Scriptable Blendshapes Documentation

For instant replies to advanced queries about using Scriptable Blendshapes in your game, you can use our custom code-support GPT that is trained on source code for both scriptable blend shapes, and Unity Engine. *Note: There are usage restrictions for users without plus accounts.*

Support GPT: https://chatgpt.com/g/g-d38LSkZ7s-scriptable-blendshapes-code-assistant

1) Introduction

The Scriptable Blendshapes asset is a powerful Unity tool designed to simplify and enhance the management of blendshapes in 3D models, particularly for character animation and customization. By leveraging Scriptable Objects to store blendshape data, this asset allows for advanced automation, batch editing, and hierarchical organization of blendshapes, which can significantly improve workflows in projects with complex character rigs and facial animations.

Key Problems Solved:

1. Tedious Blendshape Management: In projects with numerous blendshapes, managing them through Unity's default interface can become cumbersome. Scriptable Blendshapes allows you to organize blendshapes into logical groups, classes, and subtypes, making them easier to search, filter, and manipulate both in the editor and at runtime.

2. Improved Readability and Maintainability of Scripts: Unity's default method for controlling blendshapes involves referencing them by index, which can make code hard to understand and maintain, especially as the number of blendshapes increases. The asset provides a solution via the UB3 Code Bridge, allowing developers to manipulate blendshapes by name or hierarchical structure instead of relying on indices.

3. Batch Processing: For characters with numerous blendshapes (e.g., for facial expressions or body part deformations), individually assigning metadata, weights, or making changes can be time-consuming. The Batch Blendshape Editor Tool allows you to process multiple blendshapes simultaneously, significantly speeding up development time.

4. Streamlined Customization Systems: The asset allows for easy integration of blendshapes into character customization systems. Using the hierarchy structure and the UB3 Code Bridge, developers can create intuitive sliders or UI elements for users to modify a character’s appearance in real time, without worrying about complex scripting or indexing errors.

2) Key Features

The Scriptable Blendshapes asset provides a range of powerful tools and features that enable users to manage and manipulate blendshapes more efficiently in Unity. Here's a breakdown of the key features:

Advanced Blendshape Management Using Metadata

The core of the asset is built around the BlendshapeMetadata Scriptable Object, which allows users to store, organize, and access all relevant data for each blendshape. This metadata encapsulates key information such as blendshape names, weights, indices, and hierarchical relationships, giving you fine-grained control over how blendshapes are processed and manipulated at runtime. By using metadata, blendshapes can be batch processed, automated, and easily accessed through intuitive structures, reducing the need for manually handling blendshapes in Unity's default system.

Hierarchical Structure for Blendshape Classes, Groups, and Subtypes

Managing a large number of blendshapes can be overwhelming without proper organization. The Blendshape Class Hierarchy system addresses this by allowing you to group blendshapes into Classes, Groups, and Subtypes, which creates a logical structure for filtering and searching through them. This hierarchical system is fully customizable, enabling developers to define categories that make sense for their project—whether it's facial expressions (e.g., eyes, mouth, nose) or body deformations. The structure can be reused across multiple assets, making it ideal for projects with multiple characters or modular meshes.

UB3 Code Bridge for Simplified Scripting

Writing scripts to control blendshapes is much easier with the UB3 Code Bridge, which acts as a bridge between blendshape metadata and custom scripts. Instead of working with obscure blendshape index numbers, the UB3 Code Bridge allows developers to reference blendshapes by their names, classes, groups, and subtypes. This makes code more readable and maintainable, which is especially useful in large projects or when multiple team members are involved. The bridge also offers a set of public methods for manipulating blendshapes, reducing the amount of custom code needed for basic operations such as setting blendshape weights or resetting them.

Editor and Runtime Tools for Batch Editing and Automation

The Batch Blendshape Editor Tool and Blendshape Management Tool provide powerful, user-friendly interfaces in the Unity editor to streamline the process of managing blendshapes. These tools enable users to:

- Batch process multiple blendshapes at once, assigning class, group, or subtype data to large groups of blendshapes.
- Apply consistent changes across many blendshapes in a few clicks, saving valuable development time.
- Automate blendshape tasks like setting initial weights, creating presets, and reorganizing blendshapes in the hierarchy, ensuring efficient workflows for projects with large numbers of blendshapes.
-

Additionally, the UB3 Code Bridge methods can be executed at runtime, allowing developers to dynamically adjust blendshapes during gameplay, such as for character customization sliders or emotional state systems.

3) Supported Unity Versions:

- Unity 2021.3 LTS or newer: The asset is compatible with Unity versions from 2021.3 LTS onwards. It fully supports Unity's Scriptable Object system and works with both all render pipelines. Note: Our demo scene is only for the built-in render pipeline.

- Animator Support: The asset works seamlessly with Unity's Mecanim animation system, as well as any other systems that use Skinned Mesh Renderers and blendshapes.

Back to Table of Contents

4) Use Cases

The Scriptable Blendshapes asset is a versatile tool that can be applied to a variety of scenarios in game development, animation, and interactive applications. Below are some common use cases where the asset provides significant benefits:

Facial Animations

- For games and applications that require real-time facial expressions, the asset provides an efficient way to manage and animate blendshapes.
- With the UB3 Code Bridge, you can easily script dynamic facial expressions, creating complex reactions like smiling, frowning, blinking, and other facial deformations by simply referencing blendshapes by name rather than index numbers.
- The hierarchical structure of blendshapes helps developers organize facial expressions by categories such as emotions (happy, sad, angry) or specific parts of the face (eyes, mouth, eyebrows).

