Introduction

Thank you for purchasing LipSync Pro, the powerful facial animation tool from Rogo Digital. With LipSync Pro, you can create retargetable lip-syncing animations, including emotions and gestures, with full control over the visual appearance and end-results.
This document will provide an overview of the features and tools provided by LipSync Pro.

If you have any further problems or questions not explained in this guide, check our website for the most up-to-date information and downloads: **http://lipsync.rogodigital.com.**

You can find a current API reference here.

Contents

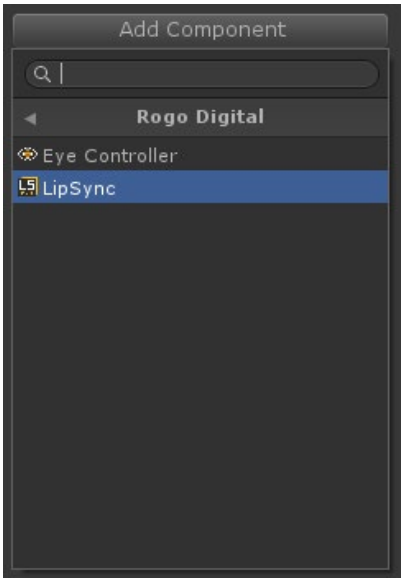
Getting Started

This is a basic guide on how to start using LipSync Pro for the first time. More detailed information and alternative workflows are detailed later on in the manual.

First, make sure you have a compatible character, by default this requires a SkinnedMeshRenderer with blend shapes, but other systems exist. Characters using LipSync Pro can be made into prefabs after setup if you wish, but setup is much simpler if the character exists in the scene.

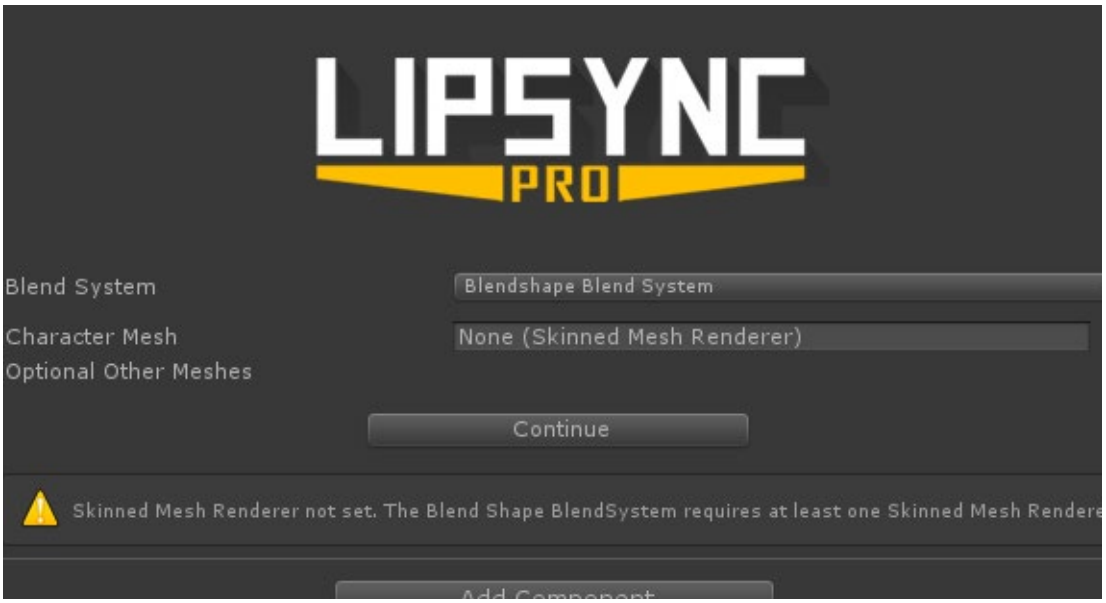
NOTE: AutoSync requires the 2015 Visual C++ Redistributable to be installed. A .zip file with both 32 and 64 bit versions is available from the extensions window or the downloads page. Please ensure you have the correct version installed.

Step one. Select your character's root GameObject and add the LipSync component from the AddComponent menu.



Step two. Select a Blend System (see BlendSystems) from the dropdown at the top of the LipSync component. By default, only the Blenshape and Sprite blend systems will be available. Others can be downloaded (see Extensions) or created yourself (see Creating Blend Systems).

Step three. Fill in the fields required by the Blend System, then press the Continue button. Using the Blendshape Blend System, this is the Main Skinned Mesh Renderer, and any optional additional renderers. These can be changed at a later point.

