




Harmony Animation Toolkit Abridged



Version 1.0.0



Click  [Harmony Animation Toolkit Abridged](#) to view the full web version.



[Quick Help](#)

Harmony Animation Toolkit is a suite of Unity Editor tools designed to simplify and automate the 2D and 3D animation workflow. Whether you're converting animations to JSON, replacing Animator clips in bulk, or generating animation clips from sprite sheets, Harmony aims to streamline repetitive animation tasks so you can focus on creativity.

The toolkit consists of 5 different tools with different functions to make creation of Animations and Animators a lot easier

All tools are available via

[Tools > Harmony Anim Toolkit](#)

Toolkit Overview

Tool	Purpose
<u>Animator Controller Generator</u>	Generate Animator Controllers from structured transition data. Best used with the JSON Importer.
<u>Animation JSON Importer</u>	Convert animation definitions in JSON to working Animator assets. Supports custom format handlers.
<u>Animation Asset Finder</u>	View all animation clips in a folder, including those inside FBX/Mesh assets.
<u>Animator Converter</u>	Convert Animator Controllers to JSON or bulk-replace animations within them.
<u>Sprite Animation Generator</u>	Create animations from sprite sheets or sliced sprites in just a few clicks.

Quick Overview

Animator Controller Generator

The

Animation Controller Generator is a Unity Editor tool that helps users easily create and manage animation states and transitions within an Animator Controller

This tool is especially useful when working with the Animator Converter, such as when converting JSON data into a working Animator Controller. It also comes in handy when you have a predefined setup or template and need to generate several similar Animator Controllers with consistent settings. Currently, the tool focuses on generating new Animators; full support for editing existing ones is not yet available.

How to Use

1. Add a Transition

Click the **Add Transition** button to create a new transition between two animations (either of which can also be a Blend Tree).

2. Add Conditions

Add one or more conditions for the transition. Parameters will be automatically created based on the condition names.

3. Set Animations

For each transition, you can either assign an animation clip directly or type the name of the clip. If you enter a name, the system will try to find the animation using the default logic (or your custom logic if a **Custom Finder Script** is set. See [Custom Finder](#) for more info).

4. Configure Blend Trees (if used)

If using Blend Trees, set the blend type and parameter name(s) required.

5. Repeat as Needed

Add as many transitions as needed. Animations or Blend Trees used in multiple transitions will not be duplicated.

6. Save Settings (Optional)

Click **Generate Transition SO** to save the current setup as a Scriptable Object for reuse later.

7. Generate the Animator

Click **Generate** to create the Animator Controller. It will automatically be assigned to the specified **Target Object**.

Parameters

Parameter	Description
Target Object	The GameObject that will receive the generated Animator Controller.
Animation Folder	Folder to search for animation clips. Use Assets/ to search the entire project.
Transition Data	A ScriptableObject used to load and save the editor's current settings.
Custom Finder Script	[Optional] A MonoBehaviour script that implements IAnimationFinder , allowing custom logic for locating animation

	clips. See Custom Finder for more information. Uses default logic if left empty.
Animator Controller	[Beta] An existing Animator Controller to pull transition data from. This feature is experimental and may not be fully stable.

Animation JSON Importer

The **Animation JSON Importer** is a Unity Editor tool that turns JSON files into fully working Animator Controllers.

I built this to speed up Animator creation—especially during game jams and when using **Mixamo animations**. I usually pair it with **Large Language Models (LLMs)** to quickly generate the JSON I need. I’ve shared a sample JSON file (for working with Mixamo 3D animations) and the prompt I use with LLMs on my YouTube channel:

 <https://www.youtube.com/@jeffawedev>

The tool uses a default JSON format (schema), but you can also use your **own custom JSON format** if needed. To do that, you just need to write your own importer script using the [JSONConverter](#) interface. See the [Custom JSON Importer section](#) for more details.

How to Use

1. Add a JSON File

Load your JSON by either dragging in a [TextAsset](#) or selecting a file from your system.

