Koreographer Karaoke Demo Overview

for v1.6.1

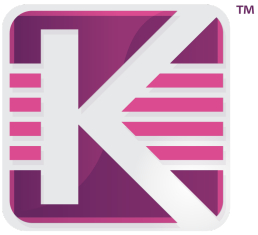

Table of Contents

Overview **3**

Karaoke System Assets Overview **3**

Karaoke System UI Structure **4**

Overview

The Karaoke Demo content for Koreographer is very simple. It shows how you can use Koreography to drive on-screen text highlighting. The demo content was developed in Unity 5.0 and should be forward compatible with newer Unity versions.

The Karaoke system uses two Unity UI Text layers and a Mask to achieve the "Paint" effect. Koreography is used to both:

1. Feed new line data to the Karaoke system at runtime and
2. Inform the system of paint timing.

While the control of the Mask "animation" used to paint the text is driven by Koreography, the line-feed animation uses Mecanim. More on this below.

Karaoke System Assets Overview

The Karaoke Demo contains the following assets.

- **Scripts**
  - **KaraokeController** - Registers for Koreography Events that drive Line Feed and painting. The line feed event (mapped to the "Lines" Event ID) is used to prepare the state of applicable *TextRow*s for animation and then trigger that animation. The painting event (mapped to the "Lyrics" Event ID) is used to control the state of the overlay text UI *Mask*s. Finally, this script is responsible for updating all texts when the *KaraokeLineFeed* animation completes via an *Animation Event* (see below).
  - **KaraokeTextLine** - Represents a single line of Text in the Karaoke UI. It manages the actual displayed text of both the backing UI *Text* and the overlay UI *Text*, as well as the UI *Mask* that controls text painting.
- **Prefabs**
  - **TextRow** - A single instance of a line of text for the Karaoke System. This is a prefab as all three lines used in the Demo scene are functionally identical, albeit *controlled* individually by the *KaraokeController* script.
- **Animations**
  - **KaraokeTextAnimController.controller** - The Mecanim *AnimationController* that contains the animation state machine for the line feed animation. The state machine simply switches between "animating" and "idling".
  - **KaraokeLineFeed.anim** - The *Animation* data describing how the three *TextRow*s will move during a line feed. The animation is configured to automatically reset the *TextRow*s to their starting position to prepare for the next line. The animation contains an *Animation Event* that notifies the system when it completes the animation to ensure that the contents of the *TextRow*s are updated at the same instant that the position reset happens. This is what keeps the visuals from appearing to "jump" on position reset.
- **Koreography**
  - **KaraokeKoreo** - The *Koreography* data. This links the *AudioClip* to the *KoreographyTrack*s.
  - **LinesTrack** - This *KoreographyTrack* is configured with the Event ID "Lines". It controls Line Feed events, instructing the Karaoke System to move on to the next line of text (and scroll, if

necessary). *KoreographyEvent*s in this track are OneOffs with *TextPayload*s. A special distinction is made between the first event and all other events:

- **First Event** - The *TextPayload* contains the *first two* lines of text, split by the substring "\n". An example of this would be: "*My first line,*\n*My second line,*". This special case allows for the text to be set without triggering the line feed animation.
- **All Other Events** - The *TextPayload* contains the next line of text to show and will trigger playback of the *KaraokeLineFeed* animation.

- **LyricsTrack** - This *KoreographyTrack* is configured with the Event ID "Lyrics". It controls the UI *Mask* size to produce the painting effect. *KoreographyEvent*s in this track are Spans with *TextPayload*s. Each frame in which a span event occurs, the Karaoke System finds the currently active line and hands it the *TextPayload* and the percentage of the way through the Span event at the current audio position. Using this information, the *KaraokeTextLine* script updates its mask position, causing the paint effect.

Karaoke System UI Structure

The Karaoke System is a UI system. The hierarchy is structured to use as few elements as possible to achieve the desired "text paint" effect while maintaining flexibility. The structure of these elements as found in the Demo Scene is outlined here:

- **Canvas** - All Unity UI must have a Canvas at the root. Everything beneath this can be moved into your own canvas. The only special setting is that the *UI Scale Mode* is set to *Scale With Screen Size*. This allows for varying screen sizes and aspect ratios without breaking the Karaoke Text Line settings (or requiring dynamic updates). See the *TextRow** elements below for more.
  - **KaraokeText** - This is the core of the Karaoke System. This element contains the *Mask* that cuts off text during a vertical scroll: it defines the area of the screen within which the text can be viewed (via a combination of *Image* and *Mask* components). The *Animator* component is used to drive the scroll animation of the texts. Finally, the custom *KaraokeController* is installed here, providing settings for *KoreographyTrack* links and the *TextRow** elements.
    - **TextRow*** - Each *TextRow** is an instance of the *TextRow* prefab. The *KaraokeTextLine* script is attached and configured here.
      - **Text Base** - The base text. This always contains the entire line of text.
      - **Text Mask** - A simple mask. The size of this mask is controlled by the *KaraokeTextLine* script and based on highlighting information from the Koreography (the *Lyrics* Event ID). Due to how the text glyph positions are calculated, the *RectTransform must* be set to absolute horizontal sizing, not relative-to-parent (no stretch).
        - **Text Overlay** - The overlay text. This always contains the entire line of text which is identical to that in *Text Base*. The color is distinct from the *Text Base* version. As configured, in order for this text to remain properly overlaid on the *Text Base* as well as independent from the parent *Text Mask* sizing, this object must be set to absolute horizontal sizing, not relative-to-parent (no stretch). This will keep the text from resizing as the *Text Mask* resizes. **Note:** In order to work properly, the width of the *RectTransform* on this object *must* match the width of the *Canvas*.