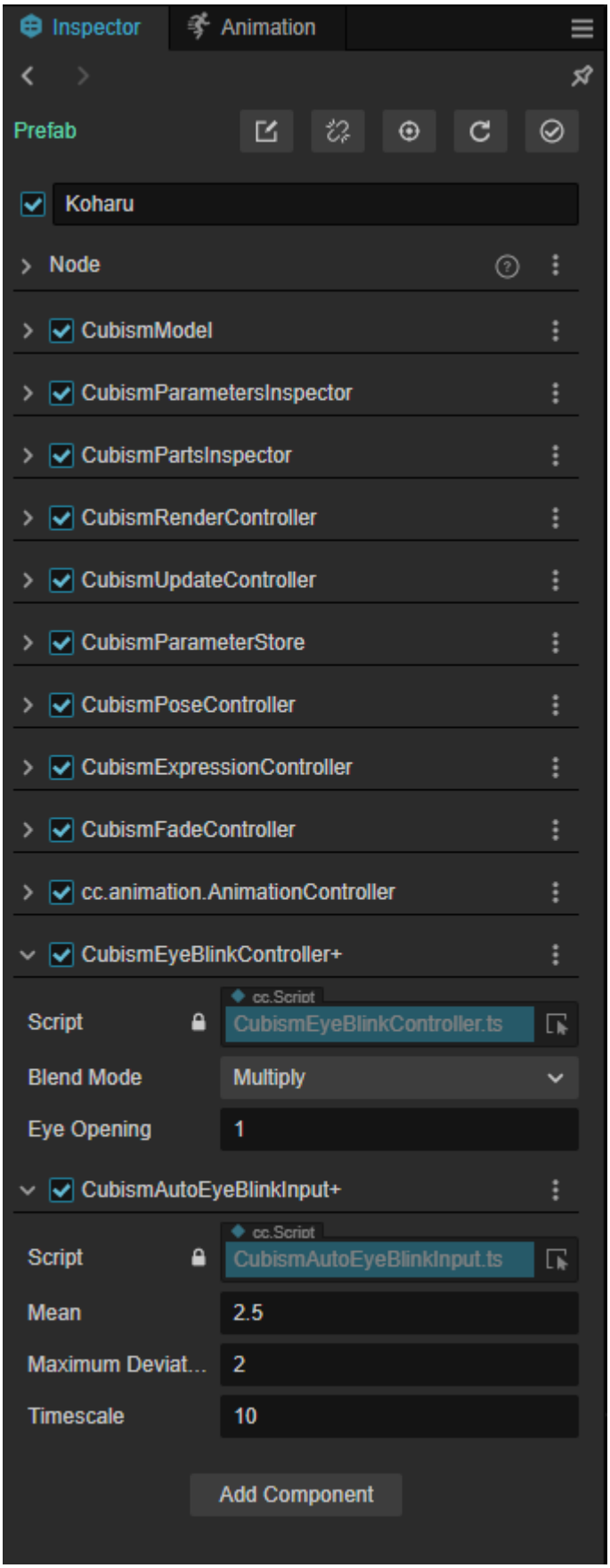
With the above settings, it is now possible to perform blink operations from scripts, etc. However, this is not enough to perform blinking automatically.

To have it blink automatically, you must also set up a component that manipulates the value periodically.

Set up a component to automatically manipulate the value of the parameter for blinking

As with the CubismEyeBlinkController, attach a component called CubismAutoEyeBlinkInput to the root of the model.


Inspector
Animation
Prefab
Koharu
Node
Position
0 X 0 Y 0 Z
Rotation
0 X 0 Y 0 Z
Scale
1 X 1 Y 1 Z
Layer
DEFAULT
Edit
CubismModel
CubismParametersInspector
CubismPartsInspector
CubismRenderController
CubismUpdateController
CubismParameterStore
CubismPoseController
CubismExpressionController
CubismFadeController
cc.animation.AnimationController
CubismEyeBlinkController+
cc.Script
Script
CubismEyeBlinkController.ts
Blend Mode
Multiply
Eye Opening
1
Add Component
cubismautoeyeblinkinput
Custom Script
CubismAutoEyeBlinkInput



CubismAutoEyeBlinkInputには、以下の3つの設定項目があります。

- Mean: Sets the time it takes to blink. The unit is seconds. In practice, the time is calculated by adding the error by Maximum Deviation to this value.
- Maximum Deviation: Sets the maximum value of deviation to be added to the time set by Mean. The value to be set is a number greater than or equal to 0.

. If the character blinks at the time set in Mean above, the cycle will be uniform and the character's behavior will be unnatural.
Therefore, we add random fluctuations to the set time period to make the motion natural.
The actual calculation is as follows.

```
Time to next blink = Mean + (random value between -Maximum Deviation and +Maximum Deviation)
```

- Timescale: Sets the speed at which the eyes open and close. The smaller the value you set, the slower it will be.

This time, set Mean to 2.5, Maximum Deviation to 2, and Timescale to 10.

This completes the automatic blink setup.

If you run Scene in this state, you can make them blink automatically.

Lip-sync Settings

This section explains how to make a model lip-sync (lip-sync) from the AudioSource volume. The following explanation is based on the assumption that the model will be added to the project in which the［Import SDK-Placing Models］step was performed.

Please prepare a separate audio file in a format that can be handled by Cocos Creator in order to obtain and set the volume from AudioSource.

Summary

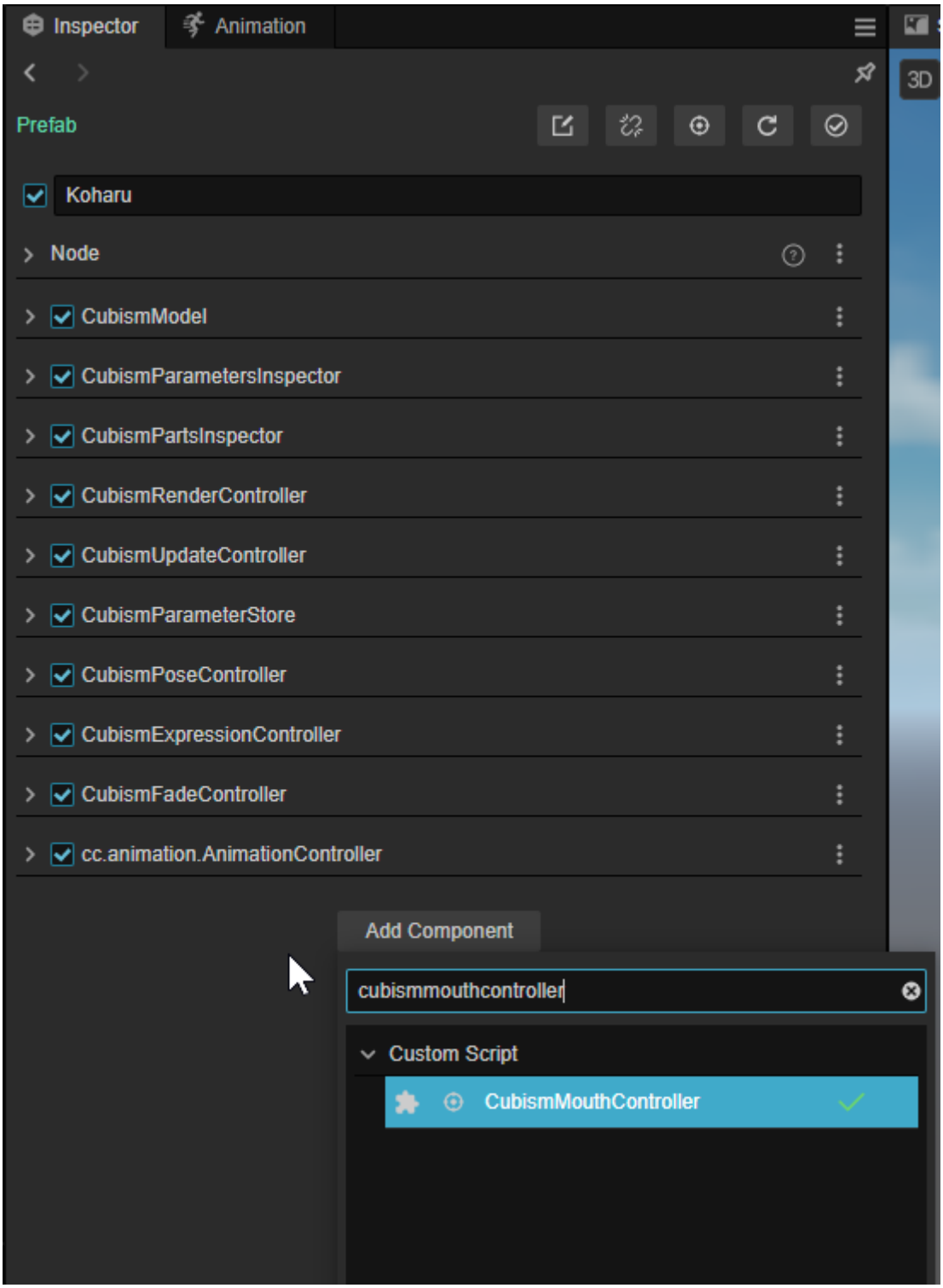
To set up lip-sync, the Cubism SDK uses a component called MouthMovement.

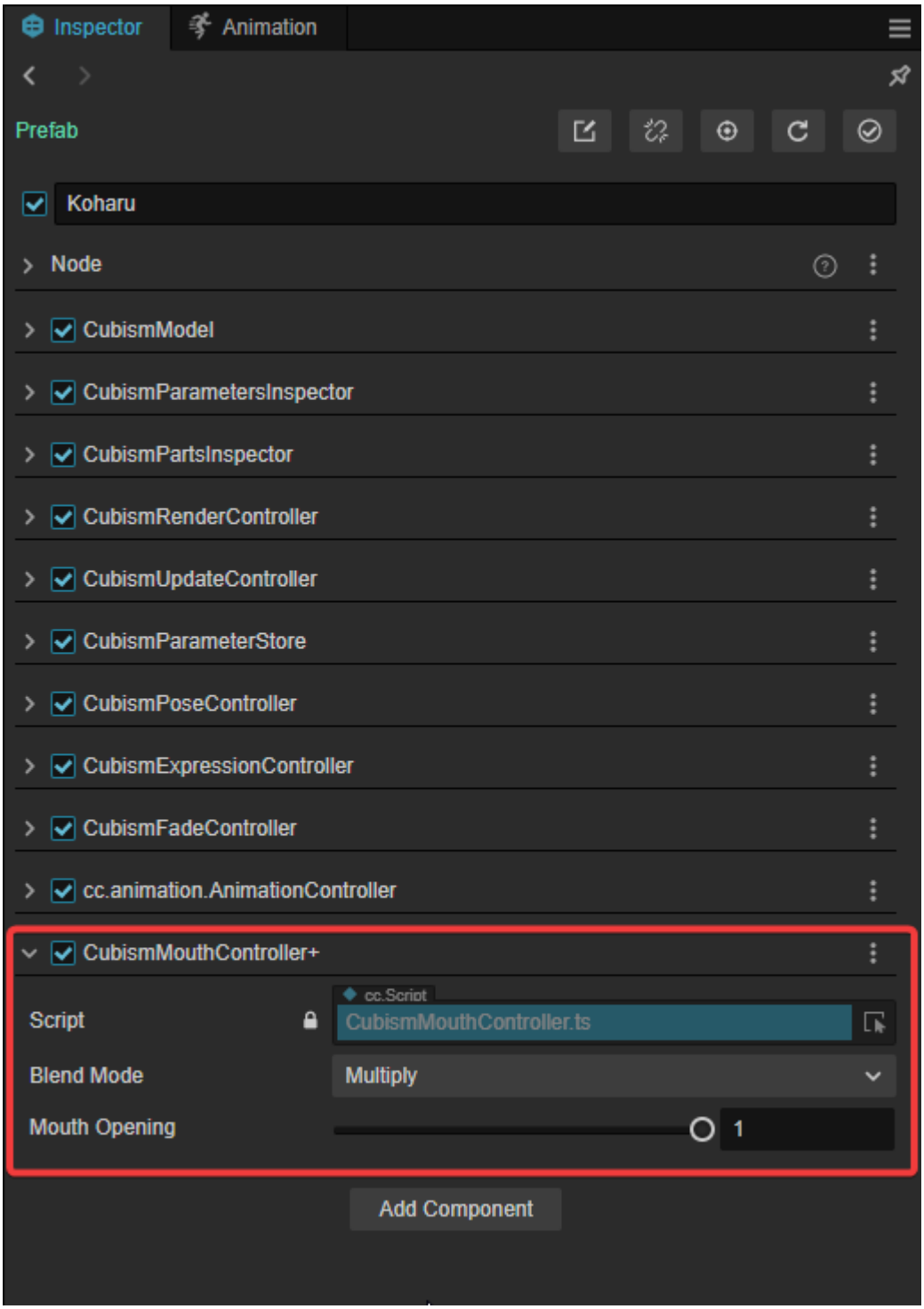
To set up MouthMovement on a Cubism model, do the following three things.

1. Attach components to manage lip-sync
2. Specify parameters for lip-sync
3. Set component to manipulate values of specified parameters

Attach components to manage lip-sync

Attach a component called CubismMouthController, which manages lip-sync, to the Node that is the root of the model.

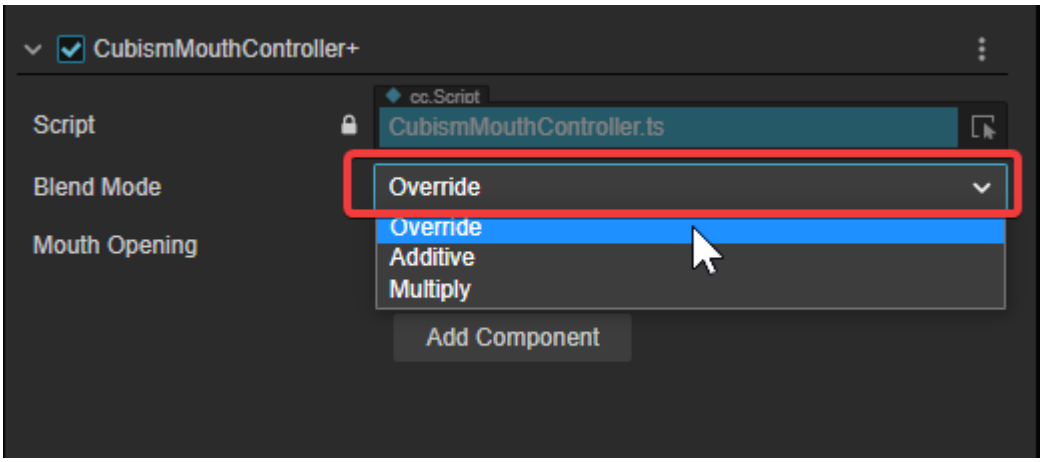

Inspector
Animation
3D
Prefab
Koharu
Node
CubismModel
CubismParametersInspector
CubismPartsInspector
CubismRenderController
CubismUpdateController
CubismParameterStore
CubismPoseController
CubismExpressionController
CubismFadeController
cc.animation.AnimationController
Add Component
cubismmouthcontroller
Custom Script
CubismMouthController



CubismMouthController has two setting items.

- Blend Mode: Specifies how the Mouth Opening value is calculated for the value currently set for the specified parameter.
    - Multiply: Multiply the currently set value by the Mouth Opening value.
    - Additive: Add the value of Mouth Opening to the currently set value.
    - Override: Overwrites the currently set value with the Mouth Opening value.
- Mouth Opening: The value of mouth opening and closing. It is treated as open at 1 and closed at 0. When this value is manipulated from the outside, the value of the specified parameter is also linked.

