Dialog System

Documentation & reference

Index

Index
Changelog
    October 2015
    July 2015
    June 2015
Current/future plans
    Currently working on
    Planned for future versions
System overview
    Typewriter
        Placeholder content
        Message Log
    Graphics manager
        WARNING: Properly importing graphic files into your Unity projects
        Background and sprite management-related properties
        Text management-related properties
    Audio manager
        Differences between music and effect tracks
    Variable Manager
Editor tools
    Node editor
    Script editor
Script command reference
    Typewriter manager commands
        {\bold}, {\endBold}
        {\italic}, {\endItalic}
        {\color:RRGGBBAA}, {\endColor}
        {\size:<number>}, {\endSize}
        {\wait}
        {\endMessage}
        {\autoEndMessage}
        {\waitForSFX}
        {\waitForSFX:<trackNumber>}
        {\delay:<time>}
        {\waitForAllFlags}
        {\waitForFlag:<flagNumber>}
        {\raiseFlag:<flagNumber>|<flagDuration>}
        {\typingDelay:<delay>}
        {\typingSFX:<name>}

- {\typingPitch:<pitch>}
- {\typingVolume:<volume>}

Audio manager commands

- {\play:<name>|<type>|<trackNumber>}
- {\audioVolume:<type>|<trackNumber>|<volume>}
- {\audioPan:<type>|<trackNumber>|<pan>}
- {\audioPitch:<trackType>|<trackNumber>|<pitch>}
- {\audioVolumeTween:<trackNumber>|<loop>|<volume0>|<delay0>|...|<volumeN>|<delayN>}
- {\audioPanTween:<trackNumber>|<loop>|<pan0>|<delay0>|...|<panN>|<delayN>}
- {\audioPitchTween:<trackNumber>|<loop>|<pitch0>|<delay0>|...|<pitchN>|<delayN>}

Graphics manager commands

- {\changeGraphic:<name>|<type>|<layer>|<sorting>|<fadeDelay>}
- {\moveGraphic:<type>|<layer>|<positionX>|<positionY>}
- {\sortGraphic:<type>|<layer>|<position>}
- {\tweenGraphicsColor:<type>|<layer>|<loop>|<color0>|<delay0>|...|<colorN>|<delayN>}
- {\tweenGraphicsPosition:<type>|<layer>|<loop>|<positionX0>|<positionY0>|<delay0>|...|<positionXN>|<positionYN>|<delayN>}
- {\tweenGraphicsRotation:<type>|<layer>|<loop>|<rotation0>|<delay0>|...|<rotationN>|<delayN>}
- {\tweenGraphicsScale:<type>|<layer>|<loop>|<scaleX0>|<scaleY0>|<delay0>|...|<scaleXN>|<scaleYN>|<delayN>}
- {\tweenGraphicsFill:<type>|<layer>|<loop>|<fillClockwise>|<fillMethod>|<fillOrigin>|<fill0>|<delay0>|...|<fillN>|<delayN>}
- {\tweenGraphicsOffset:<layer>|<loop>|<offsetX0>|<offsetY0>|<delay0>|...|<offsetXN>|<offsetYN>|<delayN>}

Variable manager commands

- {\addVariable:<name>|<value>}
- {\changeVariable:<name>|<value>|<operation>}
- {\removeVariable:<name>}
- {\clearAllVariables}

Changelog

October 2015

- Finished coding deserialization for the Audio Manager sub-system
- Added flag-related typewriter commands:
    - WaitForAllFlags
    - WaitForFlag
    - RaiseFlag
- Audio manager
    - Removed the track type parameter from audio tweens since only music tracks are supposed to be able to tween
        - This change has affected text commands as well
    - Refactored method names for volume, pan and pitch tweens and their respective coroutines

July 2015

- Implemented Serialization/Deserialization for the following sub-systems:
    - Typewriter
    - Message log
    - Variable Manager
- Added Message Log class (only model-controller, no view implementation yet)
- Command tags
    - Tags can now be grouped
    - Added the following tags
        - AutoEndMessage
        - Audio volume, pan and pitch
        - Audio volume, pan and pitch tweens
    - Fixed graphics tweens not accepting graphics type and always defaulting to Backgrounds
- Audio Manager
    - Implemented volume, pan and pitch, as well as tweening for those properties

June 2015

- Message node editor

- Implemented drag support for multiple nodes at a time
    - All nodes
    - Parent nodes only
    - Children nodes only
- Implemented node sorting through breadth-first searches
    - To do: Implement exception when a node is the children of a shallow and a deep parent at the same time so that it gets placed near the deepest parent instead