Character Customization

- The asset is particularly useful in creating character customization systems, where users can adjust the appearance of characters in real-time using UI sliders or dials.
- By organizing blendshapes into categories such as body type, facial features, or clothing adjustments, the tool makes it easy to create an intuitive interface where players can control various aspects of the character's appearance.
- The batch processing tools allow developers to quickly set up large-scale customization systems with consistent blendshape groupings across multiple characters.

Dynamic In-Game Reactions

- For games that require in-game reactions (e.g., characters responding to attacks, surprises, or emotions), the blendshape hierarchy can be tied to triggers and events.
- Using the Emotional State System or Auto Facial Reactions assets (sold separately), developers can automate facial animations based on game states like health, mood, or interaction with other characters. For example, a character can wince or grimace when taking damage, or smile in response to positive interactions.

Complex Multi-Character Animation Systems

- In large-scale animation projects with multiple characters, Scriptable Blendshapes allows developers to standardize blendshape management across different models.
- By reusing BlendshapeClassHierarchy objects, you can maintain consistency in blendshape structures across different meshes, making it easier to automate and manage multi-character scenes.

Procedural Character Generation

- The asset integrates smoothly with systems designed for procedural character generation, such as random character creation for games with large NPC populations.
- Using the Billions of Characters asset (sold separately), the tool can be used to generate a virtually infinite number of unique characters by randomly assigning weights to blendshapes across various categories (e.g., head size, nose shape, eye position), giving each character a unique appearance.

Performance Optimization for Mobile and VR Applications

- In performance-critical environments such as mobile games or virtual reality, where optimization is key, the asset can work alongside the Blendshape Optimizer (sold separately) to bake or remove unnecessary blendshapes, improving runtime performance without sacrificing quality.
- This makes it possible to use complex blendshapes in highly optimized mobile or VR experiences.

Back to Table of Contents

5) Installation

To get started with Scriptable Blendshapes, you can import the package in two ways, depending on whether you purchased it from the Unity Asset Store or downloaded it from our website.

Importing via the Unity Asset Store

Open Unity and go to the top menu. Select Window > Package Manager. In the Package Manager window, search for "Scriptable Blendshapes" under My Assets. If you haven't downloaded it yet, click the Download button. Once downloaded, the button will change to Import. Click Import to add the package to your project. Unity will then show an import window; ensure all files are selected and confirm the import.

Importing a Unity Package File from the Website

If you purchased Scriptable Blendshapes from our website, you will receive a .unitypackage file. First, save this file to your computer. Then, open Unity and select Assets > Import Package > Custom Package from the top menu. Locate and select the .unitypackage file you downloaded. Unity will display the package contents in an import window. Make sure all files are selected, then click Import.

6) Core Components

The scriptable Blendshapes asset ships with numerous editor tools and runtime components that form the basis of the UltimateBlendshapes ecosystem. Below we will cover the individual scripts included.

BlendshapeMetadata.cs

The BlendshapeMetadata Scriptable Object is designed to store and manage detailed data related to the blendshapes on a mesh. This metadata enables efficient organization and automation of blendshape-related tasks, such as processing, filtering, and customizing blendshape behavior for runtime applications.

Fields and Properties

The core data structure within BlendshapeMetadata is the BlendshapeData class, which defines the following fields for each blendshape:

- **masterIndex**: The index of the blendshape from the original list, used to track its position in the Skinned Mesh Renderer.
- **originalName**: The original name of the blendshape, as defined in the mesh.
- **nameOverride**: A user-defined name override for easier identification (if needed).
- **value**: The current weight or value assigned to the blendshape, typically ranging from 0 to 100.
- **bake**: A boolean indicating whether this blendshape should be baked (permanently applied to the mesh) during runtime or processing.
- **minValue** and **maxValue**: The minimum and maximum values allowed for the blendshape's weight.
- **defaultValue**: The default value assigned to the blendshape when resetting it to its initial state.
- **classTag**, **groupTag**, **subtypeTag**: Tags used to organize the blendshape within a broader hierarchy for filtering and classification.
- **isProcessed**: A boolean flag to track whether this blendshape has been processed through tools such as batch editors.

Creating and Managing Blendshape Metadata

The BlendshapeMetadata Scriptable Object allows you to manually or programmatically add and initialize blendshapes. You can add new blendshapes using the AddBlendshape method, which assigns default values and makes it easy to manage large numbers of blendshapes across multiple meshes.

You can also initialize all blendshapes using the InitializeBlendshapes method, which ensures all blendshapes have default values for class tags, group tags, and weight limits. Additionally, the ResetBlendshapeToDefaults method allows you to reset a blendshape to its default state, including its weight and organizational tags.

Example workflows include creating metadata for blendshapes used in character customization sliders, facial animation systems, or automated expression triggers. By organizing blendshapes with the class/group/subtype tags, it becomes easier to manipulate blendshapes in both runtime scripts and editor tools.