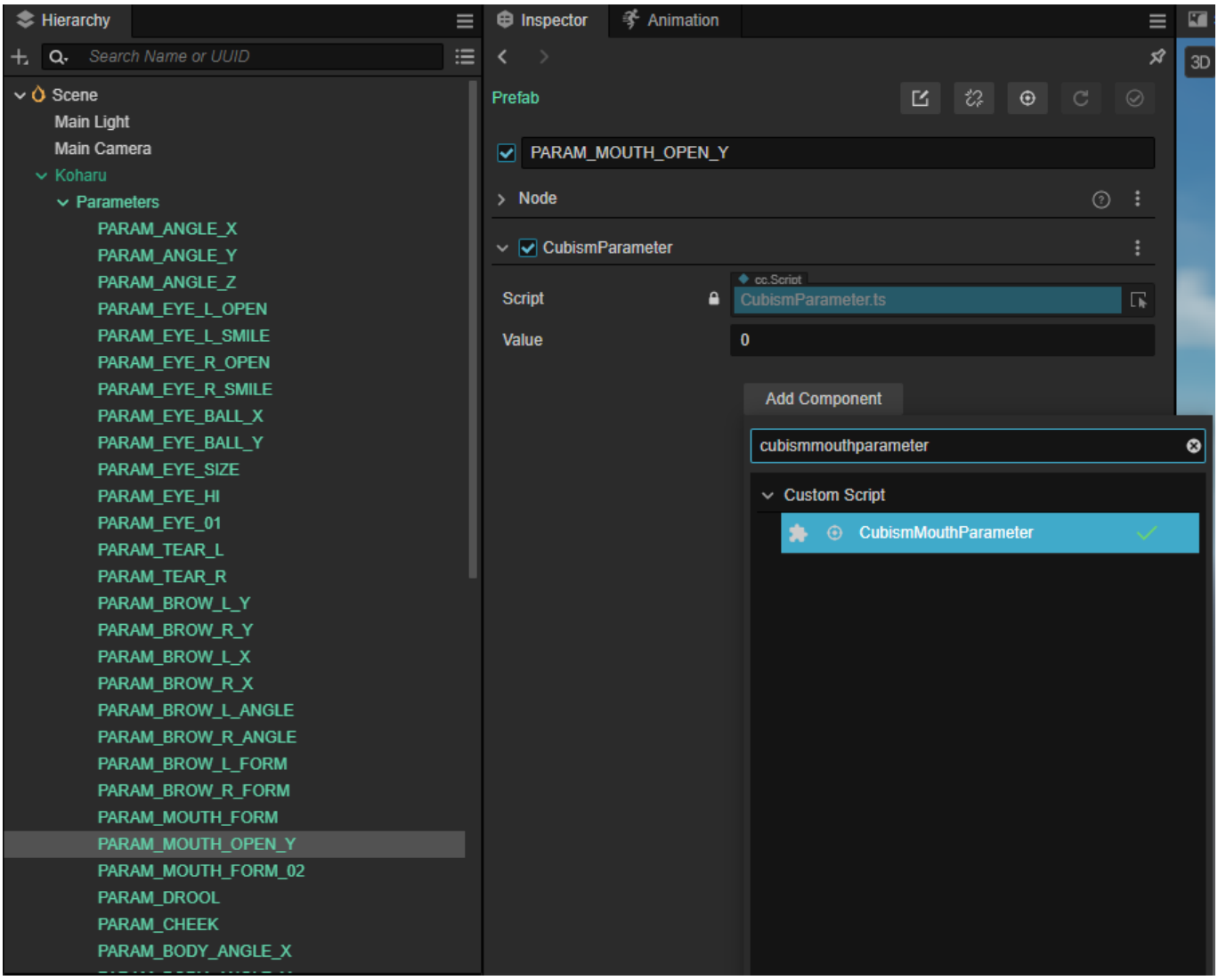
This time, set Blend Mode to [Override].

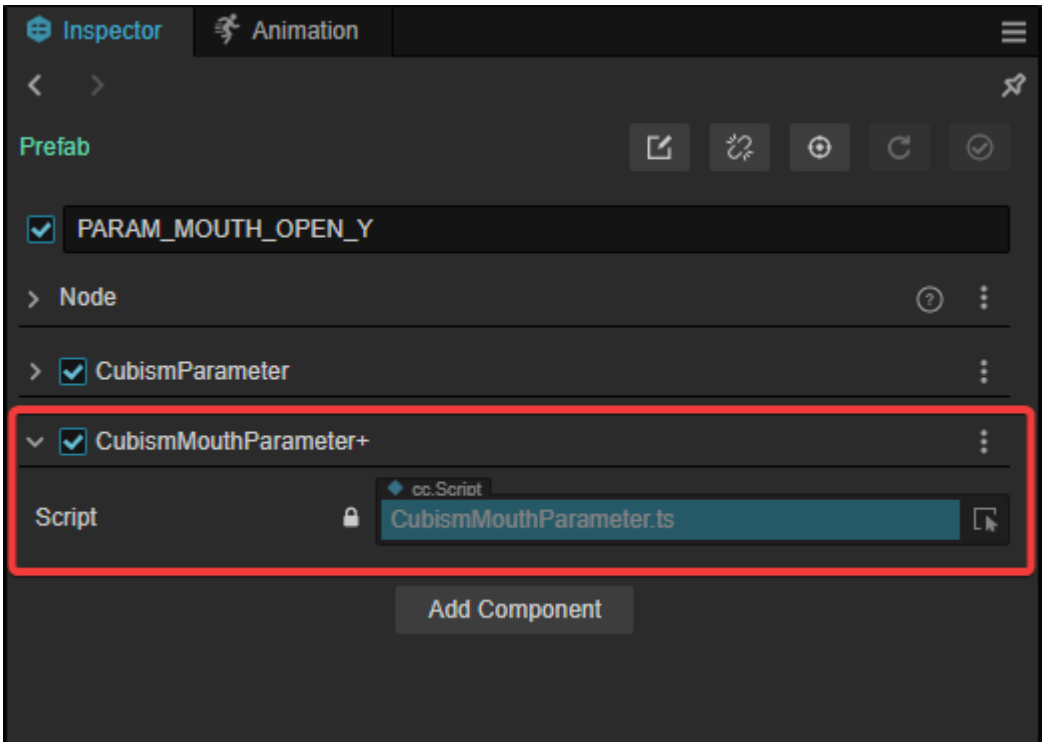


Specify parameters for lip-sync

Under [Model]/Parameters/ are Nodes that manage the parameters of the model. Also, the name set for this Node is the ID of the parameter. The CubismParameters attached to these Nodes are identical to those that can be obtained with CubismModel.Parameters().

From this Node, attach a component called CubismMouthParameter to the one with the ID to be treated as a lip sync.





With the settings up to the above, the mouth opening and closing operations can be performed. However, this alone still does not lip-sync when executed.

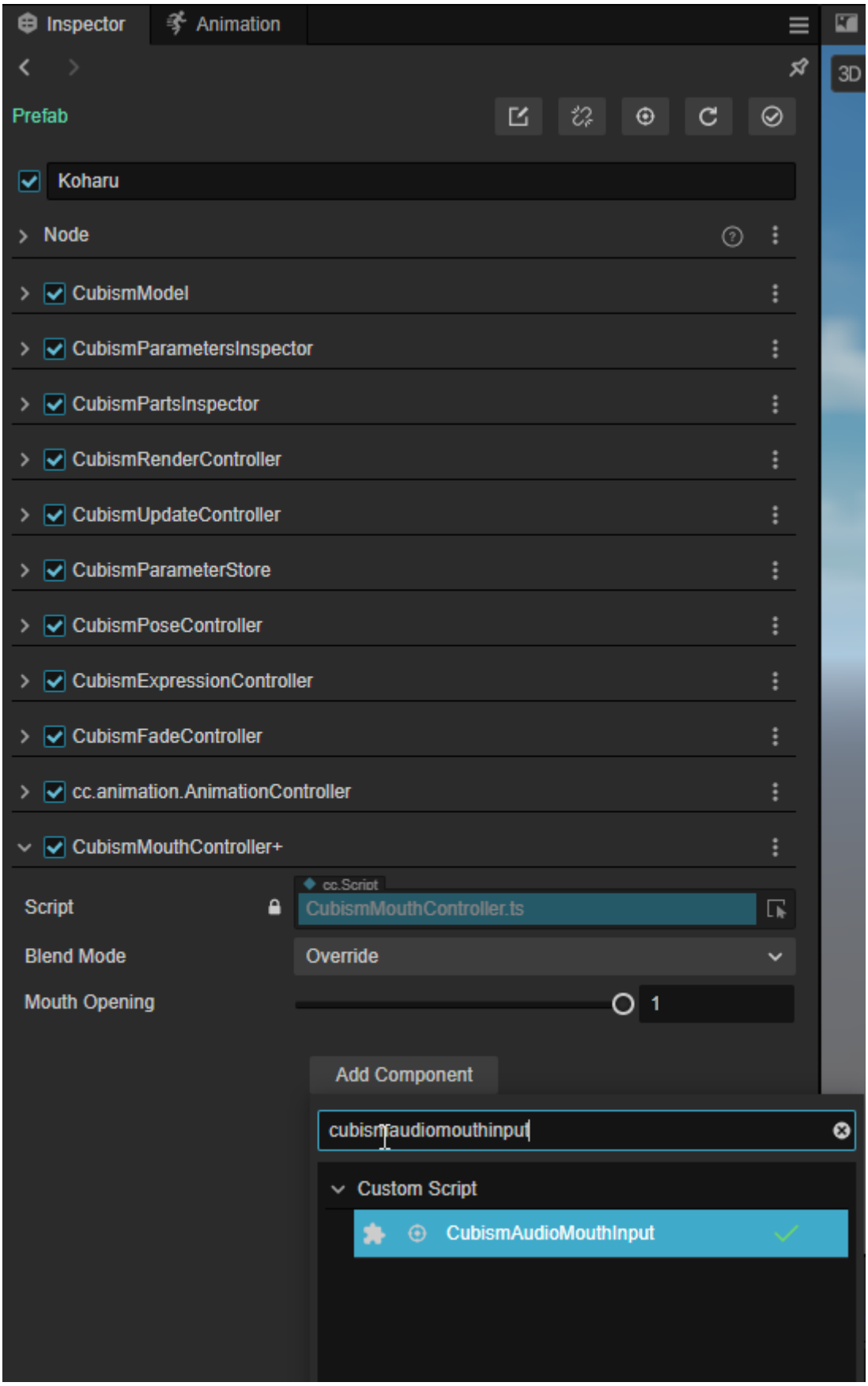
For automatic lip-sync, a component that manipulates the Mouth Opening value of the CubismMouthController must also be set.

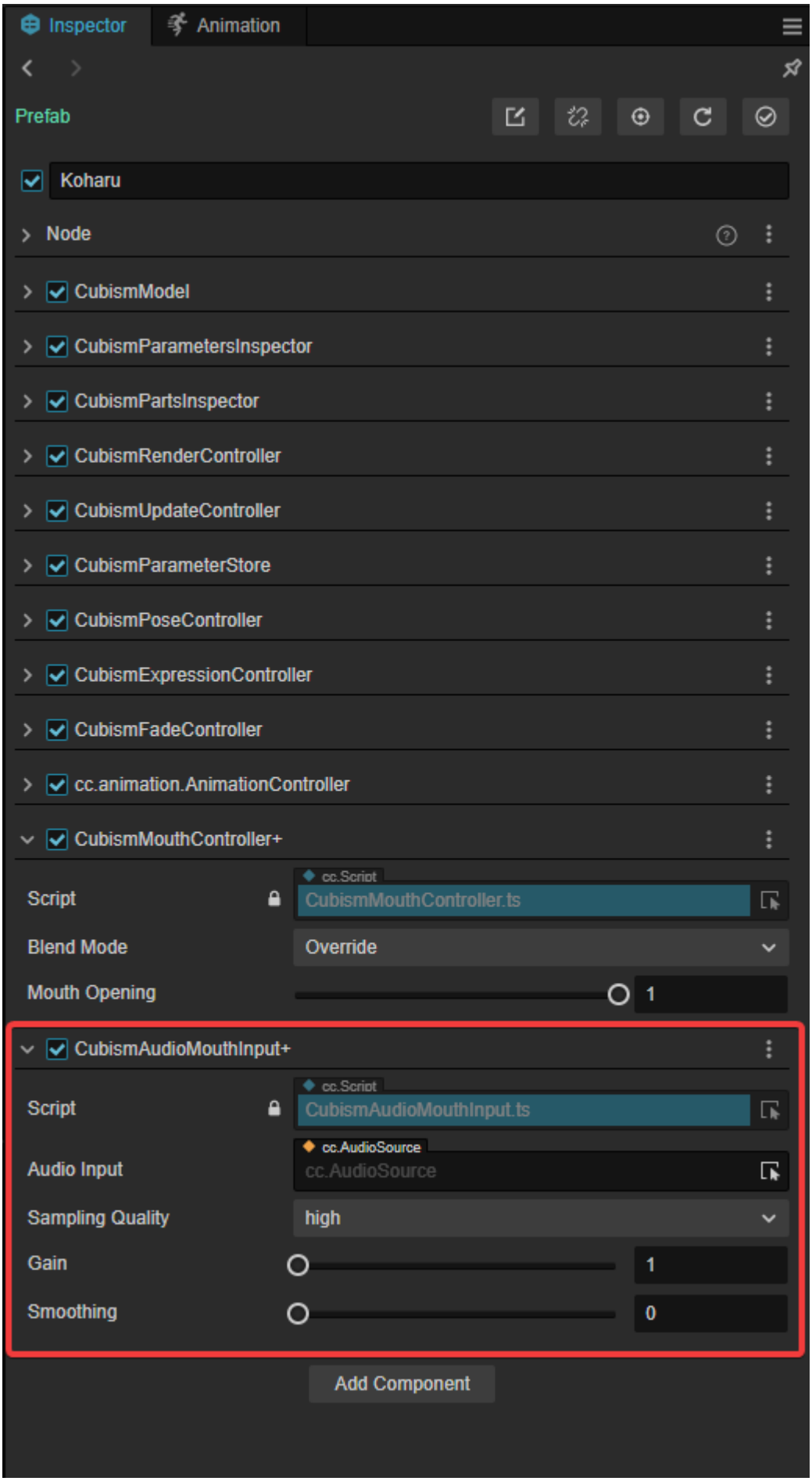
Set component to manipulate values of specified parameters

Attach a component for lip-sync input to the Node that is the root of the model.

MouthMovement contains components that manipulate the mouth open/close values from the AudioSource volume and Sin waves as input samples.

This time, we will attach a component called CubismAudioMouthInput so that we can set values from the AudioSource.


Inspector
Animation
3D
Prefab
Koharu
Node
CubismModel
CubismParametersInspector
CubismPartsInspector
CubismRenderController
CubismUpdateController
CubismParameterStore
CubismPoseController
CubismExpressionController
CubismFadeController
cc.animation.AnimationController
CubismMouthController+
cc.Script
Script
CubismMouthController.ts
Blend Mode
Override
Mouth Opening
1
Add Component
cubismaudiomouthinput
Custom Script
CubismAudioMouthInput



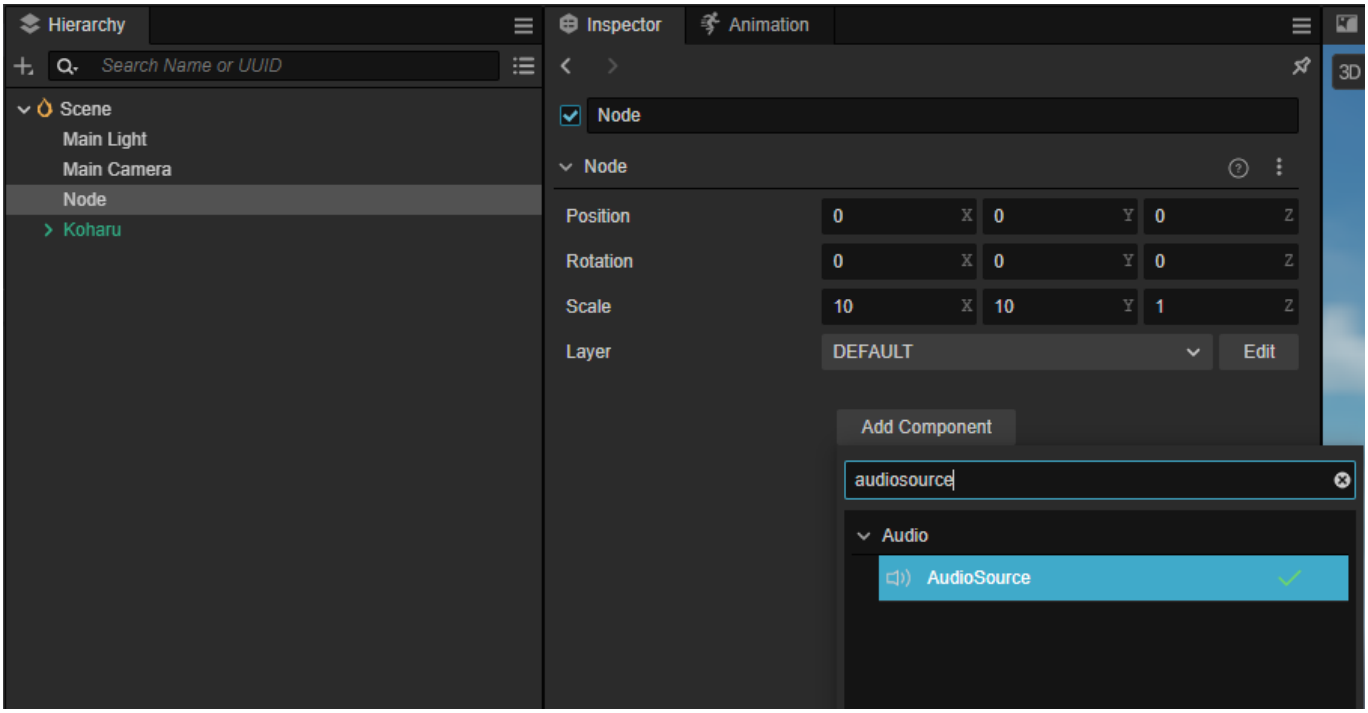
CubismAudioMouthInput has the following four settings.

