Introduction

Welcome to Lassie Shepherd.

Shepherd is the third generation of Lassie Adventure Studio, and by far the biggest and most robust build of the system to date. Developers coming from Lassie Director and LassieAS will be pleased to see some very strong similarities between the systems as far as media structuring and interactive controls are concerned. However, Shepherd is an entire new system built from the ground up using Flash ActionScript3, so developers should still be prepared to face some drastic changes in work flow and system capacity that accompanies with any major change in technology. Ultimately, Shepherd has been designed with the intent to bridge the gap between legacy Lassie, easy use, and heavy technology.

Overview

Here are some of the new, key points to be aware of while approaching Shepherd:

- Shepherd is a web application. While previous versions of the Lassie editor would run as a desktop application, Shepherd must be run on a web server so that it can take advantage of PHP's file management capabilities. This concerns the editor application ONLY. The Lassie game player runtime may still be run from the desktop once it and all supporting game assets have been downloaded from the web-based editing environment. While the decision to go web-based was largely influenced by pre-Flash 10 file-writing capabilities, it was also influenced by the fact that a web-based application would be helpful for worldwide team-based project development. Note that Shepherd can still be run on a local machine that is configured with Apache and PHP. Therefore, developers may find it helpful to run WAMP (Windows Apache MySQL PHP) or MAMP (Mac Apache MySQL PHP).

- Shepherd's asset management system has changed dramatically from previous versions of the Lassie engine. While previous versions of the engine were reliant on individually composed SWF movies that were loaded in bulk at runtime, Shepherd leverages the new capabilities of ActionScript3 that allows all kinds of multimedia assets to be compiled into a single SWF archive, and those embedded assets can be extracted and used at runtime. So, where a previous versions of Lassie might have loaded 20 external media assets to compose a single room layout, Shepherd can load a single external media asset and extract all 20 media items out of the one loaded resource. This makes runtime assets and load management far more efficient, not to mention that it makes for a much cleaner external library. The one downside is that setting up a Lassie resource file now takes a little bit more discipline on the developer's behalf. However, the additional steps for publishing assets are not difficult, and the benefits of the SWF archive media management system greatly outweigh its shortcomings. For complete details on how to compile media libraries to use within the Shepherd system, see the Shepherd Library tutorial.

- Shepherd has a drastically new a different system of publishing games. For developers who are disheartened by Shepherd's dependency on a web server, this is good news: Shepherd publishes XML data structures that can be parsed directly into the game player. For developers coming from LassieAS, this means that you will never have to publish a "_data.fla" movie again! Because the runtime can directly load and parse Shepherd data, you can skip the intermediary step of compiling a data file from Flash before testing your game. However, it is still important to note that the Lassie Shepherd editor application does not automatically publish game data when saving a layout. Whenever you make updates to your game configuration that you want to be reflected in your game, you'll need to specifically "PUBLISH" game data to generate new XML.

- Shepherd introduces an entirely new game scripting system that both beginner and advanced programmers alike can appreciate: rather than programming with psudo-ECMA style script commands like in previous versions of the Lassie engine, Shepherd is programmed entirely using XML snippits. For those unfamiliar with XML, it is a tag-based mark up language that works exactly like HTML. If you know the basics of simple HTML, then scripting in Shepherd will be a breeze. The XML syntax has the added benefit of allowing all method parameters to be labeled (as opposed to having to remember the meaning of each parameter in an unlabeled list), and also that XML supports nesting structures – which paves the way for full conditional logic operations.

- And finally, developers coming from LassieAS will find some really nice new feature in the Shepherd interface. That's not to say that the Shepherd editor is perfect – quite on the contrary, it's far from it. The current release of Shepherd is still (relatively speaking) primitive and has some usability issues to address in the long run. However, all short-comings aside, legacy developers will likely agree that even with Shepherd's quirks, it is a step up from the archaic interface of Lassie Director and Lassie AS.

So, give it a shot, good luck, and enjoy! And as always, happy adventuring!

Installation & Setup

System Requirements

- Flash CS3 or later (for publishing asset libraries)
- A PHP-enabled web hosting service (or localhost) running PHP 4 or later.
- FireFox 2+ or Safari 2+ (some display features will not work in Internet Explorer).

Installation

- Download and unzip the Lassie Shepherd archive.
- Upload the full "www" directory anywhere onto a PHP 4+ enabled web server or localhost.
- Launch FireFox 2+ or Safari 2+ (sorry, IE is not officially supported) and go to the URL at which you uploaded Shepherd's "www".
- Log into the Shepherd site with the following default credentials:
    - user: admin
    - password: lassie

If you have trouble logging in for the first time with the default password, then file permissions have probably been reset on Shepherd's login credentials file. To correct this, find the file "php/login.txt" and confirm that its permissions are set to 755 ("-rwxr-xrwx"). If login issues or file read/write issues persist, check to make sure that no hidden files were unzipped along with the install package.

Running Shepherd Locally

While Shepherd is designed to run on a web server as a web application, that doesn't mean it can't also run locally on your computer without an active internet connection. You'll just need to set up a local PHP-enabled server environment in which to run Shepherd. This can be done easily and for free on both Windows and Macintosh computers. See below for an extremely brief summary of setting up a PHP environment. For much more detailed and reliable documentation, just run a web search for "setting up Apache PHP on [your system here]".

Windows: download and run the Windows PHP installer, available at http://www.php.net/downloads.php. This should take care of setting up both Apache and PHP for you. If you need to do perform any manual configuration, see configuration instructions at: http://www.thesitewizard.com/php/install-php-5-apache-windows.shtml. Once Apache and PHP are set up, place the contents of Shepherd's "www" directory into Apache's "htdocs" directory. You should be able to open Shepherd within a web brower using the address "http://localhost/".

Mac: really easy... use MAMP (Mac Apache MySQL PHP). You can download the basic version of the application for free from http://www.mamp.info. Place the contents of Shepher's "www" folder in the "htdocs" directory within the MAMP application folder. Startup

MAMP and leave in running in the background, then go to "http://localhost:8888/" within your web broser.

Creating logins

Your Shepherd application will always have an "admin" account which provides special privileges for defining logins and creating projects. The admin login is defined by default and cannot be deleted. However, you can (and SHOULD) immediately change the admin account password from the default value of "lassie". This is currently done by overwriting the "admin" account settings with the "Create a user" form. Use the "Create a user" form to create a login name called "admin" with a password of your choice. The newly created admin login will overwrite the existing login with your new password, effectively changing the account's password. It is then highly recommended that you create one additional login to use as your primary development account. The admin account gets lots of extra options on the main Shepherd page which you won't need on a regular basis, so it's best to avoid accidental tampering with those controls by just logging in as a secondary user while working on a project.

Creating a Project

A Shepherd project is one complete set of resources making a full game. Shepherd allows you to manage multiple projects independently of one another (or, consider forking your game's build periodically to creating a development snapshot that you can revert to). To create a new project, just enter a project name in the "Create a project" form and submit. It is suggested that you limit a project's name to uppercase and lowercase letters, numbers, hyphens, and underscores. The validity of other characters may become an issue depending on your server environment. After you've created a project, you're ready to start developing!

Project Pages

Each project that you create will include four HTML pages for working with the project. The four pages include:

- **Edit**: this page brings up the project's copy of the Shepherd editor application. Use the Shepherd editor application to create and manage game layouts and data.

- **Play**: this page brings up the project's copy of the Shepherd game player application. The game player will reflect the project's current published status. By default, the game player will display as a dark lank screen until game data has been published out of the editor.

- **Library**: this page manages SWF media libraries that will be imported to provide multimedia assets to the Shepherd editor and game player. Media libraries must be created to the specifications outlined in the Library Tutorial document. Once you have created and compiled a media library, upload it into your project using the "Upload" form on the library page. NOTE: you WILL need to empty your cache and refresh your browser after uploading a new copy of an exiting SWF movie. While

Shepherd automatically prevents the caching of XML resources, it allows the bigger and heavier SWF media files to cache so that you can save on bandwidth.

- **XML**: this page lists all game XML files that have been published for the project. Keep in mind that a game layout is not automatically published as XML each time you save the layout. You must specifically opt to "PUBLISH" a layout before new XML is generated for consumption by the game player. To run a game locally on your desktop, maintain a local copy of your game resources including all media library SWFs, all XML, and the game player application.

Known Bugs and Work Arounds

Unfortuantely, the Shepherd Editor does have some known issues which can cause **SERIOUS DATA LOSS**. Unfortunately, these are structural issues within the Shepherd framework that will take massive effort and recoding to address. Thankfully however, the issues are avoidable within the Shepherd workflow. So, for the time being, these issues are the responsibility of the developer to be aware of and work around. Obviously, correcting these issues is a top priority in the long-term plan for Shepherd.

1. Room layout data is completely lost when a room is saved without first being loaded into Shepherd's room layout editor. That means that you CANNOT load a room and then just edit its scripts and dialogue trees without first loading the room layout. Doing this will cause the room to be saved without the layout data. It is ESSENTIAL that -upon loading a room– that you first press "edit layout" before saving room data again. This is an extremely high priority issue to fix, however it will be a major invasive change to the application which may potentially take months to work through.

2. Editor fields are NOT quotation-safe. That means that quote marks ("") will interfere with the Shepherd data format and will result in corrupted data. As a result, it is essential that a developer NOT place quotation marks into editor fields. Instead, use the HTML quotation entity: **&quot;**

Getting Started

Now let's take a look at the major steps involved in setting up a working game. The following does NOT provide comprehensive documentation on the use of the Shepherd editor tools. It is simply a high-level overview that walks through the process of setting up game components. Additional information on the detailed use of Shepherd controls will be included when the documentation becomes available.

1. **Skin your game UI.**

   A Lassie game's user interface (UI) is built from display elements stored within a designated SWF file. You can download and customize the provided UI template. When customizing the UI, you may alter the graphics in any way that you like as long as all exiting MovieClip structures and instances names remain unchanged. Publish your UI to a SWF file called "ui.swf", then upload that into your Shepherd project library through the "library" page. See the UI FLA article in the section titled "Resource Management in Lassie Shepherd" for more information. You do NOT need to upload the UI FLA file into your Shepherd project library.

2. **Create media libraries.**

   Lassie composes game layouts by extracting multimedia assets out of specially constructed library SWFs. A library SWF is basically just a package that contains Flash resources. You'll need to create a minimum of one library stocked with global assets (character animations, inventory items, etc), and [ideally] one separate library with resources for each room in your game. For detailed information about creating library SWFs, see the library tutorial. As you create new media libraries, upload their SWF files into your Shepherd project library through the "library" page. You do NOT need to upload any FLA files into the project library.

3. **Setup global configuration.**

   Once you've created and uploaded a media library with all global game assets included, go into the Shepherd editor and configure your global game settings, including library imports, playable characters, inventories, inventory items, etc. Start out on the "Setup" panel and add your global media library(s) to the list of media libraries to import. Then configure at least one playable actor definition within the "Actors" panel. You may detail that actor's default dialogue responses in the "Responses" panel and the actor's inventory configuration in the "Collections" panel. See Shepherd editor documentation for more details about the tool's individual editor panels.

4. **Create a room.**

   Now go to the "Rooms" panel within the Shepherd editor. Choose to "select" a room, and then create and load a new room layout. Select media libraries to import for use within this room layout, then open the room layout editor to compose the room's visual appearance. Be sure to save your room's layout data frequently! Once you have built a room layout, cite the room's ID in the "starting room" field in the upper-left corner of the Shepherd "Rooms" panel. As you develop your project, you can always change the starting room of your game by updating the starting room ID.

5. **Publish you game data as XML.**

   The Lassie player loads in compiled XML data which you must specifically publish out of the Shepherd editor. XML data will NOT be published automatically each time you save your Shepherd data. The Lassie player loads two types of XML data: one for global game configuration, and a second type that defines room layouts. At the current time, global XML and each individual room's XML must be published separately (there is currently no "publish all" macro command, sorry). Start by going into Shepherd's "Setup" panel and click the "PUBLISH" button to publish global data. Then, click the "PUBLISH" button on the Shepherd "Rooms" panel for each working layout that you update within the editor. Only the currently loaded room will be published. Make sure to publish all of your project's XML before previewing your game.

6. **Preview your game.**

   Go to the Lassie player application on your project's "Play" page to have a look at your game as it currently exists!

Creating Game Media

Resource Management in Lassie Shepherd

Lassie Shepherd follows along the same lines as LassieAS in that all game media is loaded from a collection of external resources into a pre-compiled Flash movie that assembles and runs a game. However, Lassie Shepherd consumes external resources in a slightly different manner than LassieAS. There are two types of external resources in Lassie Shepherd: media libraries, and once special resource for the game's user interface.

Media Libraries

In LassieAS, all resources were published SWF movies that loaded into the player application and went direct-to-stage, meaning that what was on the stage in the external SWF movie was what displayed within the Lassie Player. This was due to limitations imposed by ActionScript2, which did not allow SWF movies to dynamically share resources between one another.

Thankfully, this limitation was removed in ActionScript3. Now an external SWF movie can be treated as nothing more than a box or "package" which contains Flash media (art, animation, sound, etc). Another Flash application can load this "package" and then consume the individual resources contained within its library. This has huge advantages for load efficiency and asset organization, although it take a small amount of additional work on the developer's part while setting up a resource package.

Within the Lassie Shepherd framework, these packages of Flash assets are known as "media libraries", or just "libraries". The most important thing to be aware of about media library SWF movies is that the Lassie Player will never display their stage directly. The Lassie Player will simply consume resources from within a library SWF, while the SWF as a whole remains hidden away behind the scenes. So: what you place on the stage within a media library resource is meaningless. It's only the available contents of its symbol library that will be of importance to the Lassie Player.

The Lassie Shepherd Player application is entirely engineered to leverage this AS3 dynamic resource sharing in the form of media libraries. That means that all media you produce for Shepherd MUST be compiled for ActionScript3, Flash Player 9 (a Flash Player 10 version of the player is coming); and must conform to the specifications outlined in the Shepherd Library tutorial. The process will likely seem tedious to new developers at first, but it quickly becomes simple and routine after you've done the motions a few times and get the hang of it.

Anytime that you create or update a new media library SWF, be sure to upload it into you Shepherd project library through the "library" project page.

User Interface

The one external resource that the Lassie Player consumes without dynamic library sharing is a game's user interface. Thankfully, the user interface is downright simple to manage and configure compared to media libraries. All you need to do is skin the provided UI FLA template. Download and open the provided FLA template for the user interface, and start customizing the graphics to look any way that you like. The only trick is this: don't change the naming or structure of anything within the document. You'll notice that all MovieClip elements that are organized on the stage have specific instance names assigned to them. You'll also notice that MovieClips have been built into a very specific hierarchy of nesting certain elements inside one another. So, don't alter these qualities of the template. You may change the position of items on screen, you may change fonts, resize elements, apply new graphics, etc. In general, you may alter anything that you'd like concerning the appearance of the layout. However, if it concerns the technical structure of the document, leave it as is. Once you have skinned the template, publish it as a SWF movie named "ui.swf" and upload that into your Shepherd project library.

Browser Caching (IMPORTANT)

To avoid the headaches induced by banging your head against a wall, be sure to **clear your browser cache** after uploading a new version of an existing media element. While Shepherd avoids caching data elements (which have small file size and change frequently), Shepherd tries to respect your bandwidth by allowing media elements to cache within your web browser. So, you will likely continue to see outdated versions of your media assets –even after having uploaded a new version of a media library– until you have specifically emptied your browser cache and refreshed the browser view.

To speed up cache clearing, find some short cuts within your web browser. With Safari, you can trigger an immediate cache dump with a keyboard shortcut (command + shift + E). With Firefox, try installing the developer's toolbar plug-in; it will make cache clearing available with one-click into the miscellaneous options menu.

Configuring the User Interface

The Lassie Player user interface (UI) is built using assets from an external SWF movie at "lib/ui.swf". This "lib/ui.swf" is the Lassie Player's only mandatory external file requirement (all other media libraries used for a game are included by the developer). However, the UI SWF is not a true Lassie media library. The UI SWF does NOT require the "com.lassie.lib.Library" class that is required for all other Lassie media. This is because UI assets are pulled from the SWF's stage rather than its symbol library. (note: you may choose to implement the Lassie library class on the UI SWF if you wish to combine other library media with the interface graphics.)

To create a custom game UI, just open the provided "ui.fla" file and customize the default component graphics. It does not matter how you change the appearance of the graphics, just as long as the object hierarchy and naming remains in tact. What does that mean? It means that you cannot nest graphical elements into a different structure than they are by default. Also, as you click around to different UI objects, you'll notice that all items have been given a stage instance name. These instance names are extremely important and MUST remain intact.

UI media is laid out on the "ui.fla" document's stage becasue (in most cases) the position of the interface elements within the UI document will determine that element's position on screen within the Lassie Player. So, it is recommended that you first size the UI FLA's stage to match the dimensions of your game, then arrange and scale the individual graphical components to appropriately fill the space. Notes about individual UI elements are as follows:

Action Selector

stage instance name: "uiActionSelector"
type: composite (MovieClips)

The action selector is the game's verb picker. The action selector contains nested buttons used to call individual actions. These buttons are named "action_0", "action_1", "action_2", etc. You may include as many (or few) action buttons as you'd like, just so long as they're named sequentially starting with "action_0". Each action will map to a puppet's corresponding action index. The stage position of the action selector components within the UI document does not matter given that it will be dynamically positioned on top of each puppet.

Cursor

stage instance name: "uiCursor"
type: MovieClip

The cursor MovieClip bundles together artwork for all of the game's basic (non-inventory) cursors. Each cursor state should be placed on its own labeled timeline frame within the cursor movie. Shepherd expect the cursor MovieClip to have a minimum of three frames, labeled as "on" (active state), "off" (inactive state), and "wait" (disabled state). You may include unlimited additional custom states, each with a unique frame label. A common addition to the cursor MovieClip is eight directional arrow states used to highlight room exits.

Dialogue Subtitle
stage instance name: "uiSubtitle"
type: TextField

The dialogue subtitle is a dynamic text field that displays dialogue above speaking puppets. The color and position of the text field will be dynamically set during game playback. When configuring the field, make sure to embed all fonts and enable the field for HTML display (use the Flash properties button labeled as "<>" to enable HTML). While writing game texts, you may include basic HTML formatting such as <br>, <b>, and <i>. However, if you plan to use HTML bolding or italics, be sure to embed the bold and italic versions of the display font.

Contextual title
stage instance name: "uiContext"
type: TextField

The contextual title announces the current game actions being staged. The contextual title is positioned within the game at its coordinates in the "ui.fla" document.

Dialogue tree menu
stage instance name: "uiTreeMenu"
type: composite (MovieClip, TextFields)

The dialogue tree menu is the menu used for selecting topics from a dialogue tree. This is a composite element made up of one shell MovieClip with several nested elements. The shell MovieClip's position on stage determines its position within the game. The nested elements are TextFields for displaying individual topic lines, and two MovieClip buttons used as scroll arrows. The TextField line items are named "option_0", "option_1", "option_2", etc. You may include as many (or few) line item fields as you would like, just so long as they are sequentially numbered. The number of topic TextFields determines the maximum number of lines that will display before menu scrolling is enabled. The scroll arrows are labeled as "scroll_prev" and "scroll_next".

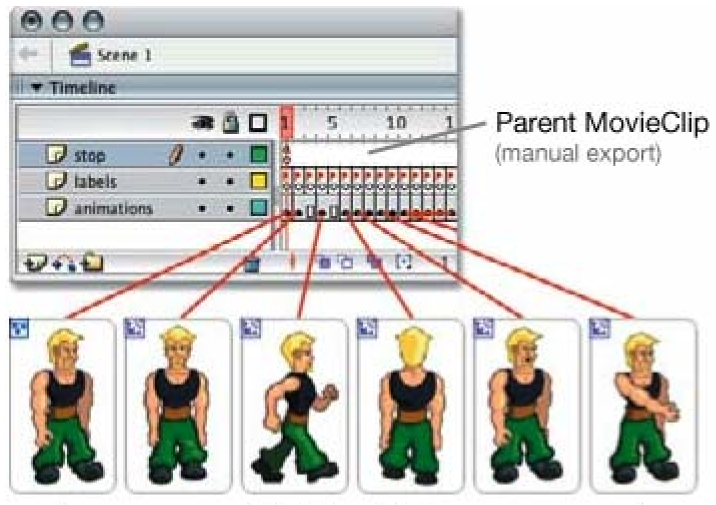
Inventory display
stage instance name: "uiInventory"
type: composite (MovieClips)

The inventory is the layout used to display items within the game's active inventory collection. The inventory is composed of one shell MovieClip, and many nested MovieClips used as item slots and scroll arrows. The shell MovieClip's position on stage determines the inventory's position within the game. The main MovieClip contains many empty "item" MovieClips, which are placeholders for inventory items within the display. The slots are named "item_0", "item_1", "item_2", etc. You may include as many (or few) slots in the inventory layout as you'd like, just so long as they are sequentially numbered. The number of slots within the layout determines the maximum number of items that will display in the inventory before scrolling is enabled. The scroll arrow MovieClips are labeled as "scroll_prev" and "scroll_next".

Creating Multi-Directional Puppet Animation Views

An actor/NPC puppet is composed of a series of animations (Flash MovieClips) arranged along a single master MovieClip timeline. Each animation's frame on the master timeline is labeled so that the Lassie Player can seek a specific animation by name. The following MovieClip formatting applies to an actor puppet, NPCs, and any other puppet layers that will speak or move intelligently within your game.

Defining Directional View States



All puppet layers have the ability to dynamically animate using 8 different directional views which reflect the puppet's implied angle to the camera. These 8 directional animations make up an animation "set" which all share a common base name followed by their respective directional number (for example: "speak1", "speak2", "speak3", etc). The numbering of directional animations is as follows:

- **1** : Back view
- **2** : Back-quarter right view
- **3** : Side right view
- **4** : Front-quarter right view
- **5** : Front view
- **6** : Front-quarter left view
- **7** : Side left view
- **8** : back-quarter left view

Of course, it's extremely impractical to make eight views of every single animation in a game. If a puppet will only ever be seen from a single view angle, then you need only create graphics for that single directional state. Also, the Lassie Player has some time-saver features built in which generate left-facing views on the fly by mirroring the corresponding right-facing view. Ultimately the Lassie Player picks a directional view for a puppet based on a procedure of checking for direction-specific animations, and then defaulting to other animations if the specific view could not be found. The Lassie Player's process of selecting a character animation works as follows:

SCENARIO: We want to set a puppet's "speak" animation, and the puppet is currently in view-state 6 (front-quarter-left).

1. First, the Lassie Player tests for an animation matching the provided set name and the puppet's current directional view. In our example scenario, the player would look for an animation called "speak6". If the animation is found, the animation is set and this process ends.

2. If a specific directional animation was not found, then the Lassie Player next looks to see if the target view angle is a left-facing view (angles 6-8). If the puppet is currently left-facing, then the Lassie Player will attempt to set the puppet's corresponding right-facing angle and mirror the puppet. In our scenario, the we are trying to set view 6, which is left-facing (front-quarter-left). So, the player would look for the front-quarter-right ("speak4") animation, then set and mirror the animation if it was found. If a right-facing animation was successfully mirrored for a left view, then the process ends.

3. If no direction-specific animation was able to be set, then the Lassie Player defaults to attempting to display the set's default animation, labeled with the base set name. In our scenario, the Lassie Player would default to trying to display an animation called "speak". While setting this default frame label, the Lassie Player works on the assumption that the raw image/animation on the frame is drawn in a right-facing direction (ie: our talking character is drawn with their body facing to the right). Working on the assumption, the Lassie Player will then mirror this default frame display if the puppet currently has a left-facing view setting (views 6-8).

4. If no custom view animation and no default set animation could be found with the provided name, then the Lassie Player defaults to just setting the puppet to it's first frame. Again, the Lassie Player works on the assumption that the raw image/animation on frame-1 is drawn in a right-facing direction (ie: the character is drawn with their body facing to the right). Working on the assumption, the Lassie Player will mirror this default display if the puppet has a left-facing view setting (views 6-8).

The Basic Lassie Player Puppet Animations

The Lassie Player has three core animation sets which it automatically attempts to use on puppets. These three core animations reflect a puppet's resting state, moving state, and speaking state. When building your main actor puppet, you'll need to create animations for all of these animation sets. When setting up NPC's, you can get away with only defining as many animations as the NPC will need (for example, if an NPC remains in a stationary location throughout the game, it would not need "move" animations). When setting up your puppet's MovieClip, place one animated state per frame along the main MovieClip's timeline, and label each frame with the animation's id. Core animation IDs are as follows:

Resting state animation frames

- "rest": The default resting view of the puppet (required)
- "rest1-5" : The five primary right-facing resting views of the puppet (suggested)
- "rest6-8" : The additional three left-facing resting views of the puppet (optional)

Moving state animation frames

- "move": The default moving view of the puppet (required)
- "move1-5" : The five primary right-facing moving views of the puppet (suggested)
- "move6-8" : The additional three left-facing moving views of the puppet (optional)

Speaking state animation frames

- "speak": The default speaking view of the puppet (required)
- "speak1-5" : The five primary right-facing speaking views of the puppet (suggested)
- "speak6-8" : The additional three left-facing speaking views of the puppet (optional)

More expressive speaking

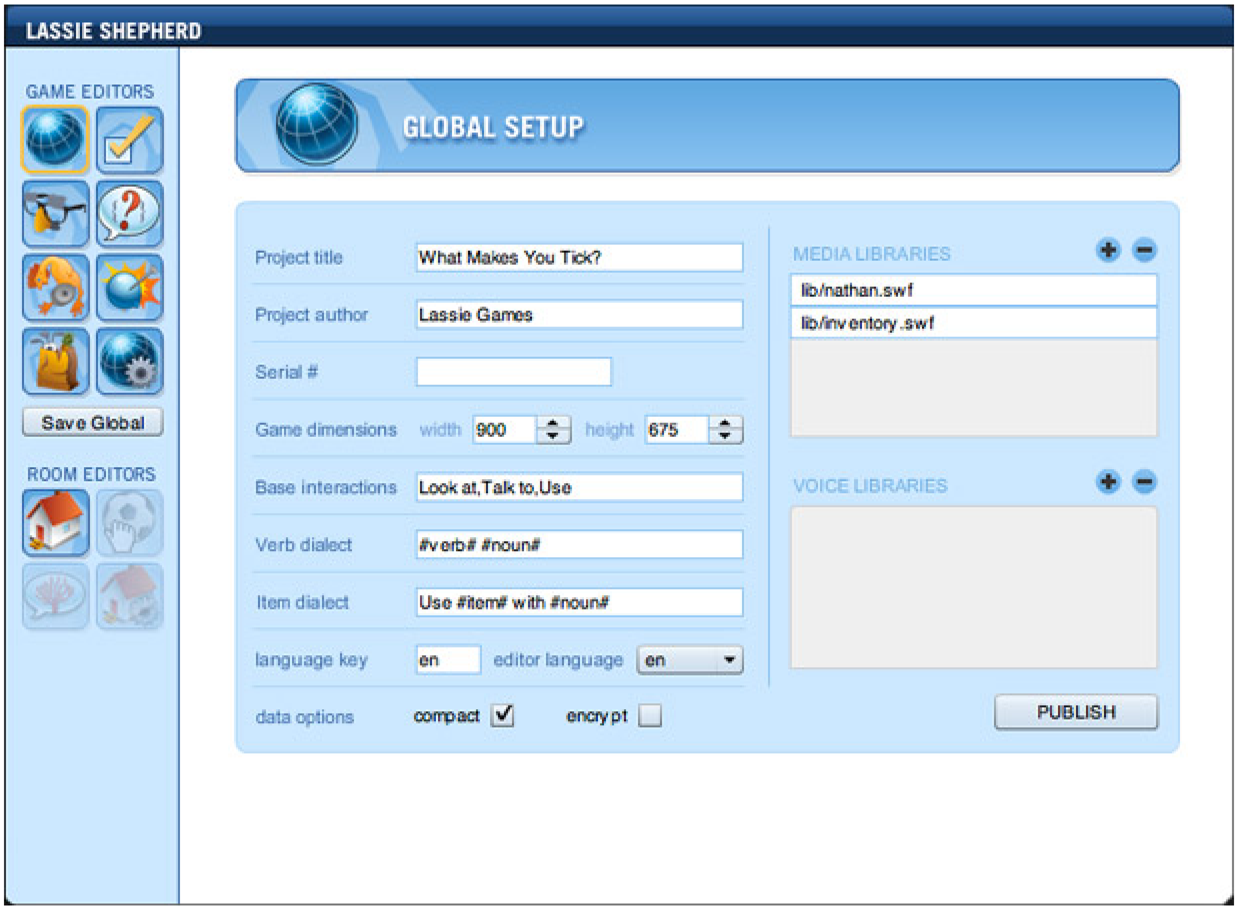
Additional talk animations are optional, though are very handy if you’d like to include a few different facial expressions of your character talking. These can be specified for each dialogue line within the Adventure Studio Editor.

Using Lassie Shepherd

The Shepherd Launcher Panels

The following is a very short primer on the Shepherd Editor tools. This section is designed to give a very broad overview on the purpose of each control panel, although it does not get into specific details. For specific information about each control field within the Shepherd Editor, roll your mouse over the field's label text within the editor interface. Most control label include a tool tip that pops open to provide specific information about the control field in question. Further information and documentation on the Lassie Shepherd Editor will become available as it is written.

The Setup Panel



Project title	What Makes You Tick?
Project author	Lassie Games
Serial #	
Game dimensions	width 900 height 675
Base interactions	Look at,Talk to,Use
Verb dialect	#verb# #noun#
Item dialect	Use #item# with #noun#
language key	en editor language en
data options	compact ✓ encrypt ☐

MEDIA LIBRARIES
lib/nathan.swf
lib/inventory.swf

Use this panel to configure general game settings. Major controls:

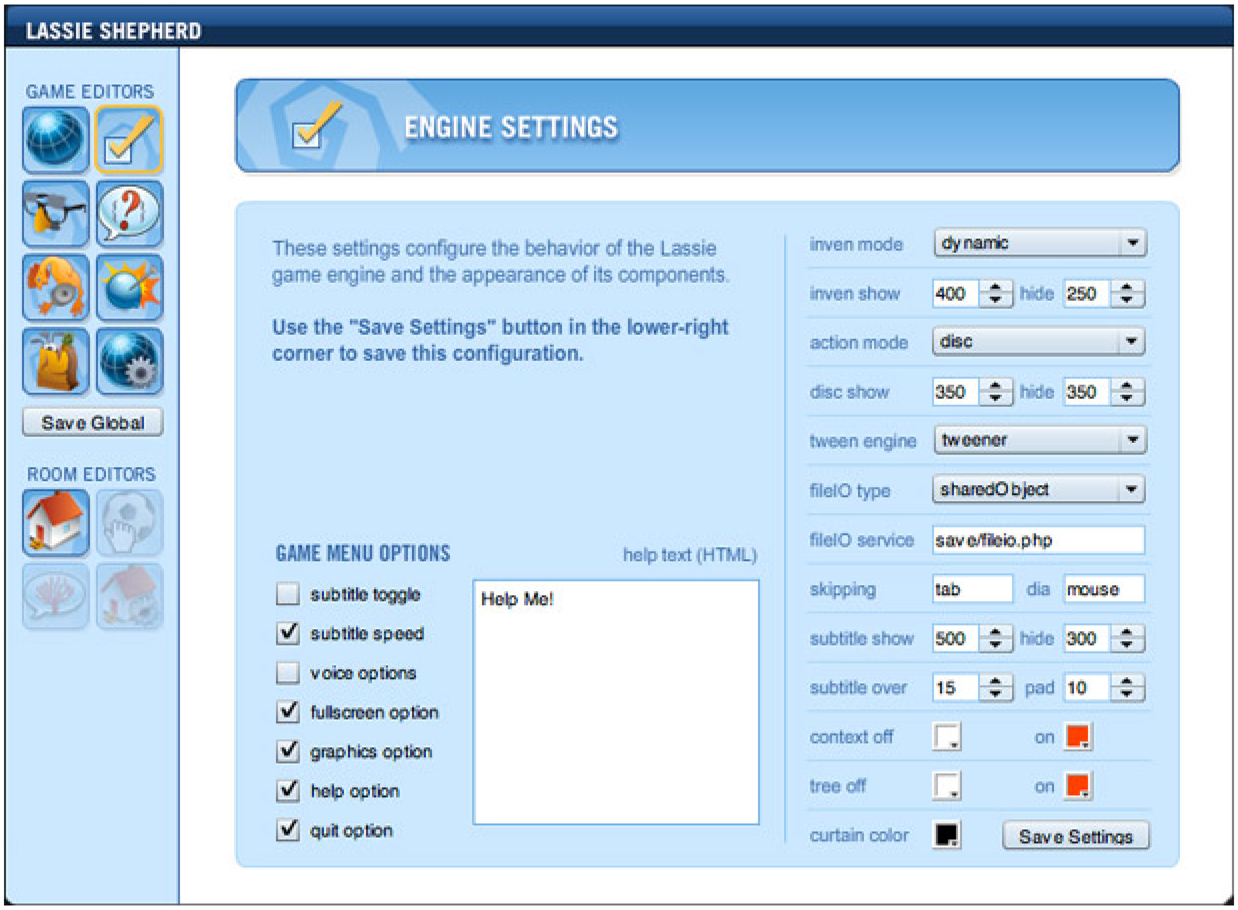
- Enter project credentials and serialization. A valid serial number will disable the Lassie Player's watermark. However, serial numbers are not currently being issued given that Shepherd is still in the early stages of beta release.

- Use the Media Library list to import global media library SWFs. Global media libraries will provide assets for global game elements such as playable characters and inventory items.

- Define a list of basic interactions. All interactive objects within your project will be created with each of those interactions.

- Click the "PUBLISH" button to export all of your global game data as XML. The Lassie Player will not be able to load your game until you have published global configuration data. Global configuration is NOT automatically published as XML when you save editor data.

The Settings Panel



These settings configure the behavior of the Lassie game engine and the appearance of its components.

Use the "Save Settings" button in the lower-right corner to save this configuration.

GAME MENU OPTIONS

- ☐ subtitle toggle
- ✓ subtitle speed
- ☐ voice options
- ✓ fullscreen option
- ✓ graphics option
- ✓ help option
- ✓ quit option

help text (HTML)

Help Me!

inven mode	dynamic		
inven show	400	hide	250
action mode	disc		
disc show	350	hide	350
tween engine	tweener		
fileIO type	sharedObject		
fileIO service	save/fileio.php		
skipping	tab	dia	mouse
subtitle show	500	hide	300
subtitle over	15	pad	10
context off	☐	on	🟥
tree off	☐	on	🟥
curtain color	■		Save Settings

The settings panel edits the Lassie Player's system configuration file ("xml/system.xml"). This system configuration file is loaded before any other game resources and defines how the Lassie Player's system architecture operates. System configuration settings manage everything from file protocols to core visual styles. While the settings panel is designed to help manage project settings, you may also open the system file in a text editor and update the settings directly. It is recommended that you configure your system for your development environment, then make a copy with custom setting to to be used with your final, hosted game build. Some settings well need to be customized specifically for a web-hosted presentation of a Shepherd game. Settings are as follows:

language

The language key to access while selecting game texts.

mediaBaseURL

An optional absolute path to the game's media library folder ("lib/"). By default, Shepherd loads media from it's local "lib" folder. You only need to specify a value for this field if you want to load media from a remote library address. This setting may define an absolute URL, such as "http://www.mymediaserver.com/lassiegame/lib/". Note: if you are loading media from a different server than where the game is running, then you will need to configure a Flash "crossdomian.xml" policy file to allow cross-domain loading. See Flash documentation on how to configure cross-domain policies.

---

xmlBaseURL

An optional absolute path to the game's XML library folder ("xml/"). By default, Shepherd loads XML from it's local "xml" folder. You only need to specify a value for this field if you want to load XML from a remote address. This setting may define an absolute URL, such as "http://www.mymediaserver.com/lassiegame/xml/". Note: if you are loading media from a different server than where the game is running, then you will need to configure a Flash "crossdomian.xml" policy file to allow cross-domain loading. See Flash documentation on how to configure cross-domain policies.

---

allowXMLCache

Values: "1" (TRUE) or "0" (FALSE)
Allows browser caching of XML files. When caching is enabled (true), the browser will retrieve XML files from its local cache if possible. This setting should be set to false (0) within your development environment where you are continually generating and reviewing new XML data. However, this value is best set to true (1) when deploying a finished game for online viewing. Enabling browser caching will speed up the player's experience by eliminating redundant file loads. It will also save you bandwidth and server requests.

---

staticWindow

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the Shepherd player window is allowed to automatically resize to game dimensions. Within the Shepherd development environment, this setting should be FALSE ("0") to allow the Shepherd player to dynamically scale its window to the dimensions of your project. When posting your finished game online, change this setting to TRUE ("1") and then configure window dimensions through Flash embed code. Using a static window in your online presentation will streamline game startup and allow you to specify custom display dimensions within your Flash embed code (ie: have a 1024x768 game scale down to display within an 800x600 window).

---

fileIOType

Values: "sharedObject" or "server"
Specifying "sharedObject" will utilize Flash's native cookie system to read and write data on the local file system. While SharedObjects are easy and universal, they do have security limitations imposed by the Flash player, so may throw security warnings when exceeding preset file size limitations. SharedObjects persist on a local file system, so saved games will be localized to the system on which they were saved. However, SharedObjects may not be [easily] transported between systems or user accounts.

Specifying "server" will instruct the Lassie Player to utilize a web server for file management. This allows files to be written by a server technology like PHP.

---

fileIOService

Specifies the path to Shepherd's server-side file management script. This setting is only used when <fileIOType> is set to "server". Shepherd projects include a simple file service script located at "save/fileio.php". Your project will need to be running within a PHP environment to utilize that file service. If you are running Shepherd on a server technology other than PHP, you may port the default file service script over to your technology of choice. Also note – when using a server to manage files, saved games will be centralized rather than localized to individual computers.

---

tweenEngine

Values: "tweener" or "tweenLite"
This value specifies a tween engine that the Lassie Player will use for basic interface transitions (fades in/out, auto-scrolling, and tweens performed using the <layerTween> command). The Lassie Player includes two third-party tween engines to choose from: GreenSock's TweenLite and the Tweener engine. Choosing a tween engine for your project is extremely important because you may face licensing restrictions if you choose to use TweenLite. Here's what you need to know:

Tweener is an open-source AS3 tween engine released under the MIT license, which means that it is unquestionably free for use on any project. Unfortunately all tween engines are not created equal, and Tweener's performance is only mediocre.

TweenLite is a much smoother and more efficient tween engine than Tweener. You'll generally see an obvious visual improvement in transitions when using TweenLite instead of Tweener. However, TweenLite has a licensing policy which requires fees for commercial use. You (the developer) are responsible for reviewing the TweenLite legal policy to assess if your project qualifies for free use. If you are not within the legal use-rights of TweenLite, then you MUST either secure a license to TweenLite or else configure Shepherd to use Tweener.

---

invenMode

Values: "static", "dynamic", or "dynamicItem"
Specifies the display style of the inventory panel. By default this value is "static", meaning that the inventory will display as a static fixture on screen at the coordinates that it appears within the "ui.swf" file.

Setting the inventory mode to "dynamic" will enable show/hide behavior for the inventory display. A dynamic inventory will be hidden by default, and will only show when triggered using an <inventory display="X"/> XML script command. You may attach inventory display triggers to buttons and keystrokes using Shepherd's XML API. Your display triggers must provide a means for the player to reveal the inventory display. Once the inventory has been revealed, it will automatically hide itself again when the player clicks the screen outside of the inventory bounds, or rolls the cursor out of the inventory bounds.

Setting the inventory mode to "dynamicItem" produces almost the same effect as the "dynamic" setting. The only difference between "dynamic" and "dynamicItem" is how inventory roll-out is handled. In "dynamic" mode, the inventory will be hidden any time that the cursor rolls out of bounds of the inventory display. In "dynamicItem" mode, the inventory will only hide when the cursor *with an item tooltip* rolls out of bounds of the inventory.

---

invenShow

The time (in seconds) over which the inventory's reveal animation will play. This setting is only used when <invenMode> is set to "dynamic" or "dynamicItem".

---

invenHide

The time (in seconds) over which the inventory's hide animation will play. This setting is only used when <invenMode> is set to "dynamic" or "dynamicItem".

---

actionMode

Values: "static" or "dynamic"
Specifies the interaction mode used by the action selector UI component. The action selector can behave as a "dynamic" verb-disc style selector, or as a "static" console of actions to select from. There is a radical difference in game mechanics between these two modes, so careful planning and consideration should be made up front to decide which mode will be implemented within your project.

Setting the <actionMode> to "dynamic" will hide the action selector UI component by default. The player can then access the action selector UI by clicking and holding on an interactive object, at which time the action selector is revealed, allowing the player to select an action. In this mode, object interactions are contextual to each specific object. For example, let's say that you have a button in your action selector labeled with an icon of a mouth. When interacting with an NPC, this mouth action may be contextualized as "Talk to

NPC", while interacting with an apple may be contextualized as "Bite apple". All puppets and inventory objects within Shepherd allow customization of action titles for that specific object. However, those custom titles are only utilized in this dynamic action selector mode. The advantage of this mode is that it allows a core set of interactive concepts adapt themselves to individual objects. The disadvantage of this mode is that it's generally not as intuitive to use because the game controls are hidden.

Setting the <actionMode> to "static" will place the action selector UI component permanently on screen at its coordinates from "ui.swf". The player interacts with objects by first selecting an action from the selector console, then selecting an object to interact with. In this mode, actions are NOT contextualized to individual objects. Each action has a static contextual title that is read from the current default dialogue set. The advantage of this mode is the obvious control scheme displayed on screen. Also, this action mode works well on touch devices. The disadvantage here is that a static action set generally requires a larger collection of actions (ie: look at, talk to, use, take, push, pull, etc) to allow the player to intuitively assemble contextualized actions.

---

actionShow

The time (in seconds) over which the action selector's reveal animation will play. This setting is only used when <actionMode> is set to "dynamic".

---

actionHide

The time (in seconds) over which the action selector's hide animation will play. This setting is only used when <actionMode> is set to "dynamic".

---

skipControl

---

skipDiaControl

---

subtitleShow

The time (in seconds) over which the dialogue subtitle's reveal animation will play.

---

subtitleHide

The time (in seconds) over which the dialogue subtitle's hide animation will play.

---

subtitleOverhead

The vertical space (in pixels) between the subtitle display and the puppet that is delivering dialogue. This space is calculated from the bottom of the subtitle rect to the top of the speaking puppet's rect.

---

subtitlePadding

The minimum space (in pixels) allowed between the subtitle display and all edges of the screen. Padding is tested on all sides of the subtitle, and the display will be shifted as necessary to keep the entire subtitle visible on screen with the minimum padding surrounding it.

---

contextOff

Specifies the inactive (default) color of the game's contextual title text. This color value is specified as a hexadecimal value with a "0x" prefix (ie: "0xFFFFFF" would be white).

---

contextOn

Specifies the active/execute color of the game's contextual title text. The game's contextual title will flash this active color when a game action is executed. This color value is specified as a hexadecimal value with a "0x" prefix (ie: "0xFF0000" would be red).

---

treeOff

Specifies the inactive (default) color of a dialogue tree's menu text. This color value is specified as a hexadecimal value with a "0x" prefix (ie: "0xFFFFFF" would be white).

---

treeOn

Specifies the active (mouse hover) color of a dialogue tree's menu text. This color value is specified as a hexadecimal value with a "0x" prefix (ie: "0xFF0000" would be red).

---

curtainColor

Specifies the default color of the game's curtain (fade in/out transition). This color value is specified as a hexadecimal value with a "0x" prefix (ie: "0x000000" would be black).

---

menuSubtitle

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu's settings panel should include a "subtitles enabled" option. This value should only be TRUE ("1") if your game includes dialogue voice-overs that can play without subtitles.

---

menuSubtitleSpeed

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu's settings panel should include a "subtitle speed" option.

---

menuVoice

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu's settings panel should include a "voices enabled" option. This value should only be TRUE ("1") if your game includes dialogue voice-overs.

---

menuFullscreen

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu's settings panel should include a "fullscreen display" option.

---

menuGraphicsQuality

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu's settings panel should include a "graphics quality" option.

---

menuQuit

Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu should include a "quit" button option. Even when enabled, a quit button will only display if the current Flash runtime supports quitting (ex: you cannot quit a game running within a web browser using the Flash plugin).

---

menuHelp

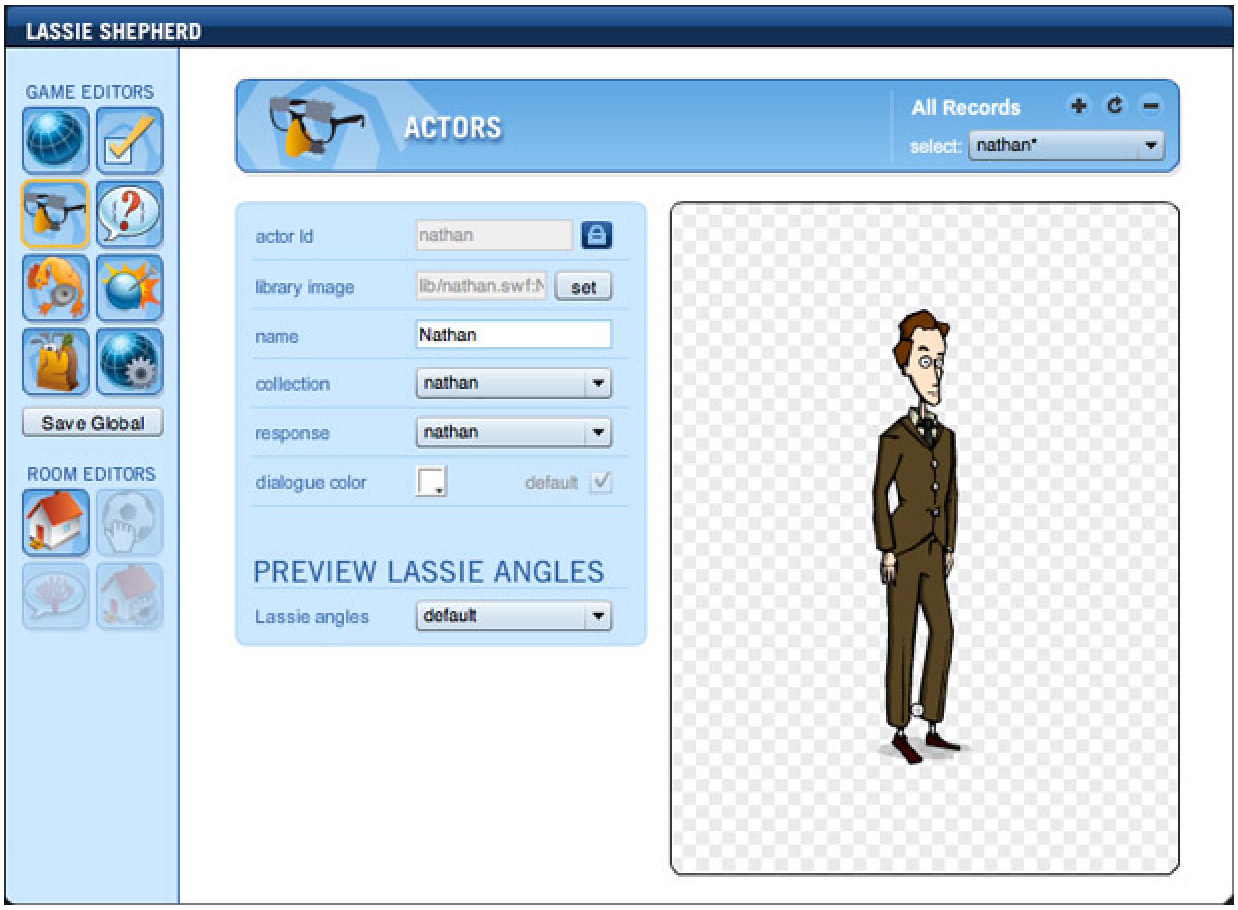
Values: "1" (TRUE) or "0" (FALSE)
Specifies if the game menu should include a "help" button option.

---

menuHelpText

Specifies a block of text to display on the game menu's help screen. For multi-lingual content, define each piece of language specific content within a language node (ex: <en>English content.</en><de>Deutscher inhalt.</de>).

The Actor Panel



select: nathan*

actor Id	nathan
library image	lib/nathan.swf:N
name	Nathan
collection	nathan
response	nathan
dialogue color	default ✓

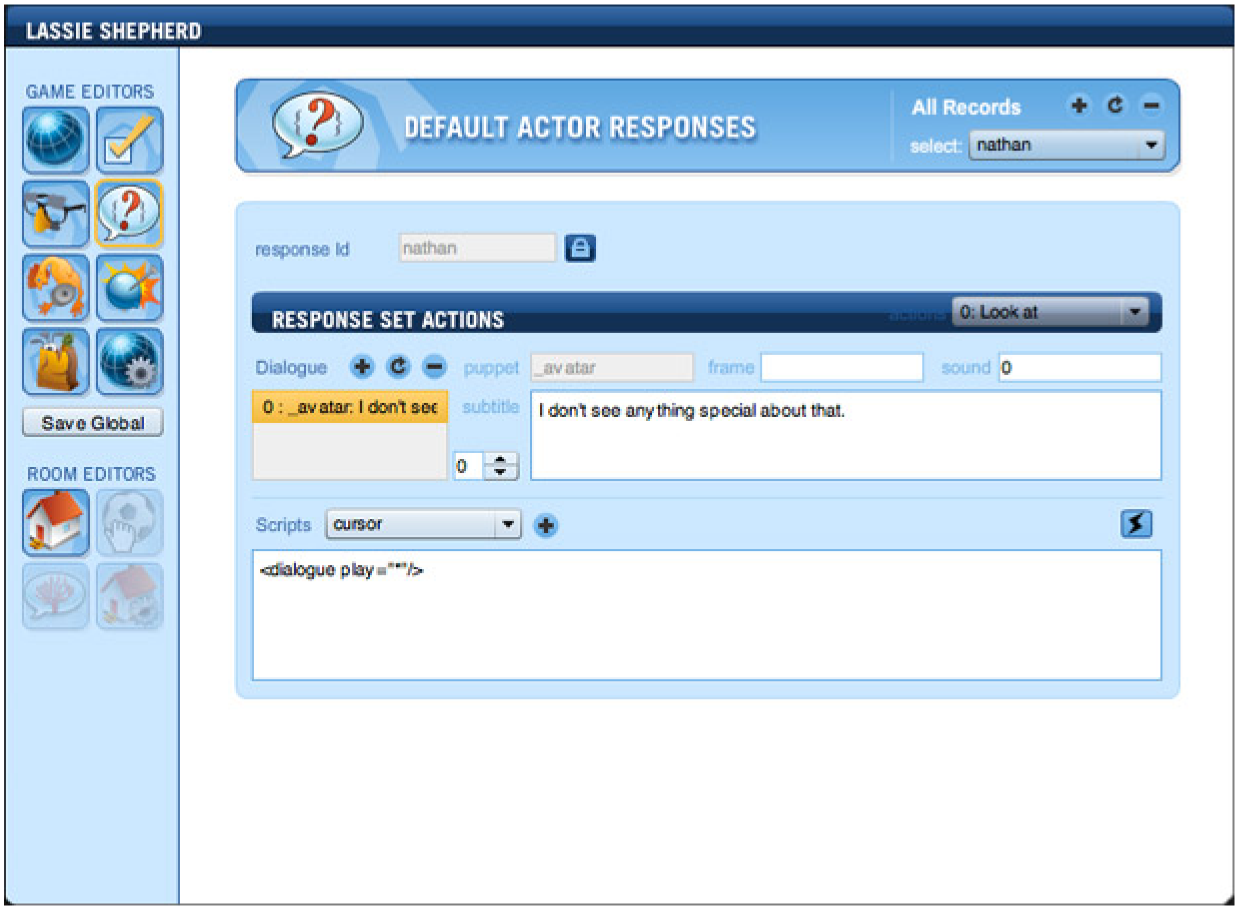
PREVIEW LASSIE ANGLES

Lassie angles	default

Here you can configure your game's playable characters. Only one actor can be active at a time within the Lassie Player, and that actor's setting will be applied to any given room's avatar layer. Actors may only utilize art assets that are contained within a global game library (import global libraries within the Setup panel).

When creating a MovieClip for an actor, use the same timeline configuration that was used in legacy versions of Lassie. A playable character's MovieClip hierarchy is exactly the same as it was in Lassie Director and LassieAS, with the exception that basic frame label states have changed to "rest", "move", and "speak". After applying a MovieClip display to an actor, you can preview the actor's animations using the "Preview" selector. All frame label presets that the Lassie Player will attempt to display are listed in the preview selector.

The Default Response Panel



select: nathan

response Id: nathan

actions: 0: Look at

0 : _avatar: I don't see

puppet: _avatar

frame:

sound: 0

subtitle: I don't see anything special about that.