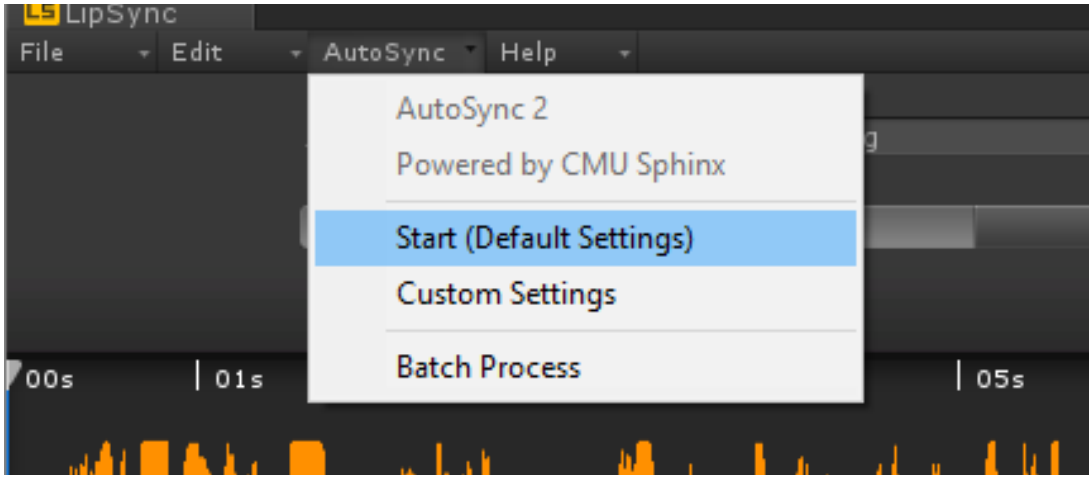


Step four. Create the phoneme poses by clicking on a phoneme, adding blendables and setting a value for each (See Blendables and The Pose Editor). You can alternatively use a preset instead by clicking the Presets button (next to the phonemes and emotions tab) and selecting one from the drop-down.

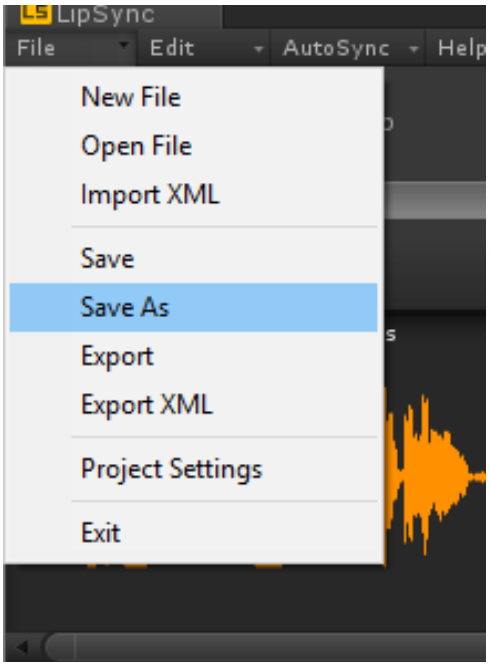
Note: LipSync comes with presets for Adobe Fuse characters only. You can create your own to work with other characters (see Presets).

Step five. Open the Clip Editor by selecting an AudioClip you want to lip-sync in the Project window and clicking Window > Rogo Digital > LipSYnc Pro > Open Clip Editor in the top menu, or using the keyboard shortcut Ctrl + Alt + A.

Step six. Add phoneme markers to the file using the editor (See The Clip Editor). You can simplify this process by using AutoSync - just click the AutoSync button in the Clip Editor toolbar and select *Start (Default Settings)* or *Custom Settings* for more advanced options.



Step seven. Save the file by clicking File > Save As in the Clip Editor toolbar, and selecting a location within your project to save it. You can alternatively export to an XML file, though for most uses, the standard LipSyncData file is recommended.



You're Done! You can now play the saved LipSyncData file using the LipSync component on your character, either by using the play on awake setting or by calling LipSync.Play from a script.