Current/future plans

Currently working on

- Saving/Loading game state
- Message log visualization
- Merging basic commands, since it'd make more sense to have them grouped in the same tab as other commands for their subsystem

Planned for future versions

- Activating/Deactivating individual subsystems
- Replacing text for "wait" and "end of message" with icons
- Revising language system
- Inserting text from file into script instead of overwriting
- Options/Settings screen

System overview

Dialog System is made up by the following components:

- **Typewriter:** Loads the game script and manages all the other subsystems
- **Graphics Manager:** Manages text, backgrounds and sprites
- **Audio Manager:** Manages music and sound effect tracks
- **Variable Manager:** Allocates, reads and writes variables

Note: All text in red indicates elements that I've planned but still haven't programmed into the system.

Typewriter

The typewriter system is the one that reads the script file and contacts the other subsystems to handle everything else.

Placeholder content

- While the “wait” and “end of message” icons are strings right now, they will become icons before release
- Text delay will probably end up depending on the text style rather than being a typewriter value

Message Log

A sub-system used by the Typewriter, the Message Log keeps track of up to a set number messages that have been seen by the user so that they can be viewed later. The log works like a queue, so the oldest node is discarted whenever the log is full and a new one is loaded.

The system doesn’t provide a way to view logs yet.

Graphics manager

This manager handles all things related to graphics, such as rendering text, sprites and backgrounds.

The following sprite and background handling functions are exposed and available for use:

- Set a graphic (if setting a layer that already has a graphic assigned to it, a cross-fade transition will be applied by default)
- Sort graphics up, down, to front or to the back
- Move, rotate or scale graphics
- Change a graphics's tint color (including alpha)
- Use Unity's fill system to create horizontal, vertical or radial wipes
- Tween between two or more values (with looping as an option) for movement, rotation, scale, tint color and/or graphics fill

WARNING: Properly importing graphic files into your Unity projects

Because the function used to move a graphic changes its texture offset, it means that **importing the source files as default Sprites will result in offsets not displaying properly**. This happens because Unity sets all sprites' textures to the "Clamp" wrapping mode by default, which means the textures can't wrap around themselves.

If you want to, for example, create a looping, moving background by tweening the graphic's offset, you'll have to do the following:

1. Select the source image file you'll be using in Unity's Project tab, so that its import settings can be seen in the Inspector tab
2. For "Texture Type", select "Advanced" from the drop-down menu
3. Check "Alpha Is Transparent" if the graphic has any transparent areas
4. Make sure "Sprite Mode" is set to "Single"
5. Set the "Pivot" property to the pivot position you'll want to use with that graphic
6. Uncheck "Generate Mip Maps"
7. Set "Wrap Mode" to "Repeat"
8. Finally, set the graphics' "Max Size" and "Format" according to the file and your project's needs
9. Repeat this process for each and every image file you want to use in this way.
   Note: You can also select multiple files and set all these settings for all of them at once

Background and sprite management-related properties

- **Background size:** The size that background layers will use by default (I think this is a deprecated function I still haven't gotten around to removing)
- **Background layer count:** The number of background layers that will be made available for use during game execution
- **Sprite layer count:** The number of sprite layers that will be made available
- **Transition delay:** The default transition delay whenever a sprite or background is changed without explicitly specifying a delay

Text management-related properties

- **Text area size:** The width and height of the main text area
- **Text area pivot:** Where the main text area should be anchored to, relative to the whole screen
- **Text area graphic:** A sprite that will be drawn surrounding the text area. Should be a sliced sprite in order to facilitate scaling (don't know how it'd work if you fed it a single sprite)
- **Text area tint:** The tint color that will be applied to the main text area
- **Text area margin:** A margin between the text area graphic and the area where text can be displayed
- **Name text area size:** If set to values other than zero, this will result in a secondary text area where the name of the actor speaking a line will be automatically rendered
- **Name text area font:** The font that will be used for the name text area
- **Name text area font size:** Self-explanatory
- **Name text area font alignment:** Self-explanatory
- **Options over text:** Checking this option will make answers to questions overlap the text area. Not setting it has the options show up in the screen space that isn't occupied by the main text area (I've only tested this with the main text area anchored to the bottom of the screen)
- **Option count:** The number of option buttons that will be prepared (this is the max that the system will be able to show at any time)
- **Option size:** The width and height of each option button
- **Option background:** The background graphics that will be used for each option button
- **Option color:** Tint color that can be applied to the option backgrounds