BlendshapeClassHierarchy.cs

The BlendshapeClassHierarchy Scriptable Object provides a structured way to organize blendshapes using a class, group, and subtype system. This hierarchy allows for more efficient searches, filtering, and assignments of blendshapes across various tools or scripts.

Class, Group, and Subtype System

The hierarchy is structured using two main nested classes:

- **ClassGroup**: Represents a high-level category (class) for organizing blendshapes. Each ClassGroup contains multiple Group objects and includes a boolean field (isBaked) to track whether the blendshapes in this class should be baked into the mesh.

- **Group**: A collection of related blendshapes within a ClassGroup. Each Group contains a list of subtypes, which further narrows the scope of blendshapes, allowing for granular control over specific features or behaviors.

For example, you might create a class called "Facial Expressions" with groups like "Mouth" and "Eyes," and then assign subtypes like "Smile" or "Blink" within each group.

Best Practices for Organizing Blendshapes

When using BlendshapeClassHierarchy, it's recommended to keep the structure simple and intuitive. Avoid overly complex classifications that could complicate your workflow. Instead, focus on logically grouping blendshapes by their function (e.g., customization sliders, facial features) or purpose (e.g., emotional states, body modifications). This organization will help with both runtime manipulation and editor-based management of blendshapes.

Sharing Hierarchies Between Assets

One of the key advantages of using BlendshapeClassHierarchy is its ability to be shared across multiple meshes or projects. You can create a BlendshapeClassHierarchy Scriptable Object in your project and apply it to various assets, ensuring consistent organization and streamlined blendshape management across different models. This feature is especially useful for projects with multiple character assets or meshes that require consistent naming and tagging conventions for blendshapes.

By leveraging this hierarchy, you can significantly enhance your workflow, particularly when dealing with large-scale projects or complex blendshape setups that need to be maintained across different assets.

Back to Table of Contents

7) Editor Tools

Blendshape Management Tool (BlendshapeManagementTool.cs)

The Blendshape Management Tool is an editor window that simplifies the process of creating, organizing, and managing blendshapes through a user-friendly interface. It allows developers to generate and edit BlendshapeMetadata and BlendshapeClassHierarchy Scriptable Objects, while efficiently processing and managing blendshapes for use in various systems, including character customization and animation.

User Interface Breakdown

The interface is divided into three resizable columns:

- **First Column**: Contains hierarchy settings, filter options, and fields for assigning Scriptable Objects such as BlendshapeMetadata and BlendshapeClassHierarchy. It allows you to assign a Skinned Mesh Renderer and either create or load metadata objects. The search and filter fields in this column enable users to quickly locate specific blendshapes based on name, class, group, or subtype.

- **Second Column**: Displays a list of unprocessed blendshapes. These are blendshapes that have not been fully assigned to a class/group/subtype or otherwise processed. You can modify their properties directly from this column, such as their weight, name override, or baking status.

- **Third Column**: Shows the processed blendshapes. These are blendshapes that have been organized and are ready for further actions, such as being assigned to animations or other systems. The processed column allows you to review and edit blendshapes or even reset them back to unprocessed status if necessary.

Detailed Workflow

The Blendshape Management Tool provides a step-by-step approach to managing blendshapes:

1. **Assign a Skinned Mesh Renderer**: In the first column, assign the mesh you want to work with by selecting its Skinned Mesh Renderer.

2. **Create or Assign Metadata**: Either create a new BlendshapeMetadata Scriptable Object or assign an existing one. You can also create and assign a BlendshapeClassHierarchy to categorize and filter your blendshapes by class, group, and subtype.

3. **Filter and Search**: Use the search filters to locate specific blendshapes by name, class, group, or subtype. This helps streamline workflows, especially when working with large meshes that have numerous blendshapes.

4. **Process Blendshapes**: In the second column (unprocessed blendshapes), adjust properties like name, value (weight), and baking status. You can also set minimum, maximum, and default values for each blendshape.

5. **Move to Processed**: Once a blendshape is fully configured, move it to the processed blendshapes column. From there, it can be further edited or marked as finalized.

6. **Save Changes**: After processing, save the blendshape metadata to retain the changes made to each blendshape.

Practical Examples

- **Creating Blendshape Hierarchies**: By using the class/group/subtype structure, you can organize blendshapes logically. For example, you can create a class called "Facial Expressions," with groups like "Mouth" or "Eyes," and subtypes such as "Smile" or "Blink."

- **Assigning Metadata**: After assigning a Skinned Mesh Renderer and creating blendshape metadata, you can quickly manage blendshape weights and override names directly within the editor, streamlining the workflow for facial rigs or character customization systems.

- **Editing Blendshape Weights**: The second and third columns allow you to adjust blendshape weights either individually or as part of a batch processing operation.

Common Use Cases

- **Managing Facial Animation Blendshapes for Character Rigs**: This tool is ideal for managing blendshapes that are tied to facial animation systems, allowing quick updates to expressions or customizing character faces for different emotions.

- **Editing Blendshape Weights for Multiple Blendshapes**: When working with complex character meshes, the tool allows you to batch edit blendshapes, modifying multiple properties at once to ensure consistency across various body parts or expressions.