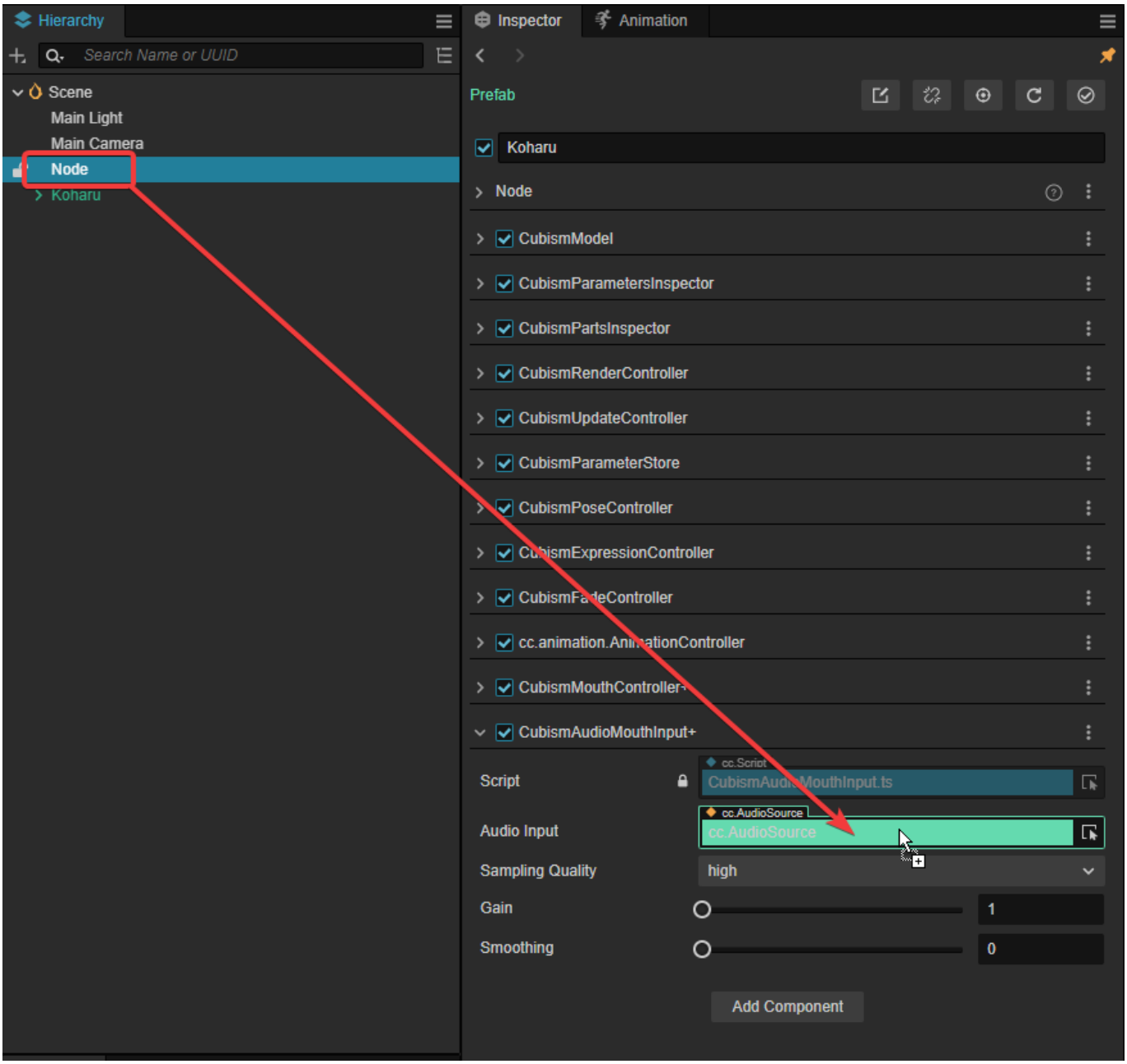
- Audio Input: Sets the AudioSource to be used for input. The volume of the AudioClip set to the AudioSource set here is used.
- Sampling Quality: Sets the accuracy of the sampling volume. The lower the order of the following settings, the more accurate, but also the more burdensome the calculation.
    - High
    - Very High
    - Maximum
- Gain: Sets how many times the sampled volume should be handled. It is equal to 1.
- Smoothing: Sets how much smoothing is applied to the opening and closing values calculated from the volume. The larger the value, the smoother, but also the higher the computational load.

In this case, the settings are as follows: Audio Input should be set to the Node to which the AudioSource is attached.

- Sampling Quality：High
- Gain：1
- Smoothing：5

Finally, to get the volume, attach an AudioSource to any Node and set it to the Audio Input of the CubismAudioMouthInput above.





This completes the lip-sync setup.

When the scene is executed in this state, the audio file set in AudioSource will be played and the model will lip-sync to the volume.

How to operate parameters cyclically

This section describes how to set up periodic movement of parameter values, like breathing or a pendulum.

Summary

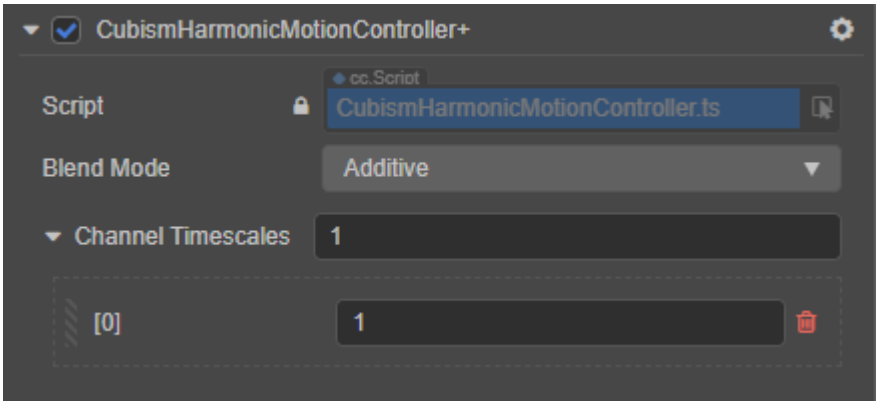
If you want to move any parameter cyclically, you can do so by using the **CubismHarmonicMotionController** .ts and **CubismHarmonicMotionParameter** .ts files.

Implementation involves two things

1. Set up to control parameters
2. Specify parameters to be moved

Configure settings to control parameters

First, attach a script to the model's foremost parent to control the motion.。 The name is **CubismHarmonicMotionController** .ts.



CubismHarmonicMotionController.ts には設定項目が 2 つあります。

There are two configuration items in the CustomHarmonicMotionController.cs.

- Blend Mode: Specifies how to calculate for the value currently set for the specified parameter.

    - Override: Overwrites the currently set value with a numerical value.
    - Additive: Adds a numerical value to the currently set value.
    - Mutiply: Multiply the currently set value by a numerical value.

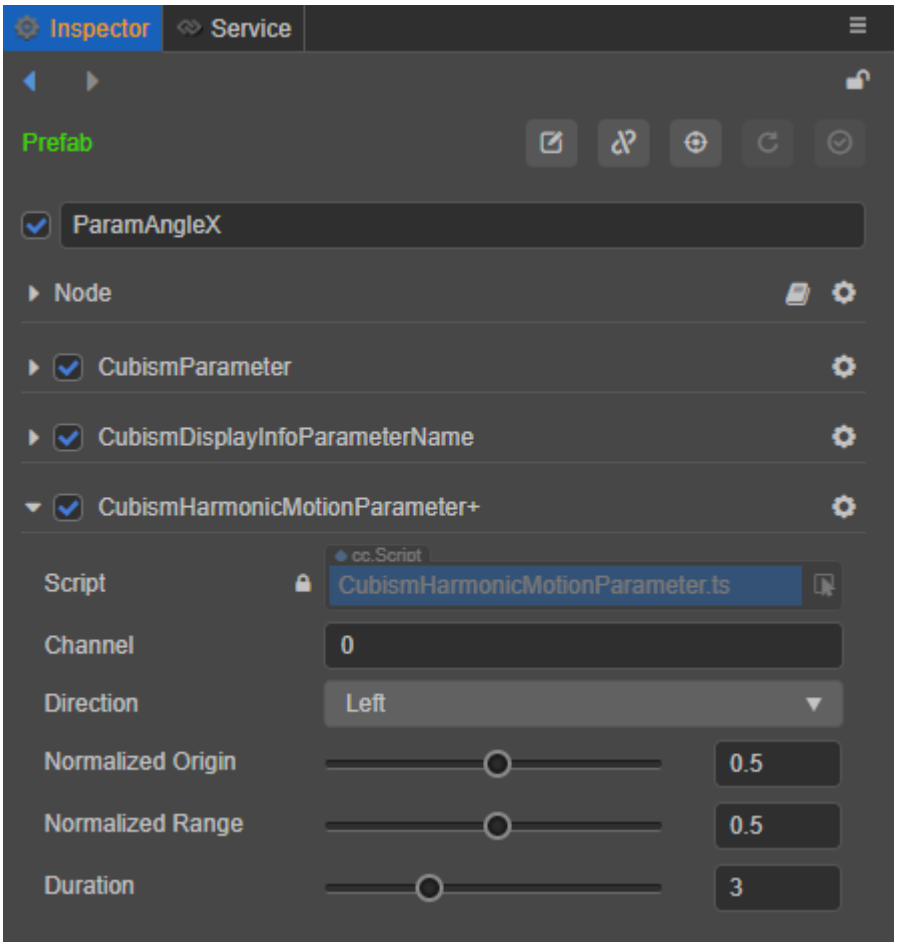
- Channel Timescales: You can optionally create multiple timescales and resize the timescales.

In this case, the Blend Mode settings are as follows.

- Blend Mode: Override

Specify parameters to be moved

Then, select the parameters you want to move and attach the file **CubismHarmonicMotionParameter** .ts. Here it attaches to angle X. Angle X will be in [root]/Parameters/ParamAngleX.



There are five configuration items in the CubismHarmonicMotionParameter.ts.

- Channel: Specify the Channel Timescale set in CustomHarmonicMotionController.cs.
- Direction: Specifies the width of the movement with respect to the center of the parameter.
    - Left: Only the left half moves from the center of the parameter.
    - Right: Only the right half moves from the center of the parameter.
    - Centric: The whole moves from the center of the parameter.
- Normalized Origin: Sets the position of the parameter to be centered.
- Normalized Range: Set the maximum distance to be moved from the center point based on the center determined by Normalized Origin.
- Duration: Adjusts the parameter cycle.

In this case, the settings are as follows:

- Channel: 0
- Direction: Centric
- Normalized Origin: 0.5
- Normalized Range: 0.5
- Duration: 3

With the above settings, the parameters can be moved periodically as shown in the video below.

Retrieve user data set in ArtMesh

This section explains how to get the “UserData” information set in the ArtMesh from the model.

Summary

[UserData] is a feature added in 3.1 that allows users to add arbitrary metadata to an ArtMesh. Depending on how the user data is handled, the ArtMesh can be used for a variety of things, such as designating that ArtMesh as a hit or applying special shading.

For information on how to set up user data in an ArtMesh, please click here.

To retrieve the user data set in the ArtMesh, follow the steps described below.

Examples of specific descriptions are:

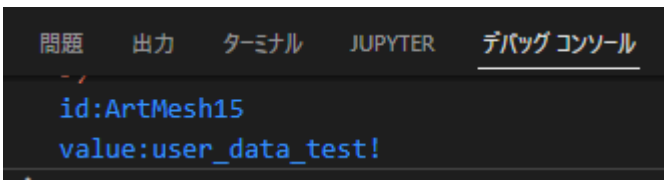
```
import { _decorator, Component, Node } from 'cc';
import CubismUserDataTag from
'../extensions/Live2DCubismSdkForCocosExtension/static/assets/Framework/UserData/C
ubismUserDataTag';
const { ccclass, property } = _decorator;

@ccclass('UserDataTest')
export class UserDataTest extends Component {
  protected start() {
    let userDatas = this.getComponentsInChildren(CubismUserDataTag);

    for (let i = 0; i < userDatas.length; ++i) {
      let data = userDatas[i];
      console.log("id:"+data.node.name + "\n"
      +"value:"+data.value);
    }
  }
}
```

Then, place the Prefab of the model with the user data set in the Hierarchy. Attach the UserDataTest component created above to the root of the placed Prefab.

Refer to the tutorial here for importing models.

This completes the setup.

When Scene is executed in this state, the Console window will output the ID of the art mesh to which the user data has been set and the string set in the user data.



Get events set in motion3.json

This section describes how to retrieve information about events set in motion3.json from events issued from an automatically generated AnimationClip.

Summary

[UserData] is a function added in 3.1 that allows users to issue events at any point in the motion. In Cocos Creator, events are issued from the AnimationClip that is converted from motion3.json. Depending on how events are handled, they can be used for various things, such as playing sound in the middle of a motion or changing the display state of a part.

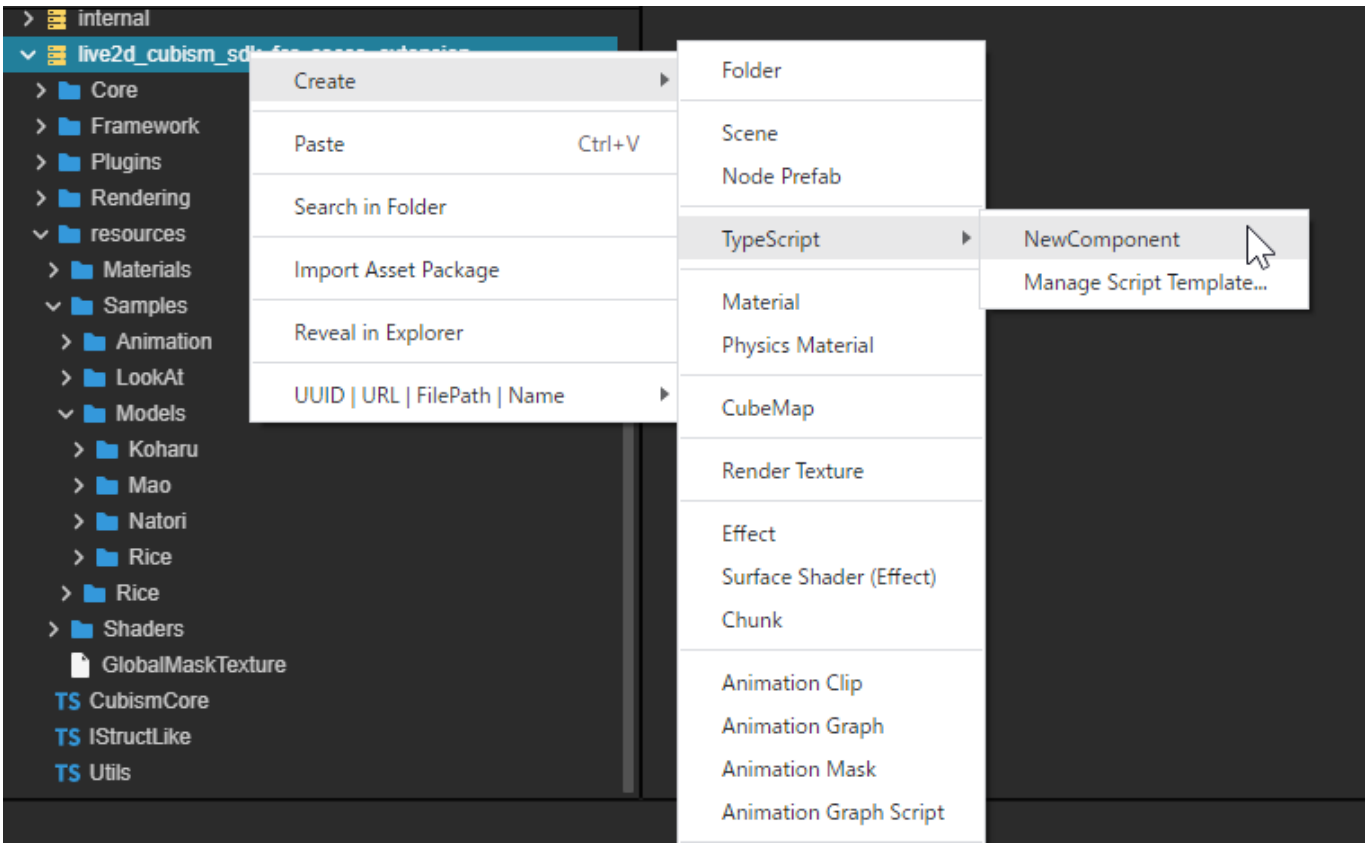
For more information on how to set up events in motion3.json, please click here.