Audio manager

A simpler manager that controls music and sound effects. The manager exposes functions for:

- Playing music and SFX on any of the available tracks
- Automatically fading out previous sounds on a track that is already playing before fading in the new one
- Setting a track’s volume
- Setting a track’s panning
- Setting a track’s pitch
- Volume, panning and pitch values can also be tweened between two or more values (said tweens can loop as well), but only for music tracks. However, tweens are only available for music tracks
- Global volume modifier for music and SFX

Differences between music and effect tracks

- Music tracks always loop, while effect tracks will play only once and stop afterwards
- The typewriter has a command that waits until a specific effect track is done playing before it continues typing, the “wait for SFX” command. Music tracks don’t have this feature because they loop forever
- The typewriter SFX that is played whenever a character is typed on-screen uses PlayOneShot, so it doesn’t use up an audio track, can’t be tracked with the WaitForSFX command and can play on top of itself
- Music tracks are the only sort of track that can have tweens applied to them

Variable Manager

This sub-system merely handles variable loading and saving. Variables can have any name and can only contain integer numeric values. More value types might be available later.

The following functions are available:

- Initialize a variable
- Read a variable’s value
- Set a variable’s value
- Perform an operation with a variable. Available operations include:
  - Add
  - Subtract
  - Multiply
  - Divide
  - Bitwise AND. Example: 5 AND 6 = 4 (in binary, 101 AND 110 = 100)
  - Bitwise OR. Example: 5 OR 2 = 7 (in binary, 101 OR 010 = 111)
  - Bitwise XOR. Example: 7 XOR 6 = 1 (in binary, 111 XOR 110 = 001)

This manager has two tweakable properties:

- **Strict mode:** Enabling strict mode basically prevents reading or modifying a variable that hasn’t been initialized beforehand, which is safer since it automatically throws out a warning if you mistyped the variable name or simply forgot to initialize it properly
- **Default value:** The value all variables will take by default upon initialization, in case you want them to be initialized with a value other than zero

Editor tools

The Dialog System has a set of editor extensions that allow you to use and edit several things:

- **Node editor:** Add, link and arrange message nodes
- **Script editor:** Edit the contents of each individual message node, applying text styles, actor names and script tags to the messages
- **Style editor:** Add, delete and personalize different text styles, which can then be selected in the script editor on a per-message basis
- **Script importer:** Allows you to load up a text asset (any .txt file) and automatically interpret it as node data. It currently overwrites all pre-existing node data, but I'll probably implement an import method that just appends the nodes to the existing data in the future

Node editor

The node editor allows you to do the following:

- Add unlinked nodes (“Add node” button)
- Add a node between two linked nodes (“+” button between the nodes)
- Click on a node to load its data on the Script Editor
- Zoom in or out within preset zoom values. Will be tweaked in the future
- Drag nodes around to arrange them. There are four movement settings:
  - **Single:** Moves the selected node only
  - **Children:** Moves the selected node and all of its children at once
  - **Parents:** Moves the selected node and all of its parents at once
  - **All:** Moves all nodes at once
- Link nodes with each other with one of three possible link types:
  - **Regular link:** One-to-one
  - **Question link:** One-to-many, with each option presented as text that is shown as a button on-screen
  - **Variable check link:** One-to-many, with a single variable name that is checked against any number of value comparisons (greater than, equal, different, lower or equal, etcetera)

Script editor

The script editor focuses on setting the properties of each individual message node, and has the following elements:

- **Editor text size:** Only affects the text size within the editor, mostly for readability sake
- **Language:** Allows you to add, rename and remove languages. The functions are all working, but the system will probably need an overhaul
- **Actor name:** This is what will be shown in the actor name text area (if it's set to be active in the Graphics Manager). You can either type a name or select it from a list of names used on other nodes within the script
- **Base style:** The text style that will be used for this message. You can add, remove and edit text styles using the Text Style Editor
- **Text:** The text that will be typed by the typewriter, including the special command tags
- **Basic commands:** A list of simple commands that don't require parameters to use. Please remember that some commands might not work depending on the text style you have set (ie: selecting a style that uses bold text by default means bold text tags will do nothing within that message)
- **Commands with parameters:** All commands that require parameters to invoke them. First you have to select a command to open up its individual parameter tools, and then you can fill in the parameters and add the corresponding command tag into the message
- **Save:** Applies all changes made to the script and saves the updated file
- **Reload:** Discards all changes made to the script and reverts them to the way they were before

Script command reference

All commands are grouped by the manager they rely on. Some of them have optional parameters that may be omitted, in which case default values will be used (more information on that is available on each specific command).

Typewriter manager commands

{\bold}, {\endBold}

All text found between these tags will be written **in bold**.

Note: This doesn't work if the style the message is using is set to render in bold, because it'd be redundant.

{\italic}, {\endItalic}

All text found between these tags will be written *in italics*.

Note: Like with bold text tags, these tags don't work if the style the message is using is set to render in italics. Other style tags are unaffected by this restriction.

{\color:RRGGBBAA}, {\endColor}

All text found between these tags will be written in the desired RRGGBBAA color (red, green, blue, alpha). Each color component is written in hexadecimal, so a value of 80808080 would mean RGB 128 128 128 with a transparency value of 128/256 (which equals 50%).

{\size:<number>}, {\endSize}

All text found between these tags will be written in the desired text size. The size is written as an integer value.

{\wait}

Stops the typewriter and waits for user input (currently spacebar only) before it continues typing the next part of the message.

{\endMessage}

Stops the typewriter and waits for user input (currently spacebar only) before moving on to the next message. Please note that all text and tags that may be typed after this tag will be ignored by the typewriter.

{\autoEndMessage}

Immediately moves on to the next message without user input. This tag can be used for things such as creating a node that only sets up a scene's sprites (so as to make the script more legible in the node which has the text), or to have more control on the script's pacing for specific situations. Please note that all text and tags that may be typed after this tag will be ignored by the typewriter.

{\waitForSFX}

Stops the typewriter until the last SFX that was played has stopped. Doesn't stop at all if the last SFX track isn't playing at the time, or if no SFX tracks have been played at all before invoking this command.

{\waitForSFX:<trackNumber>}

Stops the typewriter until the SFX on track #<trackNumber> has stopped. Values under zero or over the available number of tracks will be clamped to those two values.

Note: Track numbers use zero-index, which means they range from 0 to N-1 (ie: having five tracks means their index values range from 0 to 4). The same holds true for graphic layers as well.

{\delay:<time>}

Stops the typewriter for <time> seconds, and automatically resumes typing after the delay is over.

{\waitForAllFlags}

Stops the typewriter until all flags have been cleared (see below for more information on flags and their use).

{\waitForFlag:<flagNumber>}

Stops the typewriter until flag #<flagNumber> has been cleared.

{\raiseFlag:<flagNumber>|<flagDuration>}

Raises flag #<flagNumber> for <flagDuration> seconds, which can be used in conjunction with Wait for Flag commands to wait until a particular animation, tween or event is done (you must manually set the <flagDuration> to match the duration of the event you want to wait for).

{\typingDelay:<delay>}

Sets the delay between typing characters to <delay> seconds. A delay of zero will result in text being written immediately (except for commands that halt typing, such as delay, wait, endMessage or waitForSFX).

{\typingSFX:<name>}

Sets the audio file located at “Assets/Resources/Audio/<name>” as the sound that will be played each and every time a character is typed by the typewriter.

{\typingPitch:<pitch>}

Sets the pitch of the typing SFX whenever a character is typed by the typewriter. 1 equals normal pitch.

{\typingVolume:<volume>}

Sets the volume of the typing SFX whenever a character is typed by the typewriter in the range of 0 (muted) to 1 (full volume).

Audio manager commands

{\play:<name>|<type>|<trackNumber>}

<name>: String
<trackType>: “Music”, “SFX” or their respective indexes (0 and 1)
<trackNumber> (optional): integer (default = 0). 0 <= trackNumber <= N - 1, where N is the number of available layers of that graphics type as set in the Graphics Manager

Plays the audio file located at “Assets/Resources/Audio/**<name>**” with a <type> that must equal either “Music” or “SFX”, or can also use index values (0 for music, 1 for SFX). The track that will be played is <trackNumber>, but you can also leave this parameter blank, in which case track #0 of the selected track type will be used by default.

{\audioVolume:<type>|<trackNumber>|<volume>}

Instantly changes the volume for the specified track number of the specified track type.
Valid values range from zero (no volume) to one (full volume).

{\audioPan:<type>|<trackNumber>|<pan>}

Instantly changes the panning value for the specified track number of the specified track type.
Valid pan values are in the -1 (100% left) to +1 (100% right) range.

{\audioPitch:<trackType>|<trackNumber>|<pitch>}

<trackType>: “Music”, “SFX” or their respective indexes (0 and 1)
<trackNumber>: Integer. 0 <= trackNumber <= N - 1, where N is the number of available tracks of that track type as set in the Audio Manager
<pitch>: Float. 0 <= pitch <= 5

Instantly changes the panning value for the specified track number of the specified track type.
Valid pan values are in the 0 (inaudible) to 5 (5x normal pitch) range.

{\audioVolumeTween:<trackNumber>|<loop>|<volume0>|<delay0>|...|<volumeN>|<delayN>}

<trackNumber>: Integer in the 0 to N-1 range (N is the number of tracksof that kind created in the Audio Manager)
<loop>: Boolean (0 or 1, “True” or “False”)
<volume>: The volume value (0-1 range)

<delay>: The delay between this volume value and the next

Applies a volume interpolation to a music track between at least two values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two volume-delay pairs must be used for invoking this command, but there's no upper limit to the number of pairs that can be used.
Note 2: This command can only be applied to music tracks.

{\audioPanTween:<trackNumber>|<loop>|<pan0>|<delay0>|...|<panN>|<delayN>}

<trackNumber>: Integer in the 0 to N-1 range (N is the number of tracksof that kind created in the Audio Manager)
<loop>: Boolean (0 or 1, "True" or "False")
<pan>: The pan value (-1 to +1 range)
<delay>: The delay between this pan value and the next

Applies a pan interpolation to a music track between at least two values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two pan-delay pairs must be used for invoking this command, but there's no upper limit to the number of pairs that can be used.
Note 2: This command can only be applied to music tracks.

{\audioPitchTween:<trackNumber>|<loop>|<pitch0>|<delay0>|...|<pitchN>|<delayN>}

<trackNumber>: Integer in the 0 to N-1 range (N is the number of tracksof that kind created in the Audio Manager)
<loop>: Boolean (0 or 1, "True" or "False")
<pitch>: The pitch value (0-5 range)
<delay>: The delay between this pitch value and the next

Applies a pitch interpolation to a music track between at least two values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two pitch-delay pairs must be used for invoking this command, but there's no upper limit to the number of pairs that can be used.
Note 2: This command can only be applied to music tracks.

Graphics manager commands

{\changeGraphic:<name>|<type>|<layer>|<sorting>|<fadeDelay>}

<name>: String
<type>: “Background”, “Sprite” or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<sorting> (optional): “AsIs”, “InBack”, “InFront” or their respective indexes (0 to 2)
<fadeDelay>: Float >= 0 (a delay of zero means instant transition)

Assigns the graphics file located at “Assets/Resources/Graphics/**<name>**” to the specified layer number and type.
<sorting> and **<fadeDelay>** values are optional and can be omitted, in which case the graphic’s sorting won’t be changed and the default fade delay will be used.

{\moveGraphic:<type>|<layer>|<positionX>|<positionY>}

<type>: “Background”, “Sprite” or their respective indexes (0 or 1)
<layer>: Integer for the layer index
<positionX>, <positionY>: Float values

Moves a graphic to a specific position. (0;0) equals the lower-left side of the screen, and (1;1) the upper-right side.
Note: Please take into account the image’s pivot as you set it during sprite importing, since the graphic position will be relative to its pivot. Ie: with a pivot on the lower-left part of a graphic, setting its position to (1;1) would mean it’d be drawn entirely offscreen.

{\sortGraphic:<type>|<layer>|<position>}

<type>: “Background”, “Sprite” or their respective indexes (0 or 1)
<layer>: Integer for the layer index
<position>: Float value

Moves the selected graphics layer to a specified position within the hierachy.
Note: Layers of both graphics kinds are mixed together and can be sorted in any way, so you’ll have to keep track of their sorting positions carefully when using this command.

{\tweenGraphicsColor:<type>|<layer>|<loop>|<color0>|<delay0>|...|<colorN>|<delayN>}

<type>: “Background”, “Sprite” or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<loop>: Boolean (0 or 1, “True” or “False”)
<color>: A color in RRGGBBAA hexadecimal format.
<delay>: The delay between this color and the next

Applies a color tint that is interpolated between at least two colors with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two color-delay pairs must be used for invoking this command, but there’s no upper limit to the number of pairs that can be used.

{\tweenGraphicsPosition:<type>|<layer>|<loop>|<positionX0>|<positionY0>|<delay0>|...|<positionXN>|<positionYN>|<delayN>}

<type>: “Background”, “Sprite” or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<loop>: Boolean (0 or 1, “True” or “False”)
<positionX>, <positionY>: The position of the graphic’s pivot relative to the screen. (0,0) = bottom left, (1,1) = top right.
<delay>: The delay between this position and the next

Interpolates the target’s position among at least two points with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two positionX-positionY-delay triplets must be used for invoking this command, but there’s no upper limit to the number of triplets that can be used.

{\tweenGraphicsRotation:<type>|<layer>|<loop>|<rotation0>|<delay0>|...|<rotationN>|<delayN>}

<type>: “Background”, “Sprite” or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<loop>: Boolean (0 or 1, “True” or “False”)
<rotation>: Float. The graphic’s desired rotation (0-360).
<delay>: Float. The delay between this rotation and the next

Applies a rotation to the graphic is interpolated between at least two rotation values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two rotation-delay pairs must be used for invoking this command, but there's no upper limit to the number of pairs that can be used.

{\tweenGraphicsScale:<type>|<layer>|<loop>|<scaleX0>|<scaleY0>|<delay0>|...|<scaleXN>|<scaleYN>|<delayN>}

<type>: "Background", "Sprite" or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<loop>: Boolean (0 or 1, "True" or "False")
<scaleX>, <scaleY>: Float. The graphic's scale. (1,1) = original size
<delay>: The delay between this position and the next

Interpolates the target's scale among at least two values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two scaleX-scaleY-delay triplets must be used for invoking this command, but there's no upper limit to the number of triplets that can be used.

{\tweenGraphicsFill:<type>|<layer>|<loop>|<fillClockwise>|<fillMethod>|<fillOrigin>|<fill0>|<delay0>|...|<fillN>|<delayN>}

<type>: "Background", "Sprite" or their respective indexes (0 and 1)
<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)
<loop>: Boolean (0 or 1, "True" or "False")
<fillClockwise>: Boolean (0 or 1, "True" or "False")
<fillMethod>: An integer in the 0-4 range. Respective indexes:
<fullOrigin>: An integer in the 0-15 range. Respective indexes:
<fill>: Float. The fill percentage of the graphic (0 = 0%, 1 = 100%)
<delay>: The delay between this fill percentage and the next

Interpolates the target's fill among at least values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.
Note: At least two fill-delay pairsmust be used for invoking this command, but there's no upper limit to the number of pairsthat can be used.

{\tweenGraphicsOffset:<layer>|<loop>|<offsetX0>|<offsetY0>|<delay0>|...|<offsetXN>|<offsetYN>|<delayN>}

<layer>: Integer in the 0 to N-1 range (N is the number of layers of that kind created in the Graphics Manager)

<loop>: Boolean (0 or 1, “True” or “False”)

<offsetX>, <offsetY>: Float. The graphic’s offset. (0,0) = default, (1,1) = wrapped all the way to the other side.

<delay>: The delay between this offset and the next

Interpolates the target’s texture offset among at least two values with the specified delays between them. Setting **<loop>** to true will make the tween continue forever.

Note: At least two offsetX-offsetY-delay triplets must be used for invoking this command, but there’s no upper limit to the number of triplets that can be used.

Note 2: This command can only be applied to background layers.

Note 3: Using offset values outside of the 0-1 range requires that you set the image file you’ll be using to wrap, not clamp.

Variable manager commands

{\addVariable:<name>|<value>}

<name>: String
<value> (optional): Integer

Initializes a variable on a specific value if it's typed into the command, or the default value set in the Variable Manager otherwise.
Note: Using this while the Variable Manager isn't running under strict mode isn't necessary.

{\changeVariable:<name>|<value>|<operation>}

<name>: String
<value>: Integer
<operation>: Set, Add, Subtract, Multiply, Divide, And, Or, Xor or their respective indexes (0-7)

If an **<operation>** type is specified, that operation is carried out on the variable **<name>** using the **<value>** integer value. Otherwise, the variable will simply be set to **<value>**.
Note: If the Variable Manager is running under strict mode, invoking this command on a variable that hasn't been initialized yet will do nothing and return a warning.

{\removeVariable:<name>}

<name>: String

Deletes a variable and its value.
Note: Be careful when attempting a changeVariable operation after you've deleted a variable with this command while the Variable Manager is running in strict mode. In that case, you'll have to use the addVariable command before changeVariable.

{\clearAllVariables}

Deletes all variables and their values.