Back to Table of Contents

Batch Blendshape Editor Tool (BatchBlendshapeEditorTool.cs)

The Batch Blendshape Editor Tool is designed to streamline the process of managing and assigning blendshape data across multiple blendshapes simultaneously. This tool provides batch processing capabilities, allowing developers to efficiently categorize and edit multiple blendshapes at once, reducing the time and effort needed for large projects or complex character meshes.

User Interface Breakdown

The Batch Blendshape Editor Tool consists of two main columns that facilitate batch editing:

- **First Column: Filters and Batch List**
  This column allows you to assign a SkinnedMeshRenderer, BlendshapeMetadata, and BlendshapeClassHierarchy Scriptable Objects. It includes filters for searching blendshapes by name, and drop-down menus for selecting class, group, and subtype data. There is also a section for managing the batch list, displaying selected blendshapes that are queued for batch processing. This allows you to assign metadata to multiple blendshapes at once.

- **Second Column: Selected Blendshapes for Batch Processing**
  This column displays a list of blendshapes—either processed or unprocessed—depending on the settings in the first column. You can review and modify the blendshapes' metadata, adjust their values, and add them to the batch list for processing. The interface allows you to easily search through blendshapes, view their details, and apply batch operations.

Batch Processing Workflows

The tool enables quick batch operations for organizing blendshapes into classes, groups, and subtypes. Below is a step-by-step guide on how to use the batch processing features:

1. **Assign Required Objects**
   Start by assigning the SkinnedMeshRenderer of the mesh you want to work with. Then, assign the BlendshapeMetadata and BlendshapeClassHierarchy Scriptable Objects, which will hold the blendshape data and classification structure.

2. **Search and Filter Blendshapes**
   Use the search field to filter blendshapes by their original name. This is particularly useful when working with a large number of blendshapes. You can toggle between unprocessed and processed blendshapes to locate the ones you need to modify.

3. **Select Blendshapes for Batch Processing**
   From the list of blendshapes in the second column, you can add individual blendshapes to the batch list. These blendshapes will then appear in the batch list in the first column.

4. **Assign Class, Group, and Subtype Data**
   After adding blendshapes to the batch list, use the drop-down menus in the first column to select a class, group, and subtype. Once your selection is made, click the "Assign Class" button to apply these tags to all blendshapes in the batch list.

5. **Save and Process**
   Once your batch operations are complete, the blendshapes will be updated with the new metadata, including their class, group, and subtype tags. The tool ensures that blendshapes are reset to their default values where necessary, and the metadata is saved automatically.

Common Scenarios

- **Bulk Editing Facial Features for Character Customization**
  When working on a character customization system, you may need to adjust multiple facial blendshapes (e.g., eyes, mouth, eyebrows) at once. The Batch Blendshape Editor allows you to quickly assign classifications (such as "Facial Features" or "CustomizationSliders") to all relevant blendshapes, ensuring they are grouped correctly for future use.

- **Managing Blendshapes for Complex Characters**
  In large projects with numerous blendshapes, this tool simplifies the task of organizing blendshapes into meaningful categories. For instance, you can categorize blendshapes based on their use in animations, customization, or other systems, and apply these classifications in bulk to speed up development.

UB3 Bridge Code Generator (UB3BridgeCodeGenerator.cs)

The UB3 Bridge Code Generator simplifies generating scripts for blendshape manipulation, whether using the UB3 Code Bridge or Unity's default methods. The tool makes it easy to generate readable and organized code based on blendshape classifications such as class, group, and subtype. This allows for efficient management of complex blendshape systems.

Left Column: Blendshape Renderer and Metadata Setup

This section is used to set up the necessary components for blendshape manipulation:

- **Skinned Mesh Renderer**: This field is where you assign the Skinned Mesh Renderer that contains the blendshapes you want to manipulate.
- **Blendshape Metadata**: This field is used to assign the Blendshape Metadata Scriptable Object, which contains information about the blendshapes, including their name overrides, classes, groups, and subtypes.
- **Class Hierarchy**: Here, you assign the Blendshape Class Hierarchy Scriptable Object, which organizes blendshapes into a class, group, and subtype structure. This makes it easier to reference specific sets of blendshapes in your scripts.

Search and Filter Section

This section provides options to search and filter the blendshapes that will appear in the right column for code generation:

- **Search by Name**: Input a search term to filter blendshapes by their original name or name override.
- **Search by Original Name?**: Toggle this option to switch between searching blendshapes by their original name or the custom name override.
- **Class, Group, Subtype Filters**: These text fields allow you to filter blendshapes based on their class, group, and subtype, which are defined in the Blendshape Metadata and Class Hierarchy.

Back to Table of Contents

Code Generation from Hierarchy Section

In this section, you can generate code for entire classes, groups, or subtypes:

- **Class/Group/Subtype Dropdowns**: You can select a class, group, and subtype from the dropdowns, which are populated based on the Class Hierarchy assigned in the previous section.

- **Method Dropdown**: Select the operation you want to perform on the selected class, group, or subtype (e.g., Set, Increase, Decrease, Lerp).

- **Value and Duration Inputs**: For methods like Set or Lerp, input fields for the target value and duration (for smooth transitions) are shown.

- **Generate Code Buttons**: After selecting the method and input values, click on buttons to generate code for the selected class, group, or subtype.