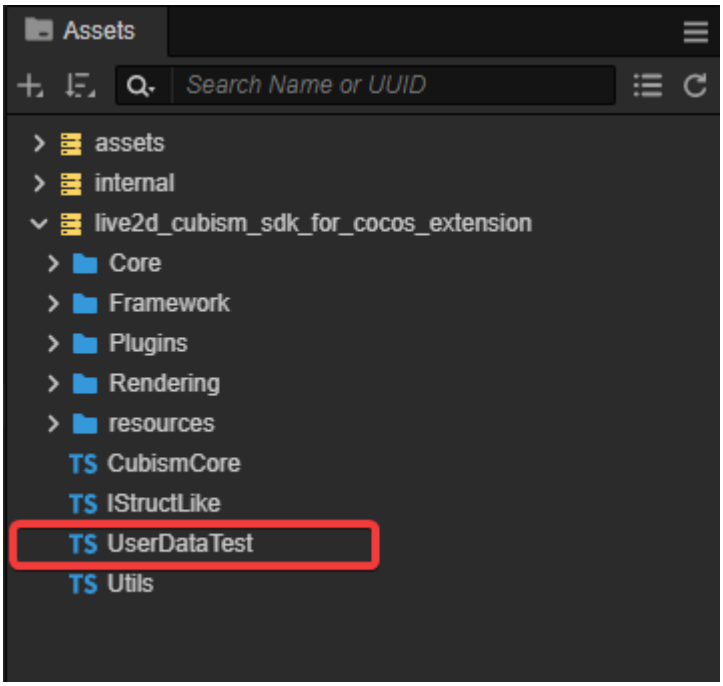
The following procedure is used to obtain events issued by motion.

1. Creating a component to receive events
2. Added description to CubismMotion3Json
3. Place model and motion in Scene

Creating a component to receive events

Right-click on the Assets window and click Create - TypeScript - New Component to create a TypeScript script. Here the name will be UserDataTest.





Rewrite the contents of the UserDataTest you created as follows:

```
import { _decorator, Component, Animation } from 'cc';
const { ccclass, property } = _decorator;

@ccclass('UserDataTest')
export class UserDataTest extends Component {

  UserDataEventListener(value: String) {

    const anim = this.getComponents(Animation)[0];
    const currentState = anim.getState(anim.defaultClip!!.name)

    console.log("Time: " + currentState.current + "\n" + "Value: " + value);
  }

}
```

Add a description to CubismMotion3Json

Open the CubismMotion3Json class and add the following description to the part of toAnimationClipB() where the AnimationEvent is created.

```
const frame = this.userData[i].time;
const functionName = `UserDataEventListener`
const params = new Array<string>();
params.push(this.userData[i].value);
animationClip.events.push({ frame: frame, func: functionName, params: params });
```

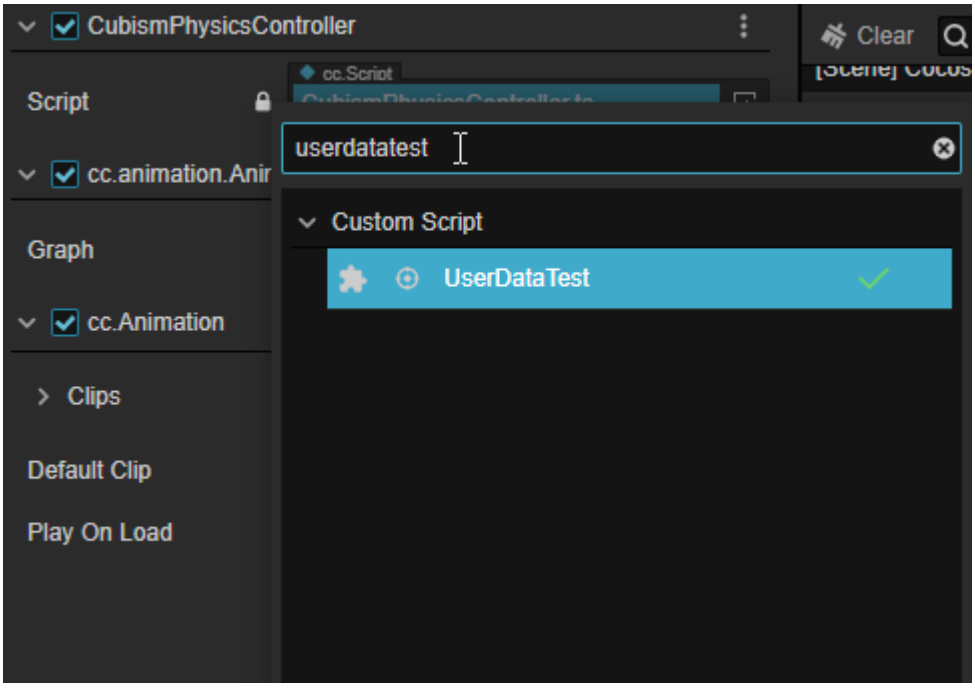
In func, set the name of the method that will receive events from the UserDataTest class created in the previous section.

Place model and motion in Scene

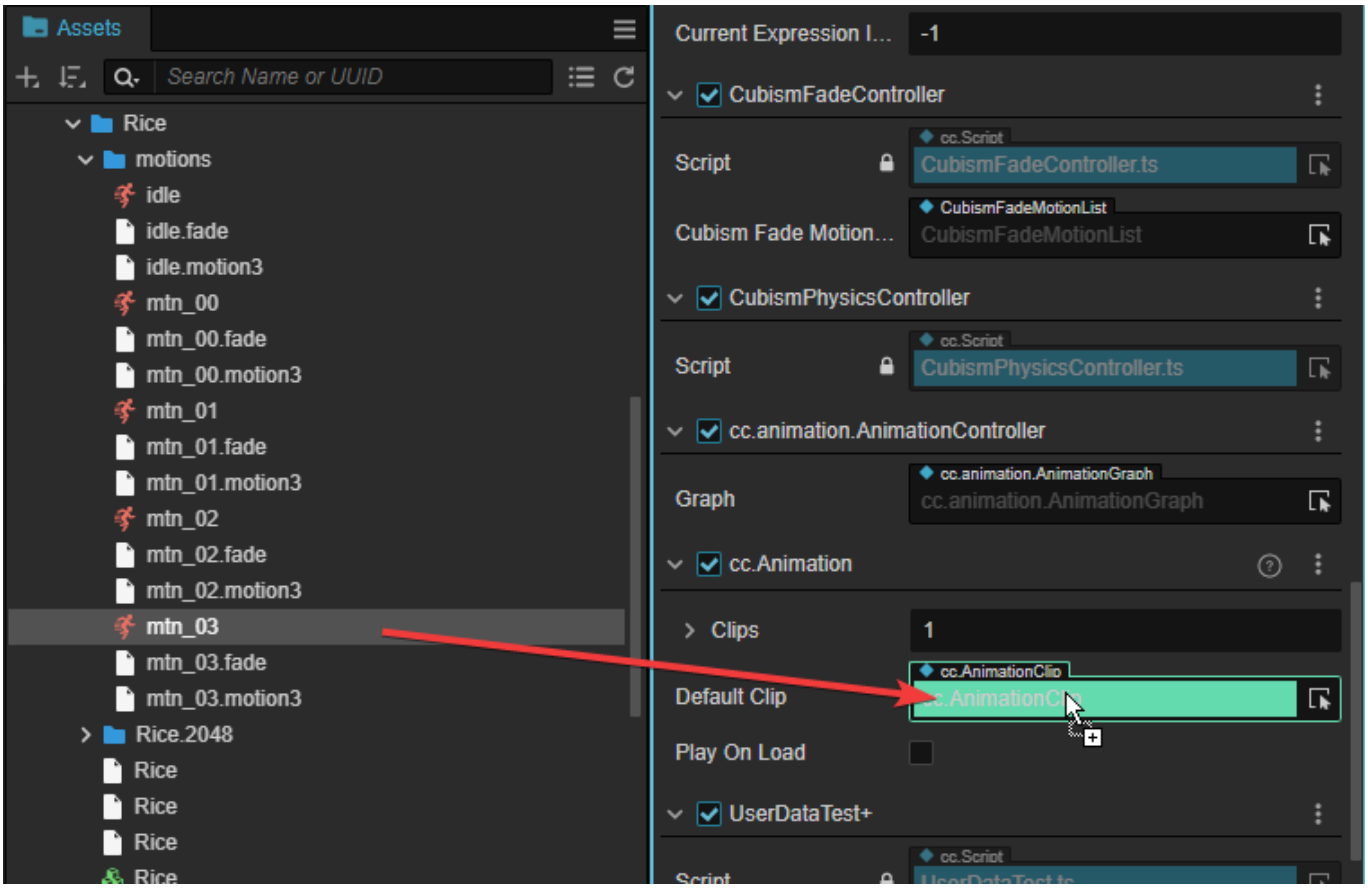
Import or re-import motion3.json with events set.

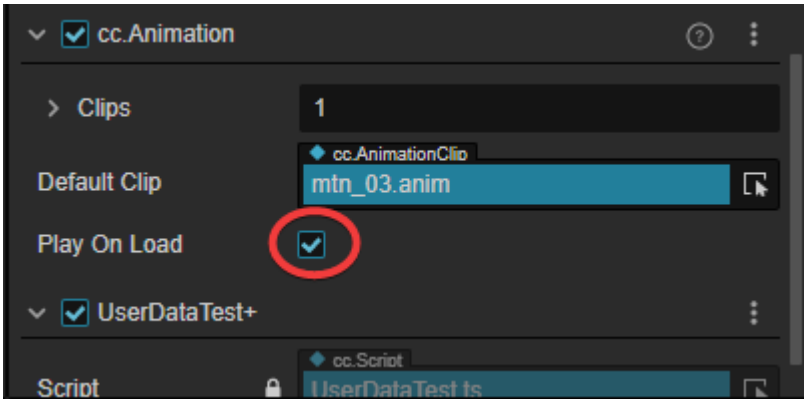
For information on importing and re-importing, please see the corresponding tutorial.

Place the model's Prefab in the Hierarchy window and attach the UserDataTest component to the root of the Prefab.



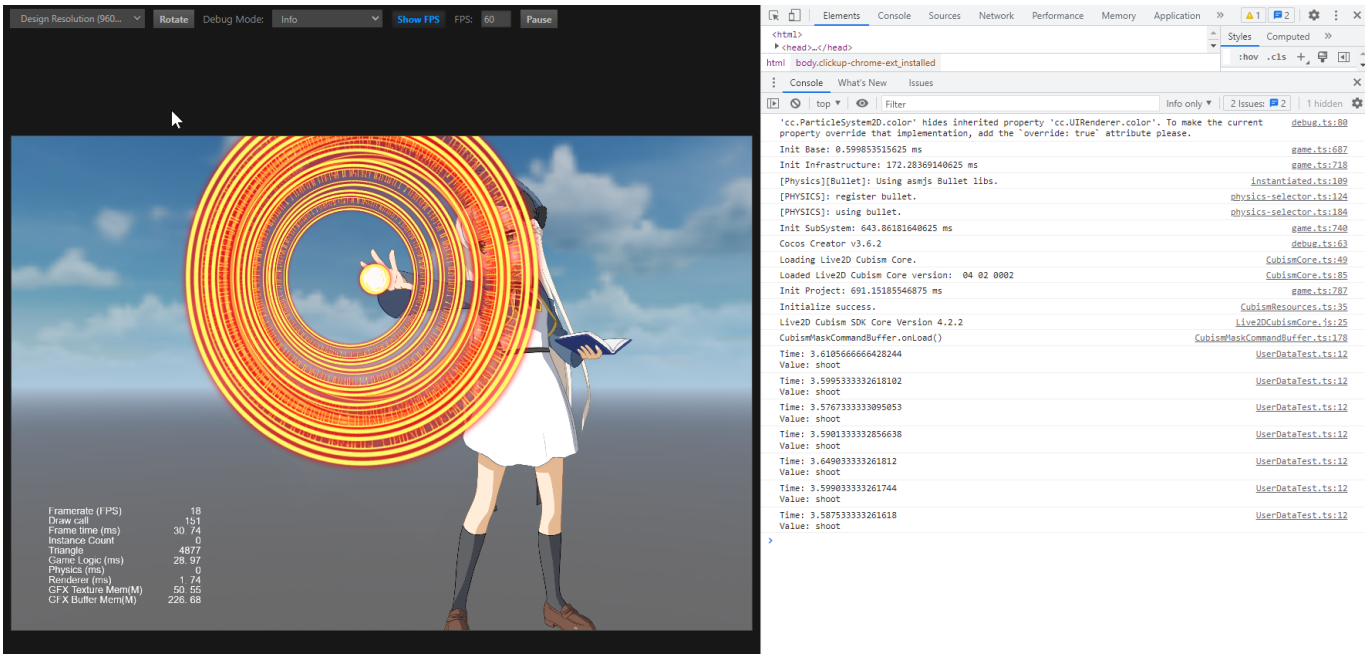
Drag and drop the AnimationClip generated from motion3.json into the Default Clip field in the Animation component to the Prefab placed in the Hierarchy.





This completes the setup.

When a Scene is executed in this state, an event is issued at a specific timing from the playing AnimationClip, and the string set in the event is output to the Console window or browser console.


Design Resolution (960...
Rotate
Debug Mode:
Info
Show FPS
FPS:
60
Pause
Framerate (FPS) 18
Draw call 151
Frame time (ms) 30.74
Instance Count 0
Triangle 4877
Game Logic (ms) 28.97
Physics (ms) 0
Renderer (ms) 1.74
GFX Texture Mem(M) 50.55
GFX Buffer Mem(M) 226.68
Elements
Console
Sources
Network
Performance
Memory
Application
Styles
Computed
Console
What's New
Issues
Filter
Info only
2 Issues: 2
1 hidden
'cc.ParticleSystem2D.color' hides inherited property 'cc.UIRenderer.color'. To make the current property override that implementation, add the `override: true` attribute please.
Init Base: 0.599853515625 ms
Init Infrastructure: 172.28369140625 ms
[Physics][Bullet]: Using asmjs Bullet libs.
[PHYSICS]: register bullet.
[PHYSICS]: using bullet.
Init SubSystem: 643.86181640625 ms
Cocos Creator v3.6.2
Loading Live2D Cubism Core.
Loaded Live2D Cubism Core version: 04 02 0002
Init Project: 691.15185546875 ms
Initialize success.
Live2D Cubism SDK Core Version 4.2.2
CubismMaskCommandBuffer.onLoad()
Time: 3.6105666666428244
Value: shoot
Time: 3.5995333332618102
Value: shoot
Time: 3.5767333333095053
Value: shoot
Time: 3.5901333332856638
Value: shoot
Time: 3.649033333261812
Value: shoot
Time: 3.599033333261744
Value: shoot
Time: 3.587533333261618
Value: shoot

Use of .cdi3.json

Summary

This section explains how to customize the display names of parameters, parameter groups, and parts using .cdi3.json.

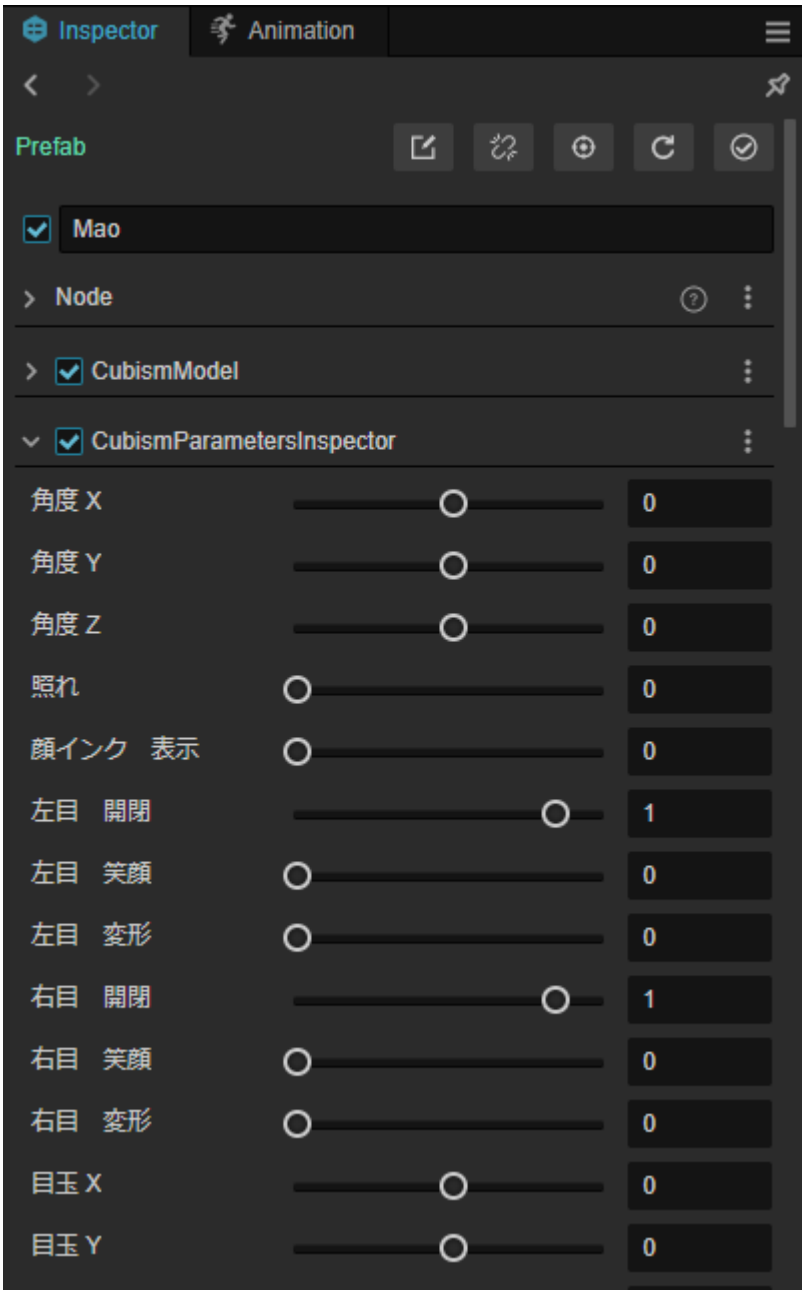
It is assumed that .cdi3.json has been written out for the model data to be used.