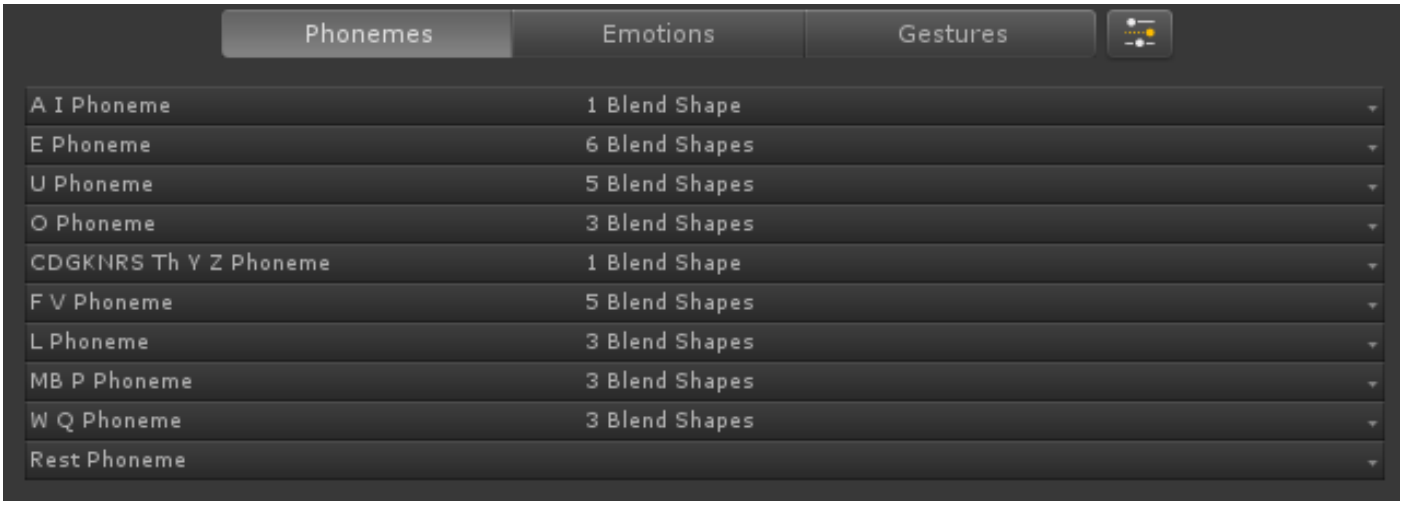
Editors

LipSync Pro is made up of two parts, a component that includes an editor for defining phoneme (viseme) and emotion poses and gesture animations, and a clip editor, that is used for syncing phonemes and emotions to an audio clip.

The Pose Editor

The Pose Editor is part of the inspector for the LipSync component. It's here that you setup the per-character poses for each phoneme and emotion in your project.

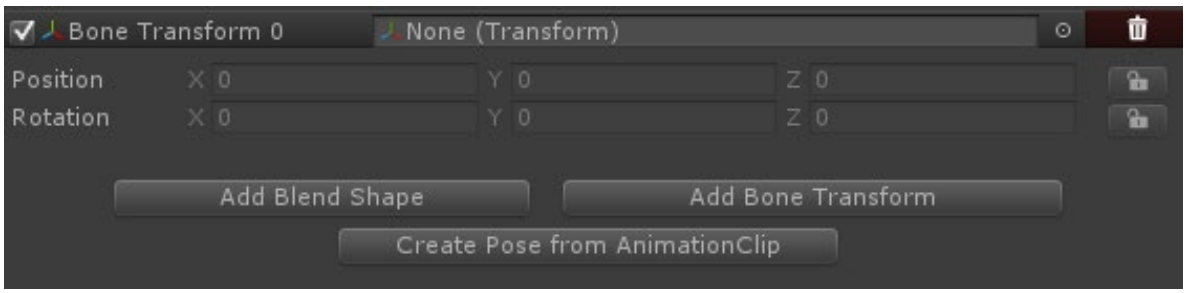
The main area of the Pose Editor is an accordion-style list of phonemes (or emotions, depending on the current tab). Clicking one of these will expand the editor for that pose.



Within this panel, you have the choice to add blendables or bones (if bones are enabled) to the current phoneme/emotion pose, and a list of all the blendables and values that make up the pose.

Each row in this list contains a dropdown (1) to choose the blendable from a list, a delete button (2) to remove that blendable from the pose, and a slider (3) to set the value/weight of that blendable in the pose.

When you add bones, the dropdown is replaced by an object picker and the value slider is replaced by position and rotation vector fields. Use these to chose and pose the bone transforms instead. You may also notice the lock buttons on the right. Locking either the position or rotation for a pose will prevent that property from being affected by the pose. This can be useful if you want ONLY the position from the pose, or vice-versa. Enabling the checkbox next to the bone transform will show position/rotation handles in the scene view for that transform.

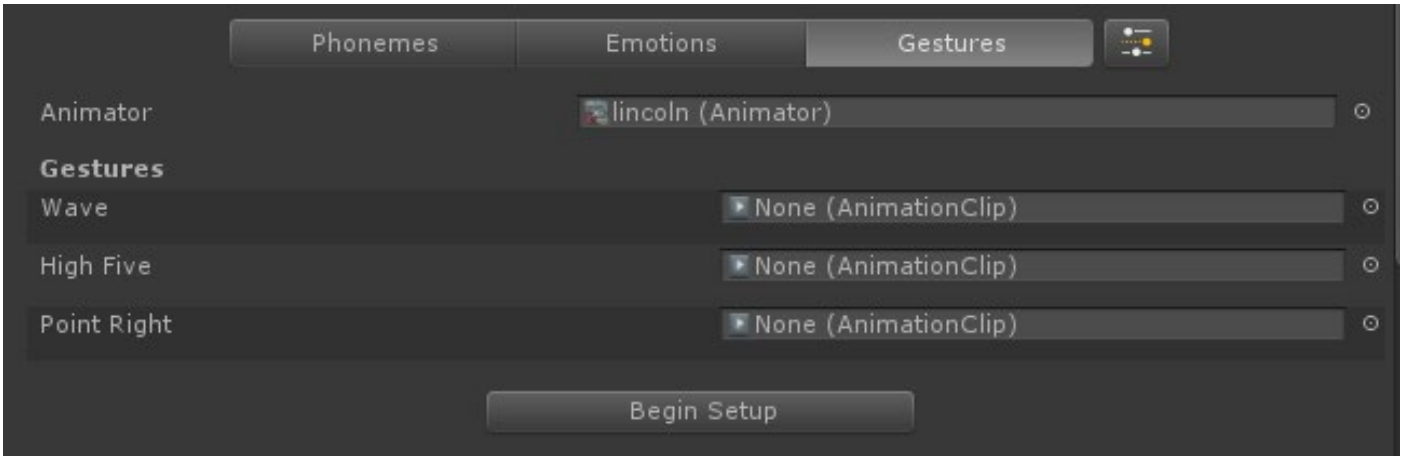


As you add/remove blendables and move the sliders, the changes you make will be reflected on the character in the scene view.
When editing phoneme poses, an illustration of that phoneme is displayed in the lower-right hand corner of the scene view. You can use this as a rough guide to pose your character's face exactly how you want for each phoneme.