Right Column: Blendshape List and Code Generation

This section shows the list of blendshapes based on the filters applied:

- **Blendshape List**: Displays blendshapes from the assigned Blendshape Metadata, along with their nameOverride and index.

- **Value Slider**: Allows real-time manipulation of the blendshape weight via a slider.

- **Foldout: Code Generation**: Each blendshape has a foldout section with buttons to generate code for different operations (Set, Increase, Decrease, Lerp, etc.) for that specific blendshape.

Generated Code Output

At the bottom of the window is a text area where the generated code snippets appear. You can copy the generated code to the clipboard or clear it:

- **Copy Code to Clipboard**: Copies the generated code to your clipboard for use in your scripts.

- **Clear Generated Code**: Clears the current generated code from the window.

This layout simplifies the process of generating script snippets for manipulating blendshapes through the UB3 Code Bridge system, making it easy to integrate blendshape manipulation into your project.

Back to Table of Contents

Generating UB3 Code Bridge Scripts

The UB3 Code Bridge enables you to manipulate blendshapes by referencing names and tags, rather than relying on numerical indices. This improves code readability and maintainability, especially when managing large numbers of blendshapes.

Start by assigning the SkinnedMeshRenderer, BlendshapeMetadata, and BlendshapeClassHierarchy in the tool. You can then use the provided filters to search for blendshapes by name, class, group, or subtype.

Once you've set the filters, select an operation to apply to the blendshapes. The available operations include Set, Increase, Decrease, Reset, and "Smooth Lerp" variations of these operations that allow gradual transitions over time. You can apply these operations either to specific blendshapes or to groups of blendshapes.

After choosing the operation, click the appropriate button to generate the UB3 Code Bridge script. For example, to set the weight of a blendshape by its name, the tool will generate the following code:

```
ub3CodeBridge.SetBlendshapeWeightByName("BlendshapeName", 50f);
```

This approach allows you to work with blendshapes by name, making the code clearer and easier to understand compared to using index-based references.

Generating Default Unity Blendshape Scripts

The tool can also generate scripts using Unity's default methods for blendshape manipulation, which rely on numerical indices.

For example, if you want to set the weight of a blendshape using its index, the tool generates the following code:

```
skinnedMeshRenderer.SetBlendShapeWeight(0, 50f);
```

This directly modifies the blendshape at index 0, setting its weight to 50. While this method is efficient, it can be harder to manage as blendshape indices change or when working with a large number of blendshapes.

If you want to transition a blendshape value smoothly over time, the tool can generate coroutine-based code like this:

```
StartCoroutine(LerpBlendshape(0, 50f, 1f)); // Lerp to 50 over 1 second
```

Pros and Cons of Using Unity's Default Methods vs. UB3 Code Bridge

The UB3 Code Bridge offers several benefits. It makes your code easier to read and maintain by using human-readable names and classifications instead of numerical indices. This is especially valuable in large projects where managing blendshapes by index can be cumbersome. However, the UB3 Code Bridge adds a small overhead in performance because of the additional layer of abstraction.

Unity's default methods are more efficient at runtime because they directly manipulate blendshapes using indices. This makes them a good choice for small projects or situations where performance is a priority. However, the reliance on indices makes the code harder to maintain in larger projects, especially when blendshape indices change or when managing large sets of blendshapes.

The UB3 Bridge Code Generator provides the flexibility to choose the method that best suits your project's needs—whether you prioritize readability and maintainability or performance and simplicity.

Back to Table of Contents

8) Runtime Tools

UB3 Code Bridge (UB3CodeBridge.cs)

The UB3 Code Bridge component allows developers to easily control blendshapes using names and classification tags (such as class, group, and subtype) instead of relying on index numbers. This approach improves the readability and maintainability of the code while giving users the flexibility to manipulate multiple blendshapes in bulk.

Component Setup

1) **Attach the Component**: Add the UB3CodeBridge to a GameObject in your scene.

2) **Assign Metadata & Mesh**: In the Inspector, assign the Skinned Mesh Renderer of the mesh that contains the blendshapes. Then, assign the corresponding BlendshapeMetadata Scriptable Object, which stores the blendshape data for the mesh.

Public Methods

The UB3 Code Bridge provides various public methods that allow you to control blendshapes easily. For a detailed list visit the Appendices near the end of the documentation. Below are a few common examples:

SetBlendshapeWeightByName(string nameOverride, float weight)

Sets the weight of a specific blendshape using its nameOverride.

SetBlendshapeWeightByClass(string classTag, float weight)

Sets the weight for all blendshapes that share the same class.

SetBlendshapeWeightByGroup(string groupTag, float weight)

Sets the weight for all blendshapes that share the same group.

SetBlendshapeWeightBySubtype(string subtypeTag, float weight)

Sets the weight for all blendshapes that share the same subtype.

Back to Table of Contents

9) Performance Considerations

When using the UB3 Code Bridge in large projects with many blendshapes, you should be mindful of performance, especially when using the smooth Lerp methods. Here are a few optimization tips:

- **Use Bulk Methods**: When possible, use methods like SetBlendshapeWeightByClass or SetBlendshapeWeightByGroup to modify multiple blendshapes at once, reducing the overhead of handling individual blendshapes.