For more information about .cdi3.json and its implementation in SDK for Native and SDK for Web, please see here.

As a preliminary preparation, refer to [Import SDK-Placing Models], import the model data with .cdi3.json exported and place the prefab, then select the root object of the prefab.。

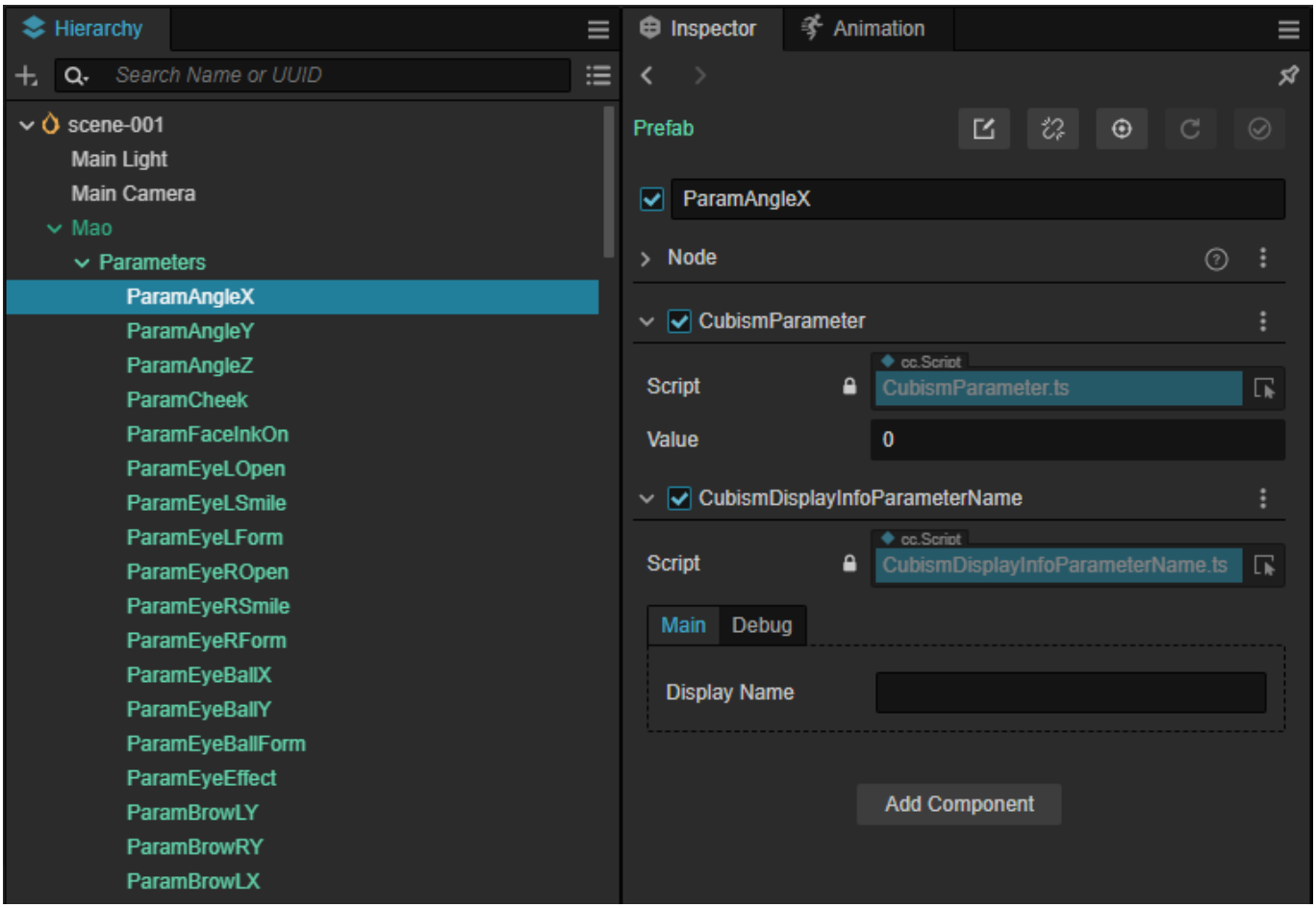
Use on inspector

When model data containing .cdi3.json is imported, the names of the parameters and parts listed in .cdi3.json can be displayed on the inspector of the model prefab.

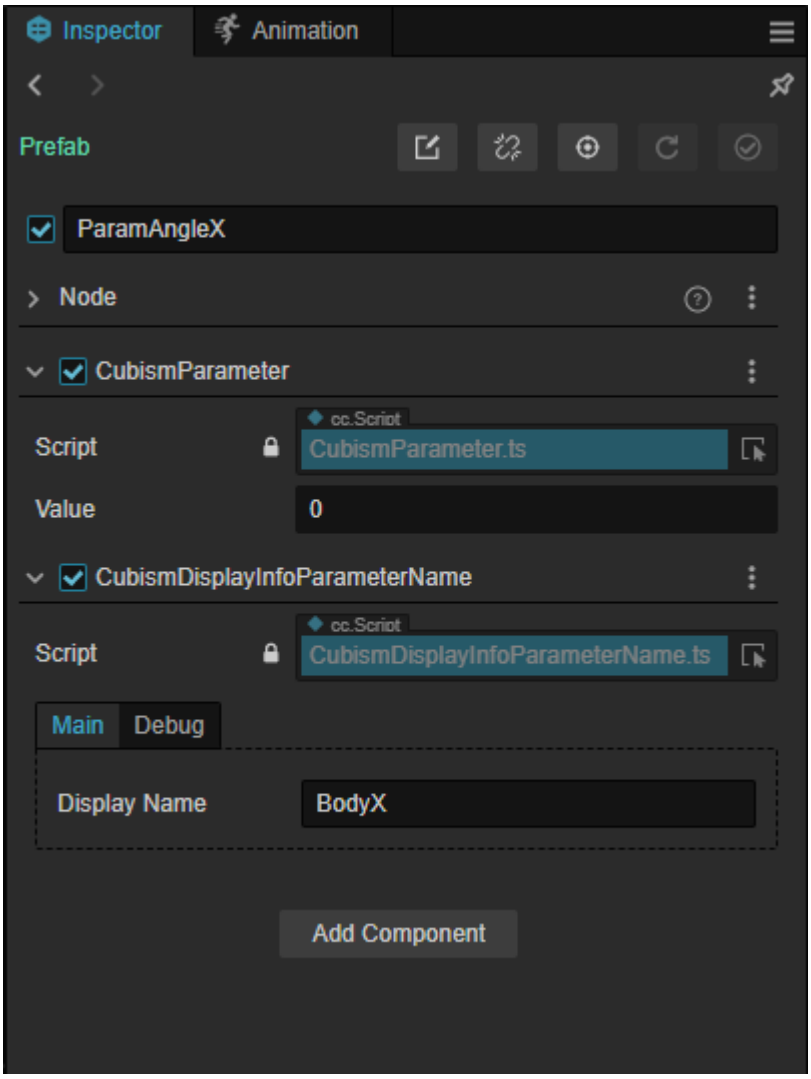


Users can also assign any name they wish. To set a name, enter the name in the Display Name field under [Cubism Display Info Parameter Name] or [Cubism Display Info Part Name]. If Display Name is empty, the name listed in .cdi3.json is used.

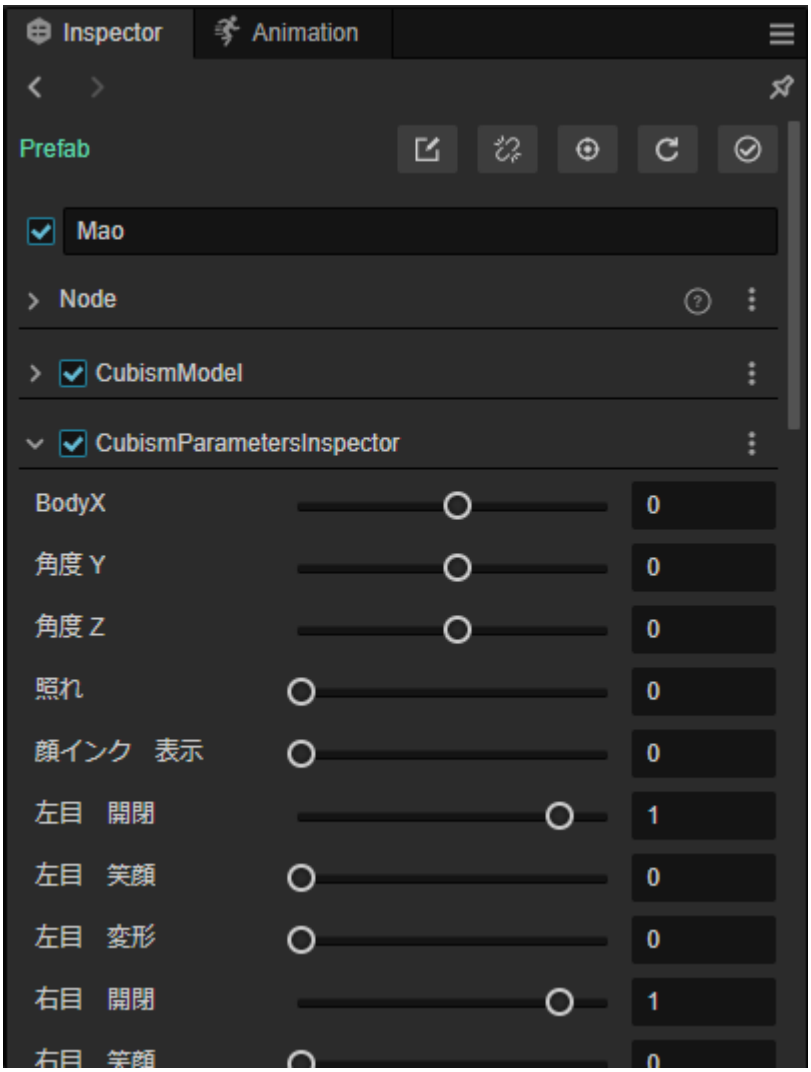
Display Name is not yet set:



With Display Name set to:



Notation of parameters when setting the Display Name:


Inspector
Animation
Prefab
Mao
Node
CubismModel
CubismParametersInspector
BodyX
0
角度 Y
0
角度 Z
0
照れ
0
顔インク　表示
0
左目　開閉
1
左目　笑顔
0
左目　変形
0
右目　開閉
1

Multiply Color/Screen Color

This section explains how to blend and draw colors on a model using multiply and screen colors.

Models created before Cubism Editor 4.2, for example, can be used without any additional coding even if the model does not have Multiply/Screen Color set. For detailed specifications and usage of the SDK for Cocos Creator, please refer to [Multiply Colors/Screen Colors] in the SDK Manual.

As a preliminary preparation, please refer to [Import SDK] to import model data and place prefabrication.

By default, it is set to always refer to the Multiply Color and Screen Color previously set for the model. If the Multiply Color and Screen Color are not set for the model, the following values are used.

- In Multiply Color (1.0, 1.0, 1.0, 1.0)
- In Screen Color (0.0, 0.0, 0.0, 1.0)

Use on inspector

To enable manipulation of Multiply Color and Screen Color from the SDK side, enable the following flags.

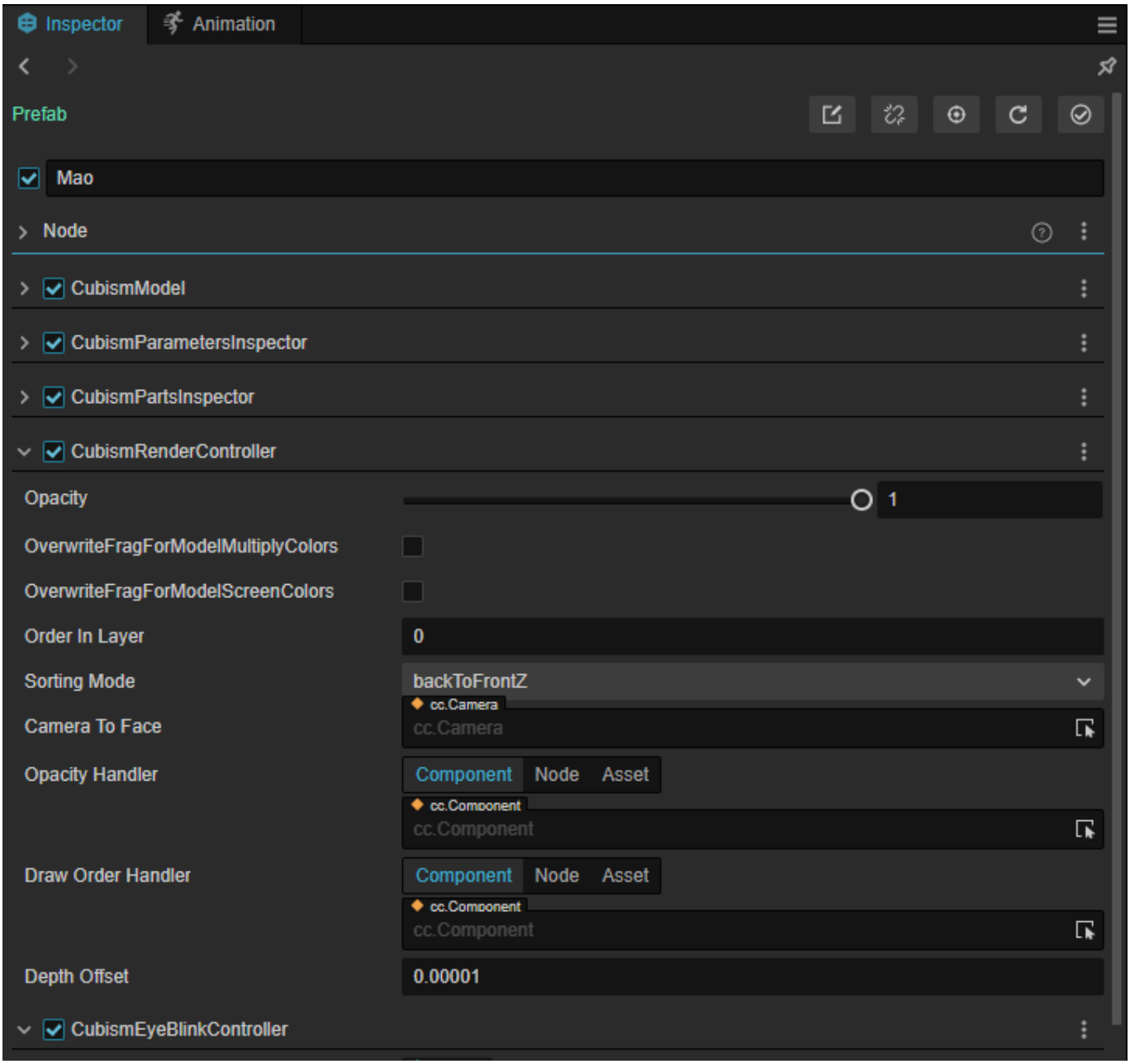
Multiply Color:

overwriteFlagForModelMultiplyColors or overwriteFlagForMultiplyColors

Screen Color:

overwriteFlagForModelScreenColors or overwriteFlagForScreenColors

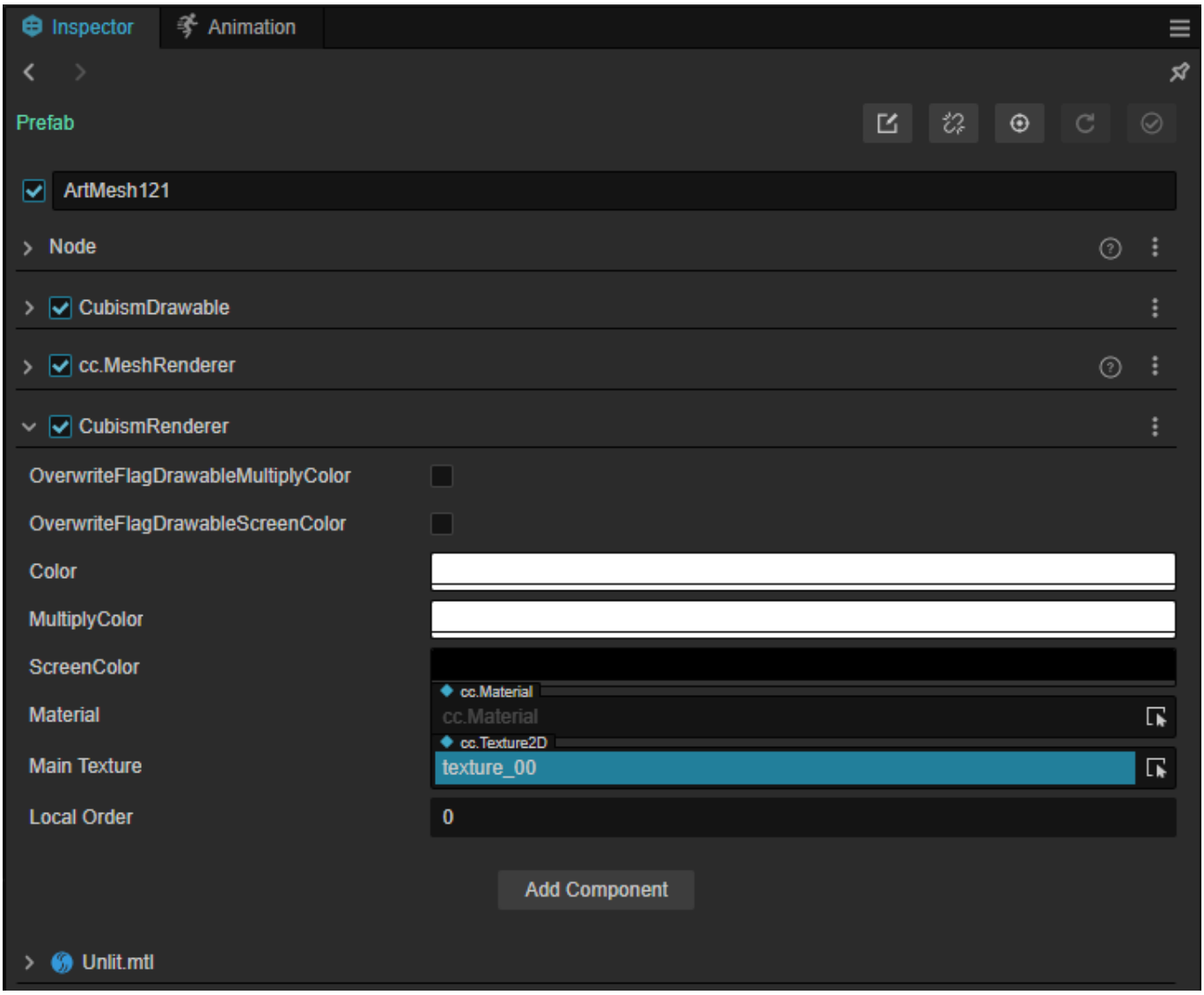
OverwriteFlagForModelMultiplyColors and OverwriteFlagForModelScreenColors are flags that determine whether the SDK can manipulate Multiply Color and Screen Color for all Drawables. These flags can also be manipulated in the Inspector of the [CubismRenderController], which is attached to the model prefab root object in Cocos Creator.



OverwriteFlagForMultiplyColors and OverwriteFlagForScreenColors are flags that determine whether each individual Drawable can manipulate Multiply Color and Screen Color from the SDK side. It can also be manipulated in the inspector of the [CubismRenderer] attached to each Drawable object in the model.

If the Multiply Color and Screen Color flags that [CubismRenderController] has as described above are enabled, they will take precedence.

Multiply Color and Screen Color settings can be controlled not only from scripts, but also from the [CubismRenderer] inspector.



Use on scripts

The following code is useful for use in applications and other cases where control is needed on scripts.

The following code is designed to change the Screen Color of all Drawable objects at the same time in a certain period of time.

You can create a TypeScript script that rewrites the contents as follows and attach it to the root object of the model prefab.

```
import { _decorator, Component, math } from 'cc';
import CubismRenderController from './Rendering/CubismRenderController';
const { ccclass, property } = _decorator;

@ccclass('BlendColorChange')
export class BlendColorChange extends Component {

  private renderController: CubismRenderController | null = null;
  private _colorValues: number[] = new Array<number>();
  private _time: number = 0;

  protected start() {
```

```
    this._colorValues = new Array<number>(3);
    this._time = 0;
    this.renderController = this.getComponent(CubismRenderController);
    this.renderController!.overwriteFlagForModelScreenColors = true;
  }

  protected update(deltaTime: number) {
    if (this._time < 1.0) {
      this._time += deltaTime;
      return;
    }

    for (let i = 0; i < this._colorValues.length; i++)
    {
      this._colorValues[i] = Math.random();
    }

    const color = new math.Color(
      this._colorValues[0] * 255,
      this._colorValues[1] * 255,
      this._colorValues[2] * 255,
      1.0);

    for (let i = 0; i < this.renderController!.renderers!.length; i++)
    {
      this.renderController!.renderers![i].screenColor = color;
    }

    this._time = 0.0;
  }
}
```

Receive notification of Multiply Color and Screen Color updates from the model side

If a model parameter is associated with a change in Multiply Color/Screen Color, the model may change the Multiply Color/Screen Color when the model is animated, rather than from the SDK side.

The property IsBlendColorDirty is implemented in [CubismDynamicDrawableData] to indicate that the Multiply Color and Screen Color have been changed.

This property is true when either the Multiply Color or Screen Color is changed on the model side, and does not determine whether the Multiply Color or Screen Color is changed.

For details, please refer to [Multiply Color/Screen Color] in the SDK Manual.

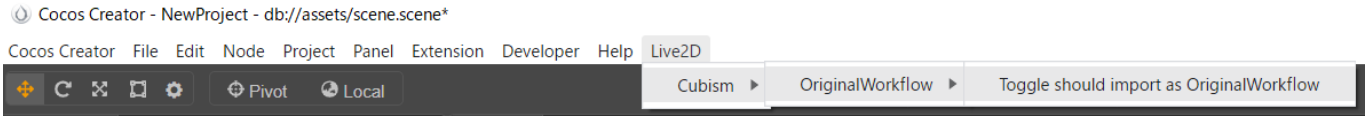
How to set up UpdateController

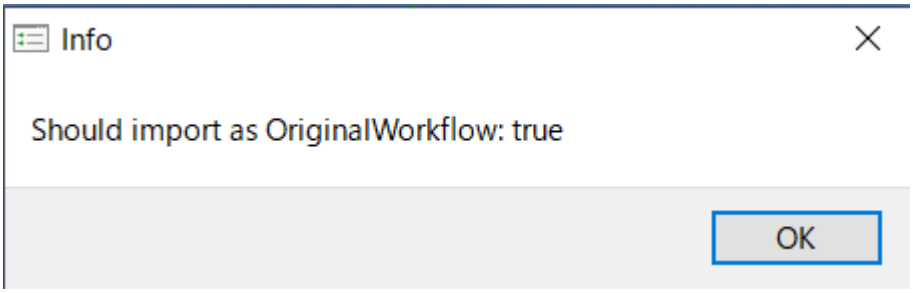
This page describes the procedure for controlling the order of execution of each component of Cubism in Cocos Creator. The following assumes that the SDK will be added to a project that has Import SDK.

Summary

CubismUpdateController is used to control the order of execution of each component.

Click [Live2D/Cubism/OrignalWorkflow/ **Toggle Should Import As Original Workflow**] in the Cocos Creator menu. If the model is imported with **true** as shown in the following image, the generated Prefab will be set to this component.





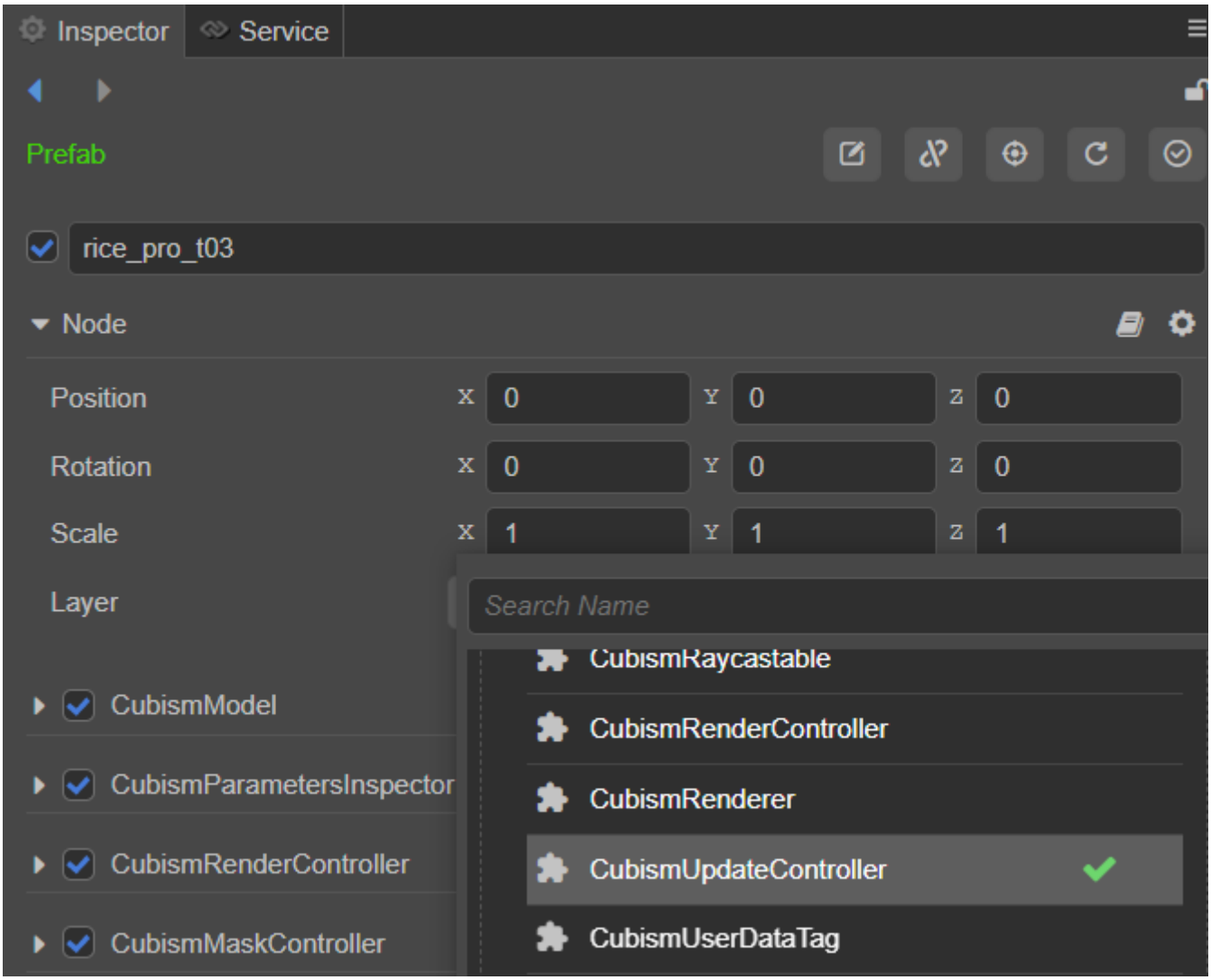
If you wish to have the update order of each Cubism component of a model that has not been configured above controlled, you can do so by following the procedure described in this article.

To set **UpdateController** for a Cubism model, follow the steps below.

1. Attach CubismUpdateController

1. Attach CubismUpdateController

Attach a **CubismUpdateController** to the Node that is the root of the model to control the update order of components.


Inspector
Service
Prefab
rice_pro_t03
Node
Position
Rotation
Scale
Layer
Search Name
CubismRaycastable
CubismRenderController
CubismRenderer
CubismUpdateController
CubismUserDataTag
CubismModel
CubismParametersInspector
CubismRenderController
CubismMaskController

Use the expression function

This page describes the procedure for regenerating facial expressions on Cubism models using Expression. The following assumes that the [Import Models] [How to set up UpdateController] [Save/Restore the values to be manipulated] before being added to the project.

Summary

To playback facial expressions by .exp3.json in Live2D Cubism SDK for Cocos Creator, use the "Expression" component.

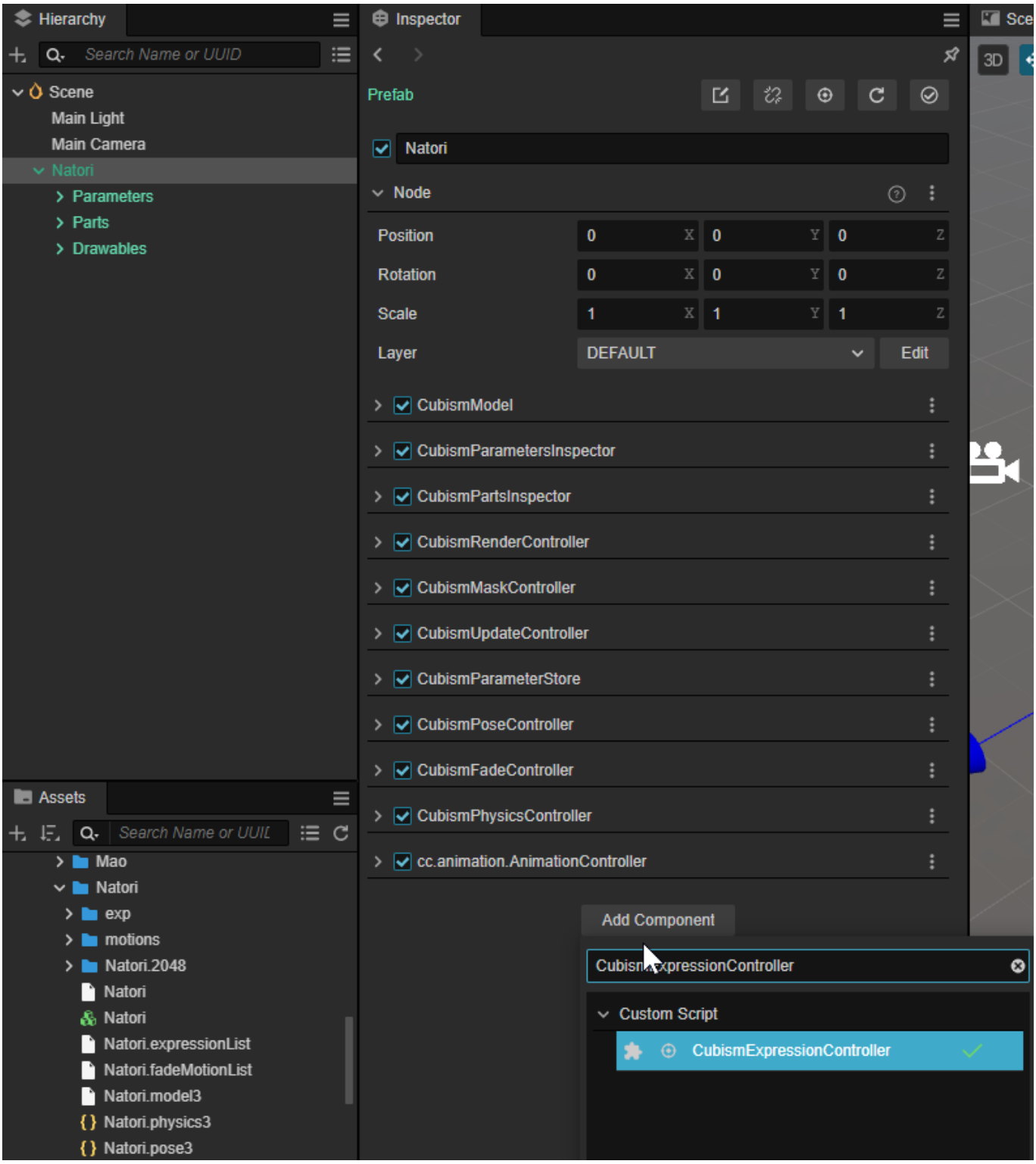
If the model was imported with the "Live2D/Cubism/OrignalWorkflow/Should Import As Original Workflow" checkbox in the Cocos Creator editor menu, the generated Prefab will be set to Expression will be set.