2. Add Custom Scripts (Optional)

You can plug in custom scripts to change how animations are found or how JSON is read. See **Custom Finder** and **Custom JSON Importer** below.

3. Validate JSON

Click **Validate JSON** to check if your file matches the expected format. (Note: This only works with the default importer. If you’re using a custom one, validation won’t apply.)

4. Save Your Setup (Optional)

Click **Create SO** to save your current setup as a ScriptableObject, so you can reuse it later.

5. Generate the Animator

Click **Generate Animator** to open the Animator Controller Generator. From there, you can customize your Animator and finish the setup.

Parameters

Parameter	Description
<code>JSON TextAsset</code> or <code>JSON FilePath</code>	Load your JSON file as a Unity TextAsset or pick a file from your computer.
<code>Animation Folder</code>	Folder to search for animation clips. Use <code>Assets/</code> to search the whole project.
<code>Custom Finder Script</code>	[Optional] A <code>MonoBehaviour</code> script that implements <code>IAnimationFinder</code> , allowing custom logic for locating animation clips. See Custom Finder for more information. Uses default logic if left empty.
<code>Custom JSON Importer</code>	[Optional] A script that uses <code>IJSONConverter</code> to define your own way to read and convert JSON. See Custom JSON Importer for more information. Leave empty to use the default.
<code>Animator Controller</code>	[Beta] Load an existing Animator Controller to pull transition data from. This feature is still experimental.

Animation Asset Finder

The **Animation Asset Finder** is a simple Unity Editor tool that helps you quickly find all animation files in a selected folder. It lists out the names of all the animations it finds—including animations inside mesh files like FBX.

This is useful when you're working with a large number of animation assets and want a quick overview.

How to Use


1. Click "Select Folder"

Pick the folder where you want to search for animations.

2. View Results

The tool will list all the animation clip names found in that folder, including those inside mesh files.


Parameters

Parameter	Description
	Opens a file picker to choose the folder where the tool will search for animations.

Animator Converter

The **Animator Converter** is a Unity Editor tool that lets you do two main things:

1. **Convert an Animator into JSON** – great for backing up Animator structure or using with other tools.
2. **Replace animations inside an Animator** – super useful when reusing the same Animator setup with different animations (e.g., replacing Mixamo animations).

 Currently, it only supports the default JSON schema. Custom schema support will be added in future updates.

You can also **filter** both the existing and replacement animations using a simple text match, and there are options to **auto-match** by name and prevent replacing animations with the exact same ones.

How to Use

1. Select an Animator

Pick the Animator Controller you want to convert or edit.

2. Select a Folder

Choose the folder that contains the animation clips to replace the existing ones in the Animator.

3. Convert to JSON (Optional)

If you just want to convert the Animator to JSON, click the **Convert to JSON** button.

4. Set Filters (Optional)

Use **Source Filter** and **Destination Filter** to narrow down which animations are shown and matched.

5. Scan Animations

Click **Scan Animations** to list all animation clips currently in the Animator. It will show dropdowns next to each, allowing you to choose replacements.

6. Replace Animations

Once you've selected replacements, click **Replace Animations**. This will swap out the animations in the Animator with the selected ones.

Parameters

Parameter	Description
Animator Controller	The Animator to convert or edit.
Animation Folder	Folder to search for replacement animations
Source Filter	A string to filter the source animations (e.g., typing "run" finds animations starting with "run").
Destination Filter	A string to filter replacement animations.
Auto Match Similar Names	If enabled, it will try to auto-match animations that have similar names.
Safety Check	If enabled, it prevents replacing an animation with one that has the exact same name.

Sprite Animation Generator

The **Sprite Animation Generator** is a Unity tool that helps you quickly create 2D animations either from a sliced sprite sheet or from individual sprites. It supports two workflows:

- **Sprite Sheet:** You define the start index and number of frames from a pre-cut sheet.
- **Individual Sprites:** Drag and drop sprites or use all the sprites found in a folder.