- **Cache Blendshape Data**: The UB3 Code Bridge caches blendshape indices for faster access. Ensure the InitializeBlendshapeMap() method is called to optimize the lookup of blendshape names.

- **Avoid Excessive Lerp**: While Lerp methods provide smooth transitions, they can add performance overhead if used excessively in complex scenes. Limit their usage to key interactions or reduce the duration for smoother performance.

By following these guidelines, the UB3 Code Bridge can be an effective tool for managing blendshapes in both small and large-scale projects.

10) Detailed Workflow and Tutorials

To begin working with blendshapes in UB3, the first step is to create a BlendshapeMetadata object, which will store information about each blendshape and allow for easy manipulation later in the workflow.

Steps to Create BlendshapeMetadata:

1. Open the Blendshape Management Tool from the "Ultimate Blendshapes" menu in Unity.
2. Assign a Skinned Mesh Renderer by dragging the mesh (e.g., a character model) into the appropriate field.
3. Create New Metadata by clicking the “Create New Blendshape Metadata” button. This will generate a Scriptable Object to store all the blendshape data for that mesh.
4. Populate Metadata: The tool will automatically detect the blendshapes present in the Skinned Mesh Renderer and list them under unprocessed blendshapes.

Example: Setting Up Blendshapes for a Facial Rig For a facial rig, blendshapes might include expressions like Smile, Frown, or Blink. Once the blendshapes are detected:

- Assign meaningful overrides for the blendshape names if necessary (e.g., rename BS_01 to Smile).
- Define the blendshape weight limits (e.g., 0-100 for a standard expression).
- Set default values to ensure consistent behavior across different poses.

Organizing Blendshapes with Hierarchies

Organizing blendshapes into logical hierarchies is essential for simplifying blendshape management, especially in complex rigs. The hierarchy system allows you to group blendshapes into categories like Eyes, Mouth, and Face and sub-categorize them further if needed.

Steps to Create a Class, Group, and Subtype Hierarchy:

1. Open the Blendshape Management Tool and assign the appropriate Blendshape Class Hierarchy object.

2. Create New Class Hierarchy by clicking the "Create New Class Hierarchy" button.

3. Define Classes: Create broad categories like FacialExpressions, BodyMorphs, or CustomizationSliders.

4. Add Groups within each class. For example, in FacialExpressions, create groups like Eyes, Mouth, or Brows.

5. Assign Subtypes within each group to fine-tune the organization. For example, under the Mouth group, add subtypes like Smile, Frown, OpenMouth.

Practical Use Case: Organizing Blendshapes for Animation and Customization

Class: FacialAnimation

  Group: Eyes

    Subtypes: Blink, Squint, WideOpen

Class: FacialCustomization

  Group: Eyes

    Subtypes: Size, Position, Shape

This organization allows you to quickly access and modify related blendshapes in a systematic manner, making it ideal for symetrical character animation or customization using non-symmetrical blendshapes.

Batch Processing Blendshapes

The Batch Blendshape Editor allows you to process multiple blendshapes simultaneously, greatly speeding up workflows where similar operations need to be applied to multiple blendshapes.

Steps to Batch Process Blendshapes:

1. Open the Batch Blendshape Editor Tool from the “Ultimate Blendshapes” menu.
2. Select Blendshapes: Use the search and filter options to locate the blendshapes you want to process.
3. Add Blendshapes to the Batch by selecting the desired blendshapes and adding them to the batch list.
4. Apply Metadata: Assign the same class, group, and subtype to all selected blendshapes in the batch list.
5. Process the Blendshapes: Hit "Process" to finalize the changes.

Example: Setting Up Blendshapes for Facial Expressions

When setting up blendshapes for common facial expressions, such as Smile, Frown, and Blink:

- Filter all mouth-related blendshapes using the search term “mouth.”
- Add them to the batch list, then assign the class FacialExpressions, group Mouth, and the relevant subtype (e.g., Smile or Frown).
- Batch process the list to apply the hierarchy and prepare the blendshapes for easy manipulation in your script or UI.

Scripting with the UB3 Code Bridge

Once your blendshapes are set up and organized, you can use the UB3 Code Bridge to write scripts that interact with blendshapes by name or tag, bypassing the need to work with index numbers.

Steps to Write Scripts with the UB3 Code Bridge:

1. Assign the UB3 Code Bridge Component to a GameObject in your scene.
2. Reference the Blendshape Metadata and Skinned Mesh Renderer in the Inspector.
3. Call Methods: Use the UB3 Code Bridge's public methods to manipulate blendshapes directly by name, class, group, or subtype.

Example: Creating a Simple UI to Adjust Blendshape Weights You can create a UI slider to adjust blendshape weights for character customization, such as controlling facial expressions.

```
//Create a UI slider in Unity.
//In your script, link the slider's value to the blendshape weight using the UB3 Code Bridge


public UB3CodeBridge codeBridge;
public Slider BodyFatSlider;

void Start () {

   smileSlider.onValueChanged.AddListener (UpdateBodyFatBlendshape);
}

void UpdateBodyFatBlendshape (float value) {

   codeBridge.SetBlendshapeWeightByName("BodyFat", value);
}
```

In this example, as the slider moves, it will call the SetBlendshapeWeightByName() method on the UB3 Code Bridge, dynamically adjusting the Smile blendshape weight for the character.