Gestures Setup

When there are gestures defined (see Project Settings), you can run the Gesture Setup Wizard from the Gestures tab on the LipSync component. This will prepare the Animator Controller of a Mecanim Animator component with the necessary states, transitions and triggers to play back gesture markers.

After clicking onto the Gestures tab you will see a field to pick an Animator component. You can use the same one you're currently using for animations on this character, as LipSync will use its own layer.
Once the animator is defined, you can choose which animation clips you want to assign to each Gesture defined in the Project Settings.



With clips assigned to each Gesture, you can click Begin Setup to start the Gesture Setup Wizard. This will create or set up an animation layer, and prepare it with everything LipSync needs to play the Gesture animations.

If any characters haven't been set up like this, or you don't assign an AnimationClip to any particular gesture, the rest of LipSync will continue to function, ignoring anything it can't play.

Additionally, the wizard will need to be run again when a new Gesture is added to the project for each character you want to support it. The editor will inform you when this is the case.

The Clip Editor

The Clip Editor is a separate editor window that can be moved and docked like the other Unity editor windows. It can be opened by going to the Window > Rogo Digital > LipSync Pro menu at the top of the Unity editor, and selecting Open Clip Editor. You can also use the keyboard shortcut Ctrl + Alt + A. This action is context sensitive: if an AudioClip is selected in the Project window, it will be set as the clip for the new animation. If a LipSyncData asset is selected, it will be opened in the editor.

This editor is used for synchronising phonemes, emotions and gestures to audio clips, either manually or using AutoSync, and creates the LipSyncData asset files that your characters will play back. The next page shows an overview of the editor with an audio clip selected.

1. The name of the asset being edited.
2. Settings button. Opens the settings panel.
3. Preview button. Turns on real-time previews.
4. Linked audio clip. Changing this will cause the waveform to update.
5. Timeline. Phoneme or emotion markers will appear here.
6. Time Ruler and Waveform. Gives a preview of the audio at a specific time.
7. Zoom/pan Control. Used to zoom in on a certain part of the clip.
8. Play/Pause/Add Phoneme buttons.
9. Phoneme Filter. Limits which markers are shown in the timeline.

There are two main ways of syncing phonemes to your audio: manually, by adding phoneme markers along the timeline, or automatically using AutoSync. Emotion markers must be added manually.
You can always go back and manually add, edit or remove markers, especially if you used AutoSync to set it up originally, as AutoSync rarely produces perfect results.

AutoSync

AutoSync is the quickest way to sync phonemes to your audio, though it isn't completely accurate, and is intended more as an easy way to get a rough draft to refine later. It is, however, a good way to start, especially if you have a large number of clips that need producing. **It can only process uncompressed 16-bit, 1600 kHz mono .wav files. This limitation can be avoided by following the set-up guide for SoX Sound Exchange, which AutoSync can use to temporarily convert any Unity-compatible audio file.**

To use AutoSync, simply click the AutoSync menu in the top toolbar of the Clip Editor, and select *Start (Default Settings)*, *Custom Settings* or *Batch Process*.

Selecting ***Start (Default Settings)*** will instantly start the AutoSync process, using the currently chosen audio clip, the default Language Model (as defined in the Clip Editor's settings screen) and will use audio conversion if available.

Selecting ***Custom Settings*** will first bring up the AutoSync settings tab of the AutoSync window, where you can manually select a Language Model and turn audio conversion on or off, just for this clip.