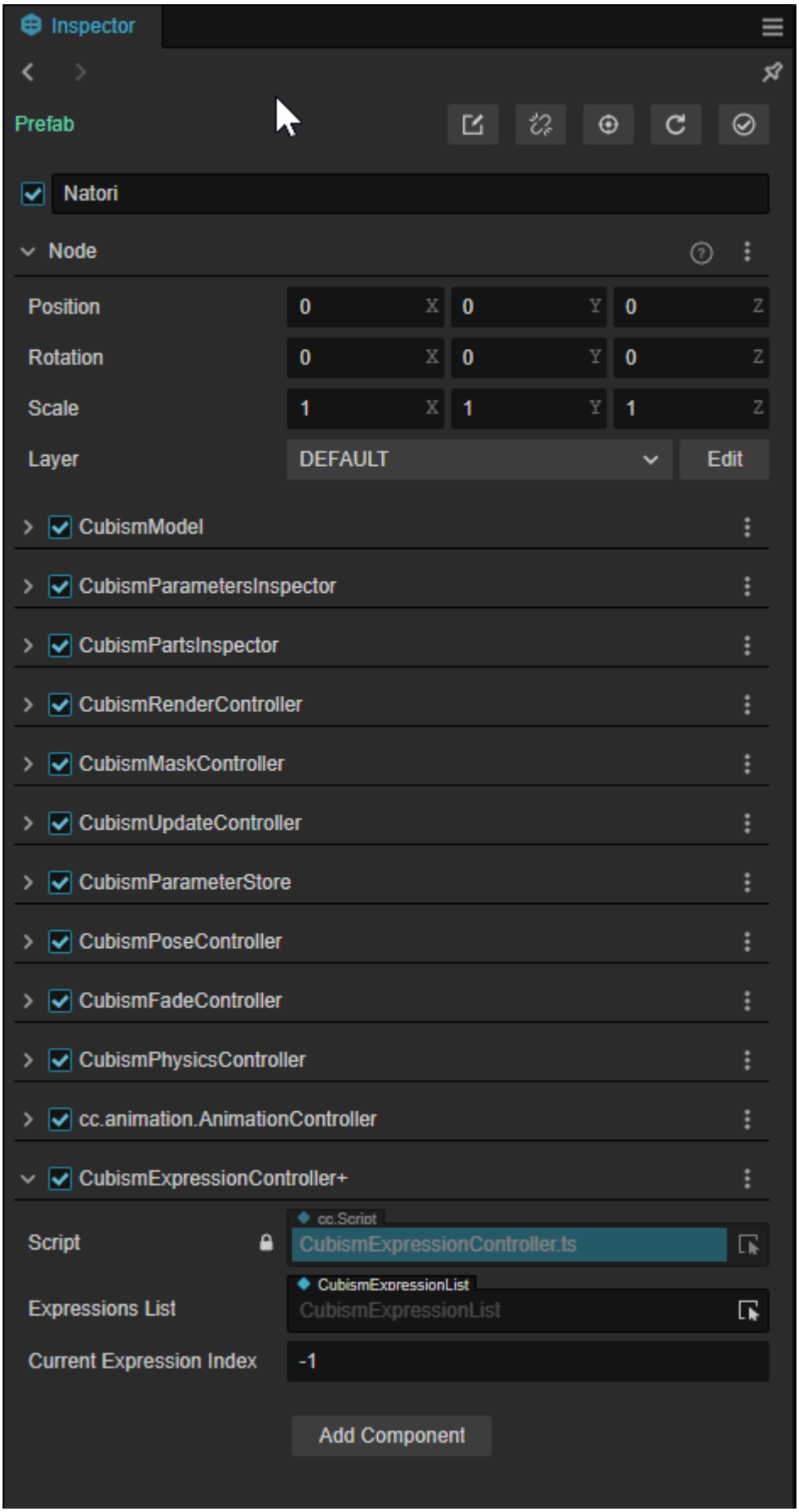
To set Expression to a Prefab that is not generated by the OW method, perform the following three steps.

1. Attach CubismExpressionController
2. Set “[model name].expressionList”
3. Set the facial expression to be played

1. Attach CubismExpressionController

Attach “CubismExpressionController” to the Node that will be the root of the model.


Hierarchy
Search Name or UUID
Scene
Main Light
Main Camera
Natori
Parameters
Parts
Drawables
Inspector
Prefab
Natori
Node
Position
Rotation
Scale
Layer
DEFAULT
Edit
CubismModel
CubismParametersInspector
CubismPartsInspector
CubismRenderController
CubismMaskController
CubismUpdateController
CubismParameterStore
CubismPoseController
CubismFadeController
CubismPhysicsController
cc.animation.AnimationController
Add Component
Custom Script
CubismExpressionController
Assets
Mao
Natori
exp
motions
Natori.2048
Natori
Natori
Natori.expressionList
Natori.fadeMotionList
Natori.model3
Natori.physics3
Natori.pose3

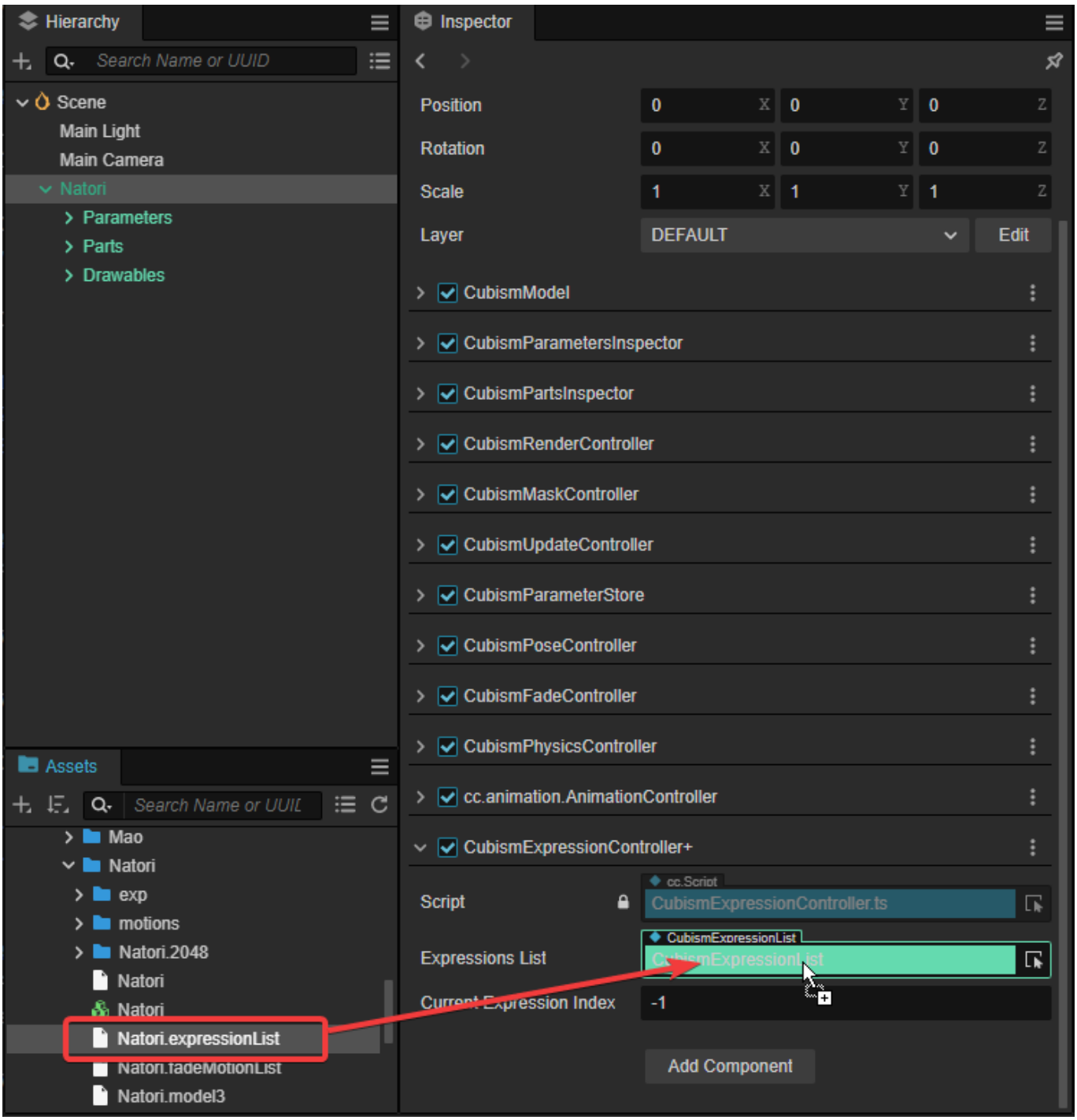


CubismExpressionController has two configuration items.

- Expression List: Set “model name.expressionList” * Details are explained in step 2
- Current Expression Index: Set the index of the expression to be played back * Details are explained in step 3 of the procedure

2. Set “modelName.expressionList”

Select the model and drag and drop “[model name].expressionList” from the Inspector view into the “Expression List” of the CubismExpressionController.



- “[model name].expressionList” is a list of “.exp3” assets.
- The “.exp” asset is the asset where Expression data is stored. The “.exp3.json” is automatically generated when imported.

3. Set the facial expression to be played

Set the index of the expression to be played to “Current Expression Index” in CubismExpressionController. This time, set to 0.

This completes the setup for playback of facial expressions using the exp3 asset generated from exp3.json.

When Scene is executed in this state, the expression number 0 in “[model name].expressionList” will be played. By resetting the “Current Expression Index,” you can switch the expression to be played back.

- If an index other than “[model name].expressionList” is set, the default expression will be played.

Use the Pose function

This page describes the procedure for having Cubism models control the display state of parts. The following assumes that the model will be added to the project where［Import SDK～Placing Models］ was done.

For information on how to set up Pose, please click here を、and for the SDK's Pose specifications, please see Manual - Pose.

Summary

If the model was imported with "Live2D/Cubism/OrignalWorkflow/Should Import As Original Workflow" in the Cocos Creator editor menu set to Enable, Pose will be set on the model.

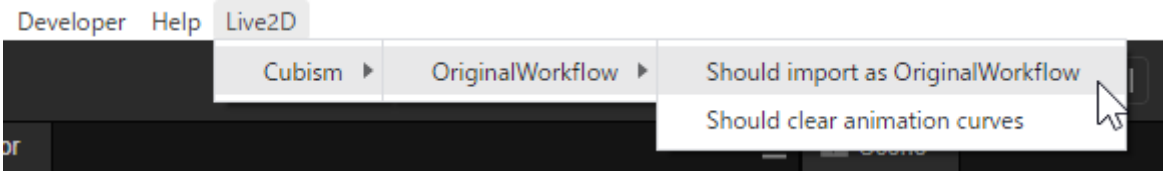
If you wish to apply the Pose function to a model that has not been configured above, you can do so by following the procedure described in this article.

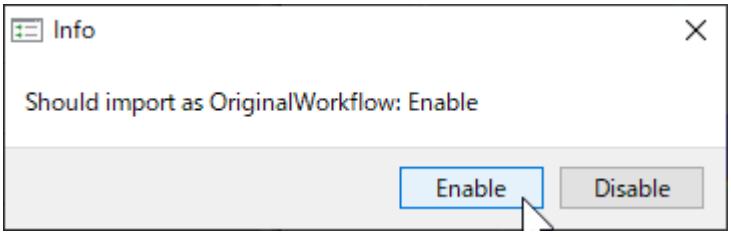
1. Clear AnimationClip curves
2. Re-import models in OW mode

- When a model is reimported in OW mode, components for OW other than Pose settings such as “CubismUpdateController”, “CubismParameterStore” and “CubismExpressionController” will be added at the same time.

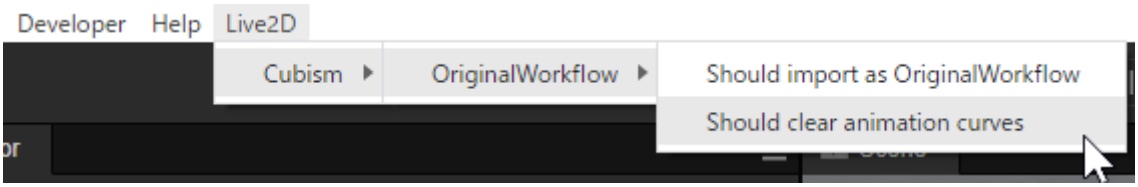
Clear AnimationClip curves

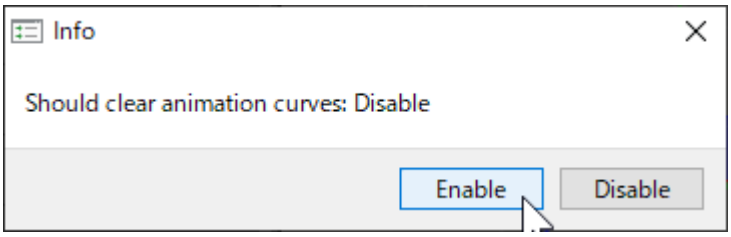
1. Select "Live2D/Cubism/OrignalWorkflow/Should Import As Original Workflow" from the Cocos Creator editor menu and Enable in the dialog that appears.



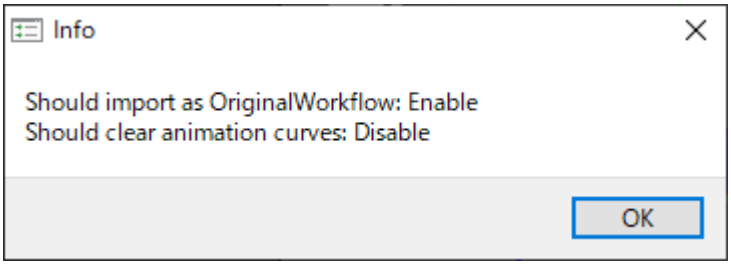


2. Select "Live2D/Cubism/OrignalWorkflow/Should Clear Animation Curves" from the Cocos Creator editor menu and set to Enable in the dialog that appears.

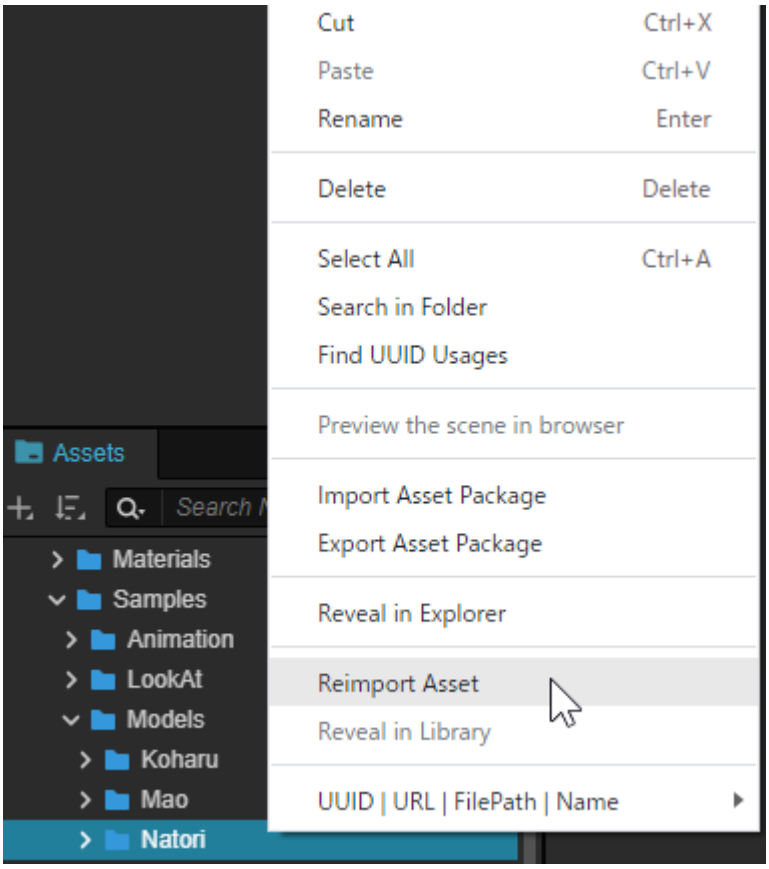




If the display looks like the following, it is OK.

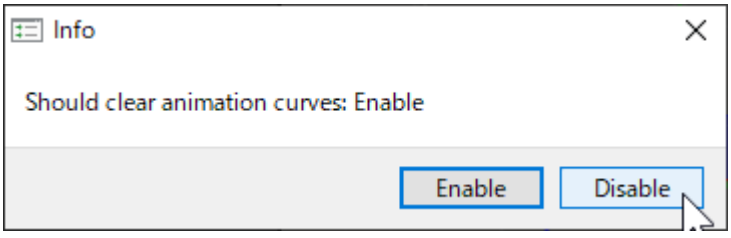


3. In the Project window, right-click on the folder of model data to be configured and click "Reimport".

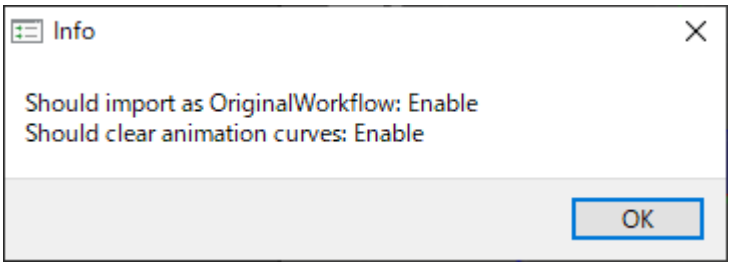


Re-import models in OW mode

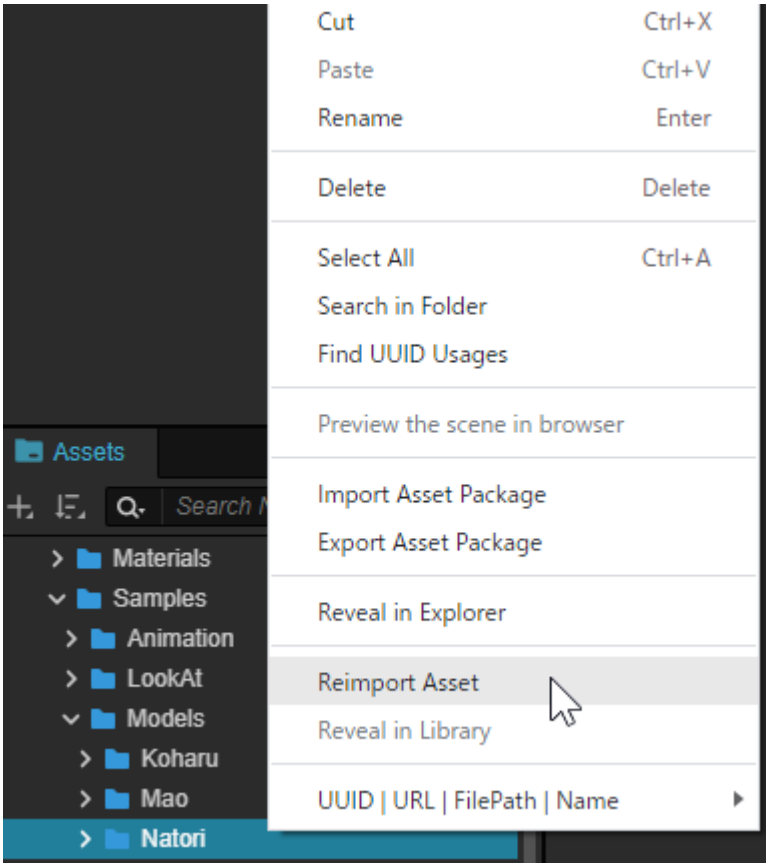
1. Select "Live2D/Cubism/OrignalWorkflow/Should Clear Animation Curves" from the Cocos Creator editor menu and set to Disable in the dialog that appears.



If the display looks like the following, it is OK.



2. In the Project window, right-click on the folder of model data you wish to re-import and click "Reimport".



Save/Restore the values to be manipulated

Summary

This page describes the procedure for saving/restoring parameter values and part opacity to a Cubism model using ParameterStore. The following assumes that the SDK is added to a project that has already been "Import SDK" and "How to set up UpdateController".

About CubismParameterStore

If you click "Live2D/Cubism/OrignalWorkflow/ **Toggle Should Import As Original Workflow**" in the Cocos Creator menu and import a model with **true** status, the generated Prefab will have a CubismParameterStore component.

CubismParameterStore is a component that restores and stores Cubism model parameter values and part opacity before and after the AnimationClip is played. This can be used to avoid problems that occur when other Cubism components manipulate values relative to Additive or Multiply. The value manipulations performed by Cubism components in Additive and Multiply blends assume that the value manipulations performed in the previous frame have been reset. If the value is overwritten by AnimationClip, then the previous value manipulation will be overwritten and the value manipulation by the Cubism component will work correctly. However, if the played animation does not manipulate its values, the last manipulated value will remain in the next update, so the values to be calculated may be duplicated, and the behavior may not work as expected. CubismParameterStore **saves** all parameter values of the model to which it is attached at the timing of **lateUpdate()** immediately after processing the animation, and **restores** the saved values at **update()** of the next frame. This allows components to operate correctly on parameters whose values are not overwritten by animation.

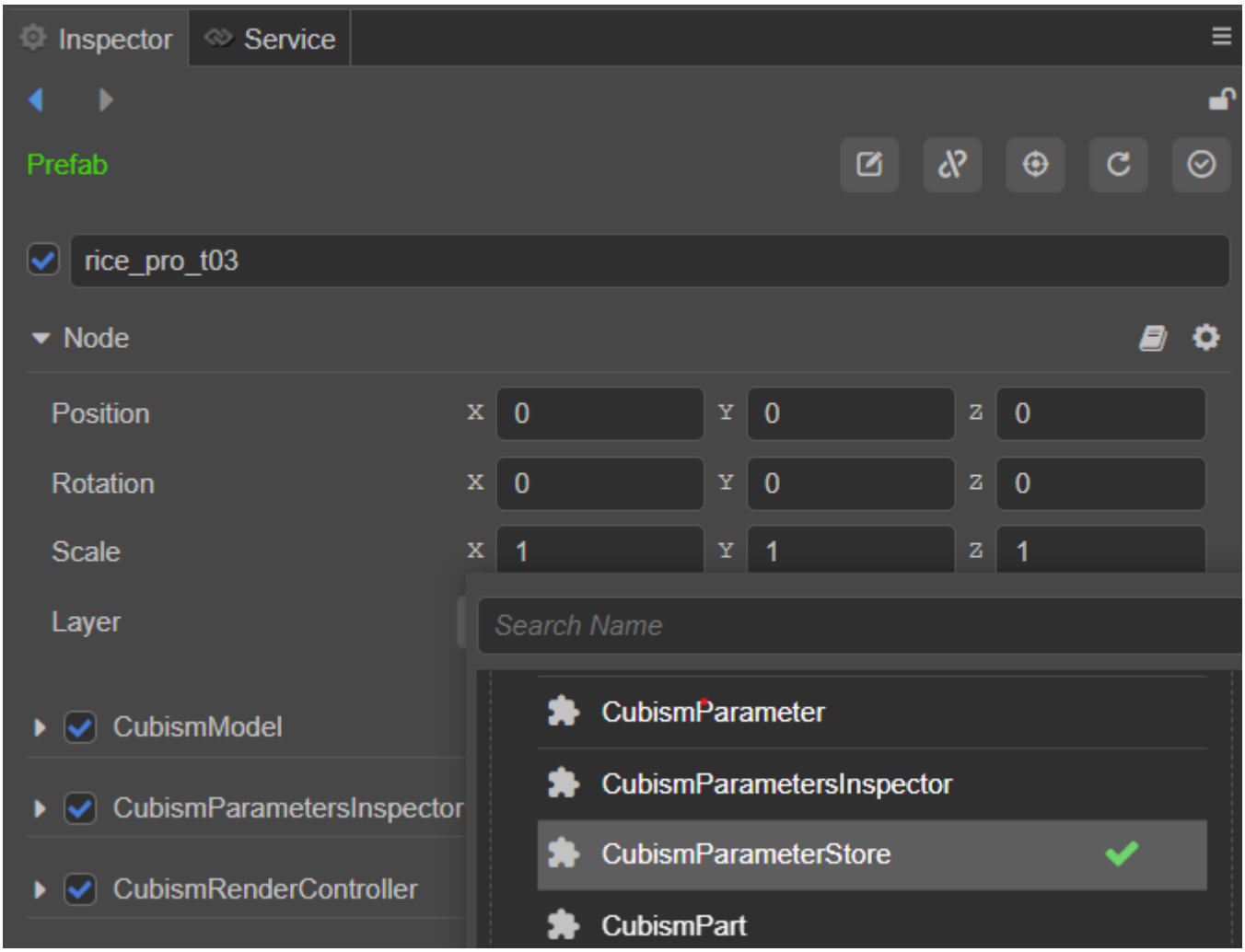
The following procedure is used to save/restore parameter values and part opacity in a model generated by the conventional method.

1. Attach CubismParameterStore

- To use ParameterStore, “UpdateController setting” is required.

1. Attach CubismParameterStore

Attach a “CubismParameterStore” to the Node that is the root of the model to store/restore values.


Inspector
Service
Prefab
rice_pro_t03
Node
Position
Rotation
Scale
Layer
CubismModel
CubismParametersInspector
CubismRenderController
Search Name
CubismParameter
CubismParametersInspector
CubismParameterStore
CubismPart

Let your own components control the order of execution

This section describes the procedure for controlling the order of execution among other Cubism components for a user's own component. The following explanation is based on the assumption that the model will be added to the project in which the［Import SDK-Placing Models］ step was performed.

Summary

Some components in the **Original Workflow** of the Cubism SDK for Cocos Creator have restrictions on the order in which they are executed.

In Cubism SDK for Cocos Creator , this can be controlled by using **CubismUpdateController**, which controls the order of execution of the above components. The component controlled by CubismUpdateController will be the component attached to the root of the Prefab of the Cubism model.

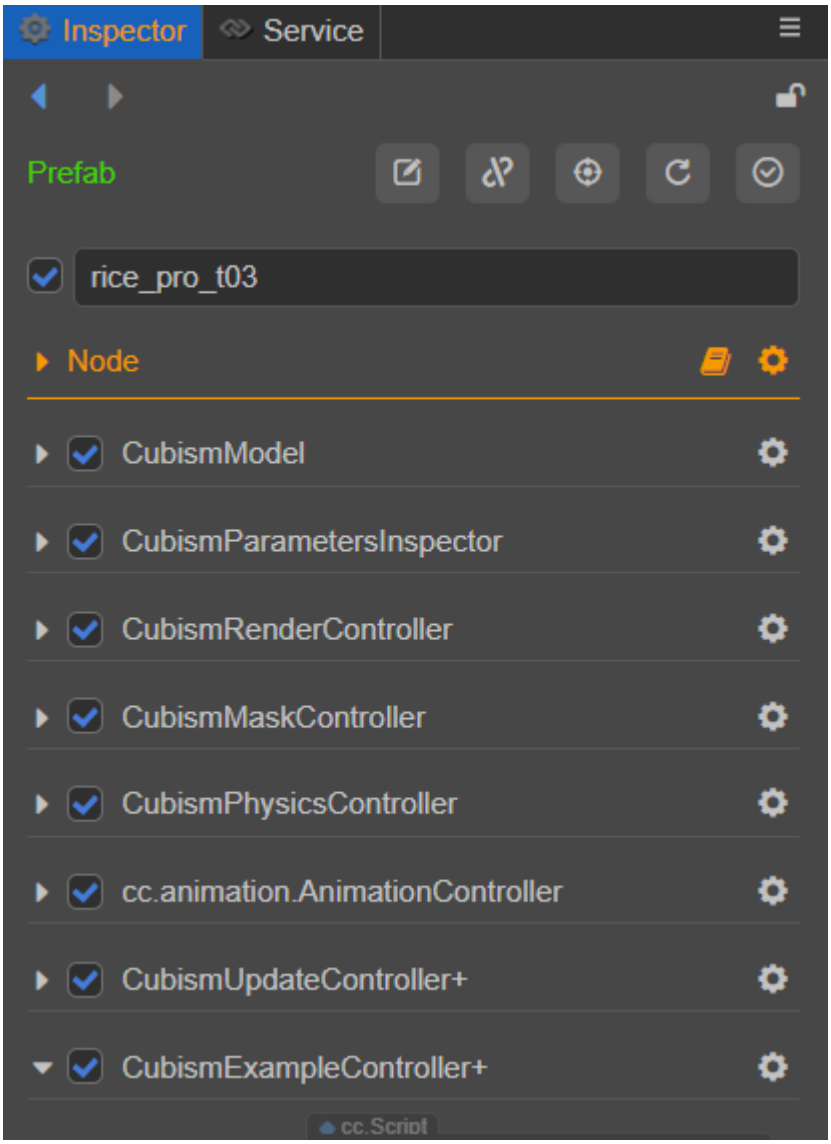
With CubismUpdateController, the user's own components can be controlled in the same order of execution.

As an example, this section describes the procedure for setting up execution order control for the following components.

```
@ccclass('CubismExampleController')
export class CubismExampleController extends Component {
    start() {
        // Initialization process of CubismExampleController
    }

    lateUpdate(deltaTime: number) {
        // Update process of CubismExampleController
    }
}
```

1. Attach components to Prefab

Attach **CubismExampleController** to the Node at the root of the Prefab placed in the Hierarchy. If Prefab is not imported in OW format, **CubismUpdateController** is also attached.



2. ICubismUpdatable is implemented in the component

Implement the **ICubismUpdatable** interface in the component that controls the execution order.

CubismUpdateController retrieves components that implement ICubismUpdatable at runtime and controls the order in which they are executed.

```
@ccclass('CubismExampleController')
export class CubismExampleController extends Component implements ICubismUpdatable
{

  bindedOnLateUpdate: ICubismUpdatable.CallbackFunction;
  get executionOrder(): number {
    throw new Error('Method not implemented.');
  }
  get needsUpdateOnEditing(): boolean {
    throw new Error('Method not implemented.');
  }
  get hasUpdateController(): boolean {
    throw new Error('Method not implemented.');
  }
```

```
  set hasUpdateController(value: boolean) {
    throw new Error('Method not implemented.');
  }

  readonly [ICubismUpdatable.SYMBOL]: typeof ICubismUpdatable.SYMBOL;

  protected start() {
    // Initialization process of CubismExampleController
  }

  protected lateUpdate(deltaTime: number) {
    // Update process of CubismExampleController
  }
}
```

The ICubismUpdatable interface implemented here is as follows.

```
interface ICubismUpdatable {
  readonly [ICubismUpdatable.SYMBOL]: typeof ICubismUpdatable.SYMBOL;
  readonly bindedOnLateUpdate: ICubismUpdatable.CallbackFunction;

  get executionOrder(): number;
  get needsUpdateOnEditing(): boolean;
  get hasUpdateController(): boolean;
  set hasUpdateController(value: boolean);
}
```

ExecutionOrder is the value that determines the order of execution for this component. The smaller this value, the earlier it is called before other components. The values set for the components included with the SDK are described in the CubismUpdateExecutionOrder.

HasUpdateController is a flag that allows components implementing ICubismUpdatable to be called from Cocos Creator event functions if the CubismUpdateController is not attached.。

```
namespace CubismUpdateExecutionOrder {
  export const CUBISM_FADE_CONTROLLER = 100;
  export const CUBISM_PARAMETER_STORE_SAVE_PARAMETERS = 150;
  export const CUBISM_POSE_CONTROLLER = 200;
  export const CUBISM_EXPRESSION_CONTROLLER = 300;
  export const CUBISM_EYE_BLINK_CONTROLLER = 400;
  export const CUBISM_MOUTH_CONTROLLER = 500;
  export const CUBISM_HARMONIC_MOTION_CONTROLLER = 600;
  export const CUBISM_LOOK_CONTROLLER = 700;
  export const CUBISM_PHYSICS_CONTROLLER = 800;
  export const CUBISM_RENDER_CONTROLLER = 10000;
  export const CUBISM_MASK_CONTROLLER = 10100;
```

3. Make the component compatible with CubismUpdateController

Modify CubismExampleController as follows.

```
import { _decorator, Component, Node } from 'cc';
import CubismUpdateController from
'../../extensions/Live2DCubismSdkForCocosExtension/static/assets/Framework/CubismU
pdateController';
import ICubismUpdatable from
'../../extensions/Live2DCubismSdkForCocosExtension/static/assets/Framework/ICubism
Updatable';
const { ccclass, property } = _decorator;

@ccclass('CubismExampleController')
export class CubismExampleController extends Component implements ICubismUpdatable
{
  bindedOnLateUpdate: ICubismUpdatable.CallbackFunction;

  // Execution order of this component
  get executionOrder(): number {
      return 150;
  }

  // Control the order of execution during Scene non-execution?
  get needsUpdateOnEditing(): boolean {
      return false;
  }

  // Is the execution order controlled?
  @property({ serializable: false, visible: false })
  private _hasUpdateController: boolean = false;
  get hasUpdateController(): boolean {
      return this._hasUpdateController;
  }
  set hasUpdateController(value: boolean) {
      this._hasUpdateController = value;
  }

  readonly [ICubismUpdatable.SYMBOL]: typeof ICubismUpdatable.SYMBOL;

  protected start() {
    // Initialization process of CubismExampleController

    // Check if CubismUpdateController is attached to the model Prefab
    this.hasUpdateController = this.getComponent(CubismUpdateController) != null;
  }

  protected lateUpdate(deltaTime: number) {
    // If CubismUpdateController is not attached, update processing is performed
from the event function of CubismExampleController itself
    if (!this.hasUpdateController) {
        this.onLateUpdate(deltaTime);
    }
  }
```

```
    // Update functions with controlled order of execution
    public onLateUpdate(deltaTime: number) {
      // Update process of CubismExampleController
    }
  }
```

The update process performed by LateUpdate() is moved to **onLateUpdate**(), which is called by CubismUpdateController.

This completes the setup for controlling the order of execution. When this script is attached to the Prefab of the Cubism model and the scene is executed, the update process of this script is called by the CubismUpdateController.