11) Troubleshooting: Common Issues and Solutions

Missing Blendshapes in the Editor

Cause: Blendshapes might not appear in the editor if the BlendshapeMetadata or Skinned Mesh Renderer is not properly assigned.

Solution:

- Verify that the Skinned Mesh Renderer is correctly linked in the Blendshape Management Tool.
- Ensure that the BlendshapeMetadata is assigned, and that the metadata corresponds to the mesh's blendshapes.
- If blendshapes are still missing, try refreshing the metadata by reinitializing the blendshape map or recreating the BlendshapeMetadata object.

Incorrect Weight Assignments

Cause: Incorrect weight assignments often occur when blendshape names in the BlendshapeMetadata do not match the actual blendshape names in the Skinned Mesh Renderer, or the wrong class/group/subtype has been assigned.

Solution:

- Double-check that the nameOverride in BlendshapeMetadata matches the actual blendshape name. The UB3 Code Bridge relies on the nameOverride for blendshape identification.
- Verify that the correct class, group, or subtype tags are assigned, especially when using batch processing or scripting with the UB3 Code Bridge.
- Use the GetBlendshapeWeightByName() method to check if the blendshape is correctly identified and its weight can be retrieved.

Back to Table of Contents

Performance Lags in Large Projects

Cause: Large numbers of blendshapes or frequent use of smooth transition methods (Lerp) can lead to performance bottlenecks.

Solution:

- Avoid running Lerp-based operations (e.g., LerpBlendshapeWeightByName()) excessively during every frame update, especially when working with many blendshapes.

- Use the UB3 Code Bridge's batch methods (SetBlendshapeWeightByClass(), SetBlendshapeWeightByGroup()) to modify multiple blendshapes at once rather than iterating through individual blendshapes.

Back to Table of Contents

12) Appendices

UB3 Code Bridge Public Methods (UB3CodeBridge.cs)

- **InitializeBlendshapeMap()**
  Initializes or refreshes the internal map linking blendshape names (nameOverride) to their corresponding blendshape indices.

- **SetBlendshapeWeightByName(string nameOverride, float weight)**
  Sets the weight of a specific blendshape using its nameOverride.

- **GetBlendshapeWeightByName(string nameOverride)**
  Returns the current weight of a blendshape identified by its nameOverride.

- **IncreaseBlendshapeWeightByName(string nameOverride, float increaseAmount)**
  Increases the weight of a blendshape by a specified amount.

- **DecreaseBlendshapeWeightByName(string nameOverride, float decreaseAmount)**
  Decreases the weight of a blendshape by a specified amount.

- **SetBlendshapeWeightByClass(string classTag, float weight)**
  Sets the weight for all blendshapes that share the same class.

- **SetBlendshapeWeightByGroup(string groupTag, float weight)**
  Sets the weight for all blendshapes that share the same group.

- **SetBlendshapeWeightBySubtype(string subtypeTag, float weight)**
  Sets the weight for all blendshapes that share the same subtype.

- **ResetAllBlendshapesToDefault()**
  Resets all blendshapes to their default values as defined in the BlendshapeMetadata.

- **ResetBlendshapeWeightByClass(string classTag)**
  Resets all blendshapes in a specific class to their default values.

- **ResetBlendshapeWeightByGroup(string groupTag)**
  Resets all blendshapes in a specific group to their default values.

Back to Table of Contents

- **ResetBlendshapeWeightBySubtype(string subtypeTag)**
  Resets all blendshapes in a specific subtype to their default values.

- **LerpBlendshapeWeightByName(string nameOverride, float targetWeight, float duration)**
  Smoothly transitions a blendshape's weight to a target value over a specified duration.

- **LerpIncreaseBlendshapeWeightByName(string nameOverride, float increaseAmount, float duration)**
  Smoothly increases a blendshape's weight by a specified amount over time.

- **LerpDecreaseBlendshapeWeightByName(string nameOverride, float decreaseAmount, float duration)**
  Smoothly decreases a blendshape's weight by a specified amount over time.

- **LerpResetBlendshapeWeightByName(string nameOverride, float duration)**
  Smoothly resets a blendshape's weight to its default value over time.

- **LerpResetBlendshapeWeightByClass(string classTag, float duration)**
  Smoothly resets all blendshapes in a specific class to their default values over time.

- **LerpResetBlendshapeWeightByGroup(string groupTag, float duration)**
  Smoothly resets all blendshapes in a specific group to their default values over time.

- **LerpResetBlendshapeWeightBySubtype(string subtypeTag, float duration)**
  Smoothly resets all blendshapes in a specific subtype to their default values over time.

- **LerpBlendshapeWeightByClass(string classTag, float targetWeight, float duration)**
  Smoothly transitions the weight of all blendshapes in a specific class to a target value over time.

- **LerpIncreaseBlendshapeWeightByClass(string classTag, float increaseAmount, float duration)**
  Smoothly increases the weight of all blendshapes in a class over time.

- **LerpDecreaseBlendshapeWeightByClass(string classTag, float decreaseAmount, float duration)**
  Smoothly decreases the weight of all blendshapes in a class over time.

- **LerpBlendshapeWeightByGroup(string groupTag, float targetWeight, float duration)**

Smoothly transitions the weight of all blendshapes in a specific group to a target value over time.

- **LerpIncreaseBlendshapeWeightByGroup(string groupTag, float increaseAmount, float duration)**
  Smoothly increases the weight of all blendshapes in a group over time.

- **LerpDecreaseBlendshapeWeightByGroup(string groupTag, float decreaseAmount, float duration)**
  Smoothly decreases the weight of all blendshapes in a group over time.

- **LerpBlendshapeWeightBySubtype(string subtypeTag, float targetWeight, float duration)**
  Smoothly transitions the weight of all blendshapes in a specific subtype to a target value over time.

- **LerpIncreaseBlendshapeWeightBySubtype(string subtypeTag, float increaseAmount, float duration)**
  Smoothly increases the weight of all blendshapes in a specific subtype over time.

- **LerpDecreaseBlendshapeWeightBySubtype(string subtypeTag, float decreaseAmount, float duration)**
  Smoothly decreases the weight of all blendshapes in a specific subtype over time.

Glossary of Terms

Below is a glossary of some of the terms we use in this documentation

Blendshape

A **blendshape** is a deformable shape in a 3D model that allows for the smooth transition between different shapes or expressions. Blendshapes are commonly used for character facial expressions (e.g., smiling, frowning, blinking), but can also be used for other kinds of morphs, such as muscle flexing or changing body features. In Unity, these are controlled via weights (usually between 0% and 100%) that define how much a particular blendshape is applied.

Metadata

Metadata refers to additional data that describes and organizes the blendshapes for a particular 3D model. In the context of Scriptable Blendshapes, metadata is stored in a BlendshapeMetadata Scriptable Object, which includes important information about each blendshape, such as its name, index, weight range, and hierarchical classification (class, group, and subtype). This metadata is critical for managing blendshapes efficiently in Unity projects.

Blendshape Metadata (BlendshapeMetadata.cs)

A **BlendshapeMetadata** Scriptable Object stores the details of each blendshape in a mesh, including:

- nameOverride: A user-friendly name for the blendshape.
- masterIndex: The blendshape’s index within the mesh.
- value: The current weight of the blendshape.
- classTag, groupTag, and subtypeTag: Tags used to categorize and organize blendshapes hierarchically. The metadata allows you to reference and modify blendshapes in a more readable way, especially when using scripting or automation tools like the UB3 Code Bridge.

Back to Table of Contents

UB3 Code Bridge (UB3CodeBridge.cs)

The **UB3 Code Bridge** is a Unity component that bridges the gap between blendshape metadata and scripts that manipulate those blendshapes. Instead of using index numbers to modify blendshapes (which can be difficult to manage), the UB3 Code Bridge allows you to reference blendshapes by their name or hierarchical tags (class, group, subtype). This greatly simplifies scripting and makes the code more readable and maintainable.

Key features of the UB3 Code Bridge include:

- Setting or retrieving blendshape weights by name.
- Batch setting blendshape weights by class, group, or subtype.
- Smoothly transitioning blendshape weights over time (Lerp functions).

Class, Group, Subtype Hierarchy

The **Class, Group, Subtype Hierarchy** is a logical structure used to organize blendshapes in a way that simplifies their management and application.

- **Class**: The broadest category, typically used to separate different types of blendshapes (e.g., FacialExpressions, BodyMorphs, CustomizationSliders).
- **Group**: A subdivision within a class (e.g., Eyes, Mouth, Arms).
- **Subtype**: The most granular categorization, often referring to specific blendshapes within a group (e.g., Smile, Frown, Blink). This hierarchical structure helps when applying batch changes or filtering blendshapes, making it easier to manage complex rigs or character customization systems.

Skinned Mesh Renderer

A **Skinned Mesh Renderer** is a Unity component that renders a 3D model that uses bones (for skeletal animation) and blendshapes (for morphing). It allows the model to deform and move in complex ways, often used for characters. Blendshapes in the Skinned Mesh Renderer can be controlled via scripts or UI components, with their weights determining how much each shape is applied.

Back to Table of Contents

Lerp (Linear Interpolation)

Lerp is a common term in programming and game development that refers to **linear interpolation** between two values over time. In the context of blendshapes, Lerp methods in the UB3 Code Bridge allow for smooth transitions between current and target blendshape weights over a specified duration. This is often used to create smooth animations or adjustments for character expressions or morphs.

Batch Processing

Batch Processing refers to the method of applying the same operation or metadata (such as class, group, or subtype assignments) to multiple blendshapes at once. The **Batch Blendshape Editor** tool allows users to streamline their workflow by processing multiple blendshapes simultaneously, reducing manual work and ensuring consistency across similar blendshapes.

Name Override

The **Name Override** is a custom name assigned to a blendshape in the BlendshapeMetadata Scriptable Object. This name is used instead of the default blendshape name provided by the 3D model, making it easier to reference and manage blendshapes in scripts, especially when using the UB3 Code Bridge.

Processed/Unprocessed Blendshapes

- **Processed Blendshapes**: These are blendshapes that have been assigned a class, group, and subtype, making them easier to reference in scripting and UI tools.

- **Unprocessed Blendshapes**: Blendshapes that have not yet been categorized or assigned metadata. These blendshapes can be processed in bulk using tools like the **Batch Blendshape Editor**.

Back to Table of Contents