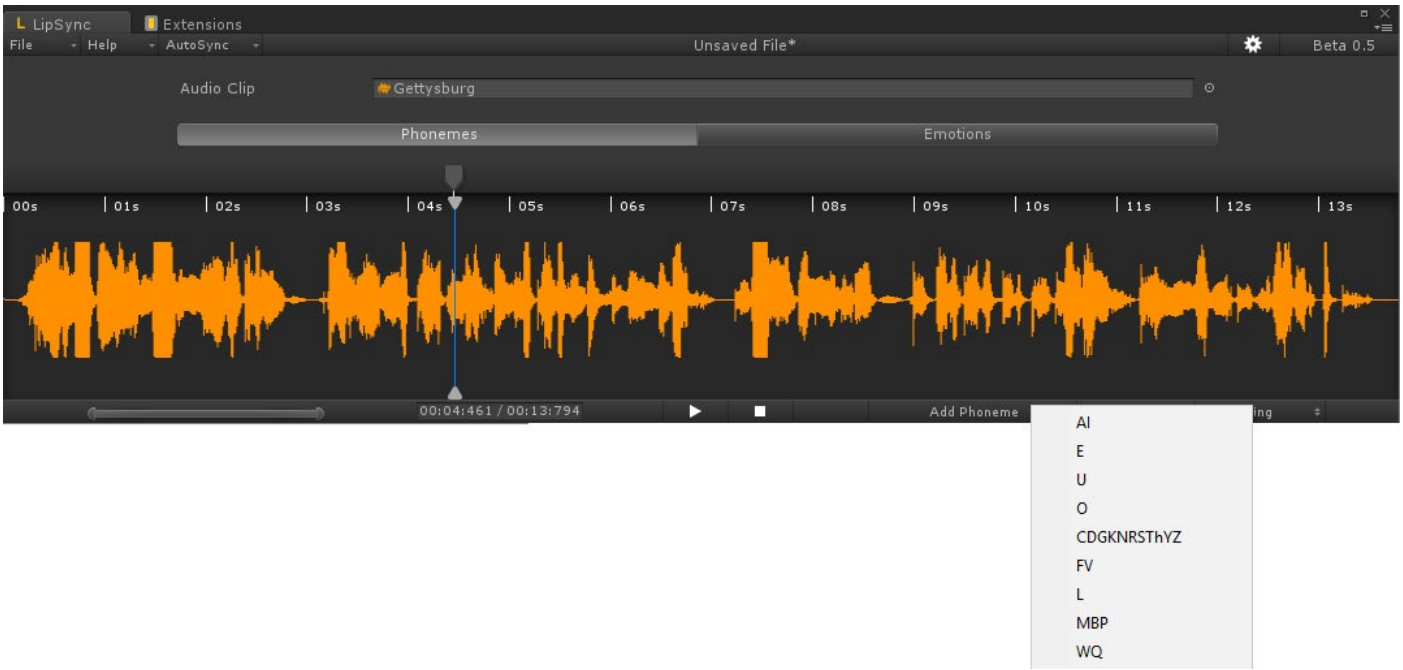
Selecting ***Batch Process*** opens the Batch Process tab of the AutoSync window, where you can add any number of AudioClips, which will be processed one after the other according to the settings on the AutoSync Settings tab.

Manually Adding Phonemes

Manually adding your phoneme markers can give you much more control over the final animation, and in fact was the only option available until beta 0.3.

Clicking and dragging anywhere within the Time ruler/waveform area will move the playhead along and scrub through the audioclip. If you click once and release, a short snippet of the audio will be played. This is useful for finding exactly where to add a certain phoneme.

Place the playhead where a phoneme should be triggered and click the “Add Phoneme” button on the lower toolbar. This will then show you a list of phonemes to add. Pick the right one for the sound you hear and a new marker will appear on the timeline. You can also right-click the waveform to add a marker at the point you clicked.



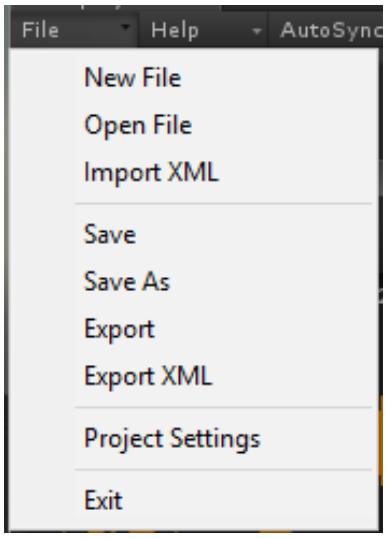
Go through the clip adding phoneme markers until complete. You can zoom in and scroll along the timeline by using the zoom/pan control in the lower left.

Dragging one side of the handle resizes it, so that the length and position of the handle represents the viewport in relation to the clip's length.

You can delete a phoneme marker, or change the phoneme it represents after placing it by right-clicking, and either clicking delete or choosing the new phoneme from the context menu.

Saving/Loading

Regardless of which method you use, once you've finished syncing a clip you will need to save it in order to play it back.
The File menu in the Clip Editor's top toolbar has a number of different options you can use for saving out your clips.



The standard *Save* and *Save As* options will create a .Asset file in our LipSyncData format. These files can be used as the default clip in a LipSync component, or passed as a parameter to the LipSync.Play or LipSync.PlayFromTime methods.
This is the recommended format for saving data to be played in, as it contains a reference to the linked audio clip.

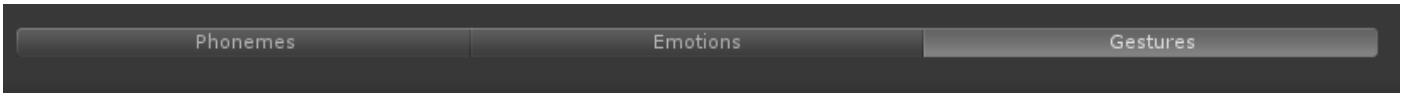
There are, however, two export options you can also use.

Export XML will create an XML document representing the phoneme and emotion marker data. This XML file can then be loaded back in to the editor using the *Import XML* option, or played back using LipSync.Play. This format can be useful for processing the data outside of Unity, as the XML format is very widely supported and easy to work with. It doesn't have any way to reference the audio clip however, so this must be passed to the Play method separately.

The final option is simply *Export.* This will create a .unitypackage file containing the data as a LipSyncData file, along with a copy of the audio clip. This can be used for easily transferring a file between two computers.