This is especially useful for prototyping, game jams, or just speeding up 2D animation setup in your project.

How To Use

- Choose the **Source Type**: Sprite Sheet or Individual Files.
- If using a **Sprite Sheet**, assign a sliced sprite sheet to the Sprite Sheet field.
- Set the **Sprite Order** (if using a sprite sheet). Default and recommended: *Numeric Suffix*.
- If you have an existing settings ScriptableObject, load it into the Settings field.
- Set the **Save Path** for where the generated animations should go.
- Click **Add Animation** to create an animation entry. The input fields will change based on your selected source type.

For Sprite Sheets:

- Set the animation name, starting index on the sheet, how many frames it should use, and the frame duration (in seconds or FPS).
- Click **Preview Animation** to see a list of the sprite names used in the animation.

For Individual Files:

- Enter the animation name and frame duration.
- Under **Specific Source**, either:
 - **Drag and drop** your sprites and manually sort them, or
 - Click **Use Folder** to automatically load all sprites in a selected folder (note: this uses all sprites found and doesn't allow for custom ordering).

Click **Generate Animations** to create the Animation Clips, or **Save Animations as SO** to save your setup as a reusable ScriptableObject.

Parameters

Parameter	Description
Source Type	Choose between Sprite Sheet or Individual Files as the source for your animation.
Sprite Sheet	A cut sprite sheet to extract frames from. Required if Source Type is set to Sprite Sheet.

Sprite Order	Method for ordering sprites. Recommended: Numeric Suffix (e.g., run_0, run_1, run_2...).
Settings SO	[Optional] Load an existing ScriptableObject to auto-fill setup parameters.
Save Path	The folder path where generated animations will be saved.
Add Animation	Adds a new animation entry to configure.
Animation Name	The name of the animation you want to create.
Start Index	[Sprite Sheet only] Index to start from in the sprite sheet.
Frame Count	[Sprite Sheet only] How many frames this animation has.
Frame Duration	Time between each frame in seconds (e.g., 0.1 for 10 FPS).
Preview Animation	View the list of sprites that will be used in the animation.
Specific Source	[Individual Files only] Choose between drag-and-drop sprites or a folder.
Use Folder	[Individual Files only] Automatically loads all sprites from the selected folder (order matters, no reordering supported).
Generate Animations	Creates the final <code>.anim</code> files based on your configuration.
Save Animations as SO	Saves the configuration into a ScriptableObject for reuse.

Customization

Custom Finder

The **Custom Finder** feature allows you to define your own logic for locating animation clips by name. This is especially useful when animations follow complex naming conventions, live in nested structures, or need specific filtering logic that the default search doesn't cover.

How It Works

In tools that support this feature, you can assign a **MonoBehaviour script** that implements a custom animation-finding strategy. If provided, the system will automatically call your script instead of using the built-in search logic.

Your script must implement the following interface:

```
public interface IAnimationFinder
{
    AnimationClip FindAnimation(string name, string path);
}
```

Once assigned, your logic will be called with the animation's name and the folder path, giving you full control over how and where to look for matching clips.

If no custom script is set, the tool will fall back to the default search method (by filename in standalone `.anim` or model files in the specified folder). The default finder function is below

```
private AnimationClip FindAnimationByName(string animationName, string
animationFolderPath)
{
    if (string.IsNullOrEmpty(animationFolderPath))
    {
        Debug.LogError("FindAnimationByName: Animation folder path is null
or empty.");
        return null;
    }

    if (string.IsNullOrEmpty(animationName))
    {
        Debug.LogError("FindAnimationByName: Animation name is null or em
pty.");
        return null;
    }

    if (!Directory.Exists(animationFolderPath))
    {
        Debug.LogError($"FindAnimationByName: The specified folder does n
ot exist: {animationFolderPath}");
        return null;
    }

    // Custom Finder Script Check
```

```

    if (customFinderScript is IAnimationFinder finder)
    {
        return finder.FindAnimation(animationName, animationFolderPath);
    }

    // Normalize folder path to Unity AssetDatabase format
    string relativePath = animationFolderPath.Replace(Application.dataPath,
    "Assets");

    // Search for standalone animation clips in the folder
    string[] animGuids = AssetDatabase.FindAssets($"t:AnimationClip {animationName}", new[] { relativePath });
    foreach (string guid in animGuids)
    {
        string path = AssetDatabase.GUIDToAssetPath(guid);
        AnimationClip clip = AssetDatabase.LoadAssetAtPath<AnimationClip>(path);
        if (clip != null && !clip.name.Contains("__preview__") && clip.name.Equals(animationName, StringComparison.OrdinalIgnoreCase))
        {
            //Debug.Log($"FindAnimationByName: Found animation clip '{animationName}' at {path}");
            return clip;
        }
    }

    // Search for model files in the specified folder
    string[] modelGuids = AssetDatabase.FindAssets("t:Model", new[] { relativePath });
    foreach (string guid in modelGuids)
    {
        string modelPath = AssetDatabase.GUIDToAssetPath(guid);
        UnityEngine.Object[] assets = AssetDatabase.LoadAllAssetsAtPath(modelPath);

        foreach (UnityEngine.Object asset in assets)
        {
            AnimationClip clip = asset as AnimationClip;

```

```

        if (clip != null && !clip.name.Contains("__preview__"))
        {
            if (clip.name.Equals(animationName, StringComparison.OrdinalIgnoreCase))
            {
                //Debug.Log($"FindAnimationByName: Found animation '{animationName}' inside model file {modelPath}");
                return clip;
            }
        }
    }
}

Debug.LogWarning($"FindAnimationByName: Animation clip '{animationName}' not found in folder: {animationFolderPath}");
return null;
}

```

Custom JSON Importer

The **Custom JSON Importer** lets you define your own logic for parsing animation data from JSON. While the toolkit includes a built-in parser that expects a specific schema (Schema classes are defined in the JSON Classes region of the Animator Converter Types script), this feature gives you full control over how that JSON is interpreted.

This is particularly useful if:

- You're working with external tools that generate different JSON formats.
- You want to handle additional metadata.
- You want to transform or filter data before it's converted into Unity-friendly structures.

How It Works

If a tool supports it, you can provide a **MonoBehaviour** that implements the `IJSONConverter` interface. When assigned, the system will automatically defer JSON parsing to your custom script.

```
public interface IJSONConverter
{
    List<AnimationTransition> ConvertJSONToAnimationTransitions(string jsonText);
    AnimatorConverterSO ConvertJsonToSO(string jsonText);
}
```

If no custom script is provided, the system falls back to the default parser. The default script for JSON parsing can be found in the Transition JSON Importer scripts under [lines 369 - 535](#)

You will also notice your implementation still has to return a List of Animation Transition which is the schema used by the toolkit. (You can find the class in the Animator Converter Types script).

? Quick Help

This documentation is hosted on **Notion**, so feel free to use **Notion AI** or the **search bar** to quickly find any tool, setting, or explanation across the page.

- Looking for what each tool does at a glance? Check out the **Quick Overview** section at the top.
- Want to understand how the scripts work behind the scenes? Scroll to the bottom and explore the **Full Workflow** section — it includes detailed notes on all core scripts and function definitions used in the toolkit.

- Most tools follow the same format:

➤ **Intro** → **How to Use** → **Parameters**

So it's easy to skim through if you're in a rush.

- Many fields are optional — don't worry if you leave them blank. The system uses fallback/default behavior when available.
- Most buttons and fields in the editor have helpful tooltips. Hover over them to get extra context.

If you're still unsure about something, check the **YouTube channel** for video guides:

👉 youtube.com/@jeffawedev or send me a mail at awagujeffery@gmail.com

I hope you enjoy using this toolkit! It's been a huge help in my own game dev journey, and I built it with the hope that it'll make yours smoother too.

If you found it useful, consider giving it a like and leaving a review on the Unity Asset Store — it really helps and means a lot! 🙏🎮