Gesture Markers

Gesture markers work in much the same way as phoneme markers, except they represent triggers for seperate Mecanim animations to start playing.
To add gesture markers to your animation, you first need to switch to the gestures tab in the Clip Editor.

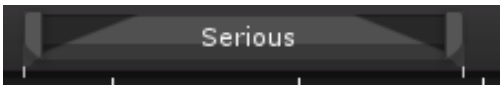


The gestures that are available to choose from the Add Gesture list are defined in the project settings file (see Project Settings). Although any gesture can be added to any clip, they will only be played back if the character has been correctly set up with animations (see Gestures Setup).

Emotion Markers

The final type of marker in LipSync is the emotion marker. These represent facial expressions that your characters can make while speaking. This helps add life to your dialogue.

An emotion marker is made up of 5 components: a start handle, a bar representing the length of the marker, an end handle and two blend handles.



Dragging the start or end handles around lets you resize the marker, and dragging the main body of the marker will move it along the timeline.

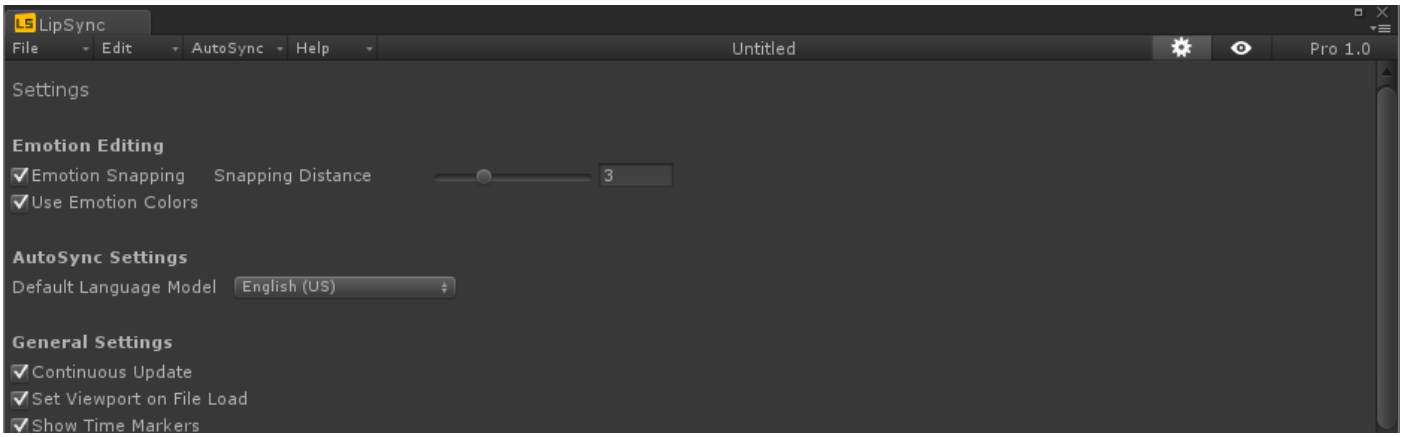
Emotions, unlike phonemes, don't use a fixed blend time from the LipSync character playing them. Instead, they each have their own blend in and blend out times. These are represented by the darker areas on either side of the marker. When an emotion marker is on its own on the timeline, you can set the blend in or out times by dragging the appropriate blend handle towards the centre of the marker. The blend handle will appear when you hover over the edge of the darker blend area.

Emotion markers are also able to blend directly into another marker, instead of blending back to neutral. To do this, you can simply drag a marker so that it overlaps with another. The dark area will change colour to match that of the other marker, and the adjacent handles on the two markers will disappear. Once this has happened, you can adjust the blend times by dragging either marker further.

The right-click menu for an emotion marker functions similarly to that of a phoneme marker, letting you delete or change the emotion. Just like with Gestures, the emotions that are available to you are defined in the project settings file (see Project Settings).

Settings

Clicking the settings icon in the top toolbar will toggle between the standard Clip Editor screen and the settings page. Many of these settings are user preference for how the Clip Editor should behave. They are as follows:



Emotion Snapping and **Snapping Distance**

When Emotion Snapping is enabled, dragging an emotion marker close enough to another will make it snap onto the other marker. Snapping Distance controls how close the markers need to be for this to happen.

Use Emotion Colors and **Default Color**

If Use Emotion Colors is enabled, emotion markers will use the color assigned to that emotion in the project settings file (see Project Settings). Otherwise, they will all be the same color, which is defined by the Default Color setting.

Default Language Model

The language model to be used by AutoSync when default settings are used. Language Models allow support for multiple languages.

Continuous Update

If enabled, this will make the Clip Editor redraw every frame. This makes it more responsive while editing markers, but could be slow to use on a low-powered machine.

Set Viewport on File Load

This will make the editor display the entire clip in the viewport when a new file is loaded, instead of staying at the same zoom level as the last file.

Show Time Markers

Shows or hides the time ruler underneath the timeline.

Scrubbing Preview Length

The length of time, in seconds, that will be played when the playhead is moved.

Preview Volume

The volume that audio from the Clip Editor will play at.

Max Waveform Width

This is the maximum width of the waveform preview image. The preview is regenerated with a larger width when you zoom in on the timeline, to preserve detail. At extreme zoom levels though, this can cause the image to become unacceptably large, so Max Waveform Width is used to clamp it. Setting this lower can make the editor use less memory, while setting it higher will allow the

preview to become more detailed. **Warning**: setting this value too high can cause the editor to crash when zooming in.

Show Extension Window
If enabled, a Rogo Digital Extensions window (see Extensions) will be docked next to the Clip Editor for easy access when the editor is opened.

Concepts

This section explains some of the terms and concepts in LipSync, and how they are used.

Poses (Shapes)

Poses (known as shapes in scripts for legacy reasons) define a set of blendables and bones, and set values for them that create a specific facial pose. There are two types of shape/pose in LipSync Pro, PhonemeShapes and EmotionShapes. As you'd expect, these perform more or less the same function - except PhonemeShapes contain a LipSync.Phoneme variable to identify them, and EmotionShapes have a string that refers to an emotion set up in the Project Settings. The number of PhonemeShapes are also limited to the number of Phonemes, 10.
They are used primarily by the LipSync component, and by LipSyncPresets.

API Reference: Shape, PhonemeShape, EmotionShape.

BlendSystems

Blend Systems are a core feature of LipSync Pro post 0.5. Whereas LipSync itself works with generic concepts like blendables and shapes, Blend Systems deal with actually updating or modifying the character to create the animation.
The BlendSystem class is a base that all Blend Systems inherit from, which defines the basic structure and methods that a Blend System needs to work.

This allows the LipSync component to function the same regardless of whether the character uses a SkinnedMeshRenderer with blend shapes or some kind of 3rd party character system. BlendSystems are actually a seperate component attached to the same GameObject as LipSync, but they are hidden from the inspector and managed by the LipSync component instead of being added manually.
LipSync comes with the BlendshapeBlendSystem and the SpriteBlendSystem included. Others are available to download (see Extensions) or you can write your own (see Creating Blend Systems).

API Reference: BlendSystem, BlendshapeBlendSystem, SpriteSwapBlendSystem.

Blendables

Blend systems can work in very disparate ways. Because of this, LipSync uses the concept of 'blendables' to refer to any type or object that controls a character's physical appearance, and where that amount of control can be transitioned between none (0) and full (100). An example of a blendable would be a blend shape in the BlendshapeBlendSystem class.

The concept is used in BlendSystem methods such as BlendSystem .SetBlendableValue, which is passed an integer index to reference a certain blendable, and a float value to set the blendable's influence on the character.

The Blendable class in LipSync is used internally by the base BlendSystem class to keep track of the current 'correct' value for a blendable and to prevent issues with other systems changing the underlying values.

API Reference: Blendable, BlendSystem.

Presets

LipSyncPresets are a kind of serialised (saved to disk) asset in LipSync that store metadata about the phoneme and emotion shapes from a LipSync component. They are used to create easily accessible setups for characters that can be added to any LipSync component in a couple of clicks.

Presets must be placed in a folder called "Presets", or an immediate subfolder of one, to allow LipSync to find them. They can be loaded or saved from the LipSync component inspector by clicking the presets button next to the Phonemes/ Emotions tab to open a dropdown list. They are sorted in this list by subfolder name, so this can be used for categorisation.

Note: From version 1.0 onwards, the LipSyncPreset format has been used, which contains more information for identifying blendables, and also has support for bone transforms. The old BlendshapePreset format still exists for legacy purposes, but LipSync can no-longer use them and they must be updated.

API Reference: LipSyncPreset

Project Settings

Each project using LipSync Pro should contain a file called ProjectSettings.Asset in the Rogo Digital/LipSync folder. There is one included with the package by default, and a new one will be generated automatically if it is missing.

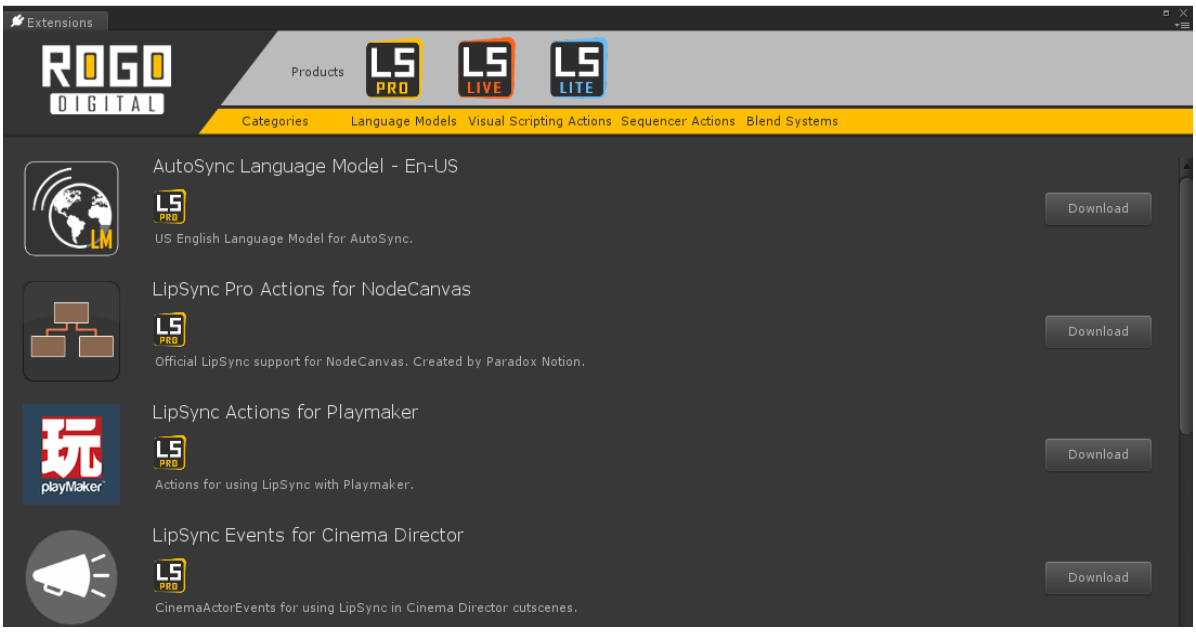
The Project Settings file contains the names and colours of emotions, and the names of gestures available to use in LipSync animations. It can be accessed either by finding and selecting it in the project view, or by clicking on Edit > Project Settings > LipSync.

Extending LipSync

Extensions were introduced in beta 0.5 as a way to add support for third-party assets to LipSync without having to update the package with code that not all users will have a use for.

The Extensions Window

The extensions window can be opened from the Window > Rogo Digital menu, or from the Help menu in the Clip Editor.



This window can be used to download officially endorsed extensions to LipSync Pro and other Rogo Digital assets when connected to the internet. Extensions can be filtered by compatible products, or category.
The extension will be downloaded as a standard .unitypackage which Unity will import into your project.

Creating Blend Systems

A key feature of blend systems is the ability for any developer to write their own, which LipSync will detect and use. This makes it possible, with a little work, to use LipSync with almost any rendering/animation system. There are already blend systems available for blendshapes, sprites, Morph3D characters and UMA 2 characters.

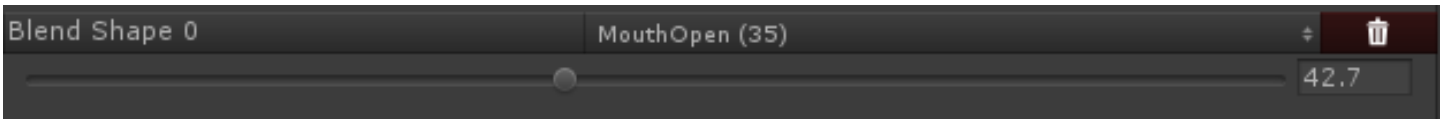
An empty blend system template can be created from the Create menu in Unity, or directly from the project window. You should first have a look at the BlendshapeBlendSystem.cs file as an example of how a working blend system can be set up.

The empty blend system template contains comments detailing what each method does. It is difficult to give exact instructions on creating a blend system, as each will be inherently unique, but the basic requirements are that they be public, not static, inherit from RogoDigital.Lipsync.BlendSystem, and contain the following methods:

```
public override void OnEnable ()
public override void SetBlendableValue (int blendable, float value)
public override string[] GetBlendables ()
public override void OnVariableChanged ()
```

When implemented correctly, this will allow your blend system to act as a bridge between LipSync and your chosen system.

Blend systems show blendables in a dropdown, and the user selects one when adding a blendable to a pose.



The “Reference Blend System” functionality contained in LipSync 0.5 is no longer supported, as it was difficult to use and encouraged blendSystems to be dependent on the LipSync component using them (which is unreliable, as other scripts can also make use of Blend Systems as of 0.6.)

The best replacement for this functionality is to convert your reference Blend System to a standard index-based one, and create your own separate component

that has a list of objects you want to use and reports them to the Blend System as blendables. This may seem messier at first, but it allows for much more advanced functionality than reference blend systems could. You can use the RequiresComponent attribute on your Blend System to automatically add your own manager script whenever that Blend System is used.

In addition to the functionality explained in the template's comments, you can create buttons that show up in the LipSync editor above the tab control. To do this, create a a public method in your blend system without a return value, then mark this method with the `[BlendSystemButton(string displayName)]` attribute.

This will create a button with the value of displayName as a label. The method itself will be called whenever the button is clicked. For example, this is the ToggleWireframe method from BlendshapeBlendSystem.cs:

```
[BlendSystemButton("Toggle Wireframe")]
public void ToggleWireframe () {
	wireframeVisible = !wireframeVisible;
	#if UNITY_EDITOR
	EditorUtility.SetSelectedWireframeHidden(characterMesh ,
!wireframeVisible);
	foreach(SkinnedMeshRenderer renderer in optionalOtherMeshes) {
		EditorUtility.SetSelectedWireframeHidden(renderer ,
!wireframeVisible);
	}
	#endif
}
```

As these buttons are used only in the editor, you may have to make use of some classes in the UnityEditor namespace. The problem here is that this namespace does not get packaged in with your project when it is built. To get around this, you can make use of the #if UNITY_EDITOR preprocessor directive, which will effectively hide that code from the compiler when compiling for anything other than the Unity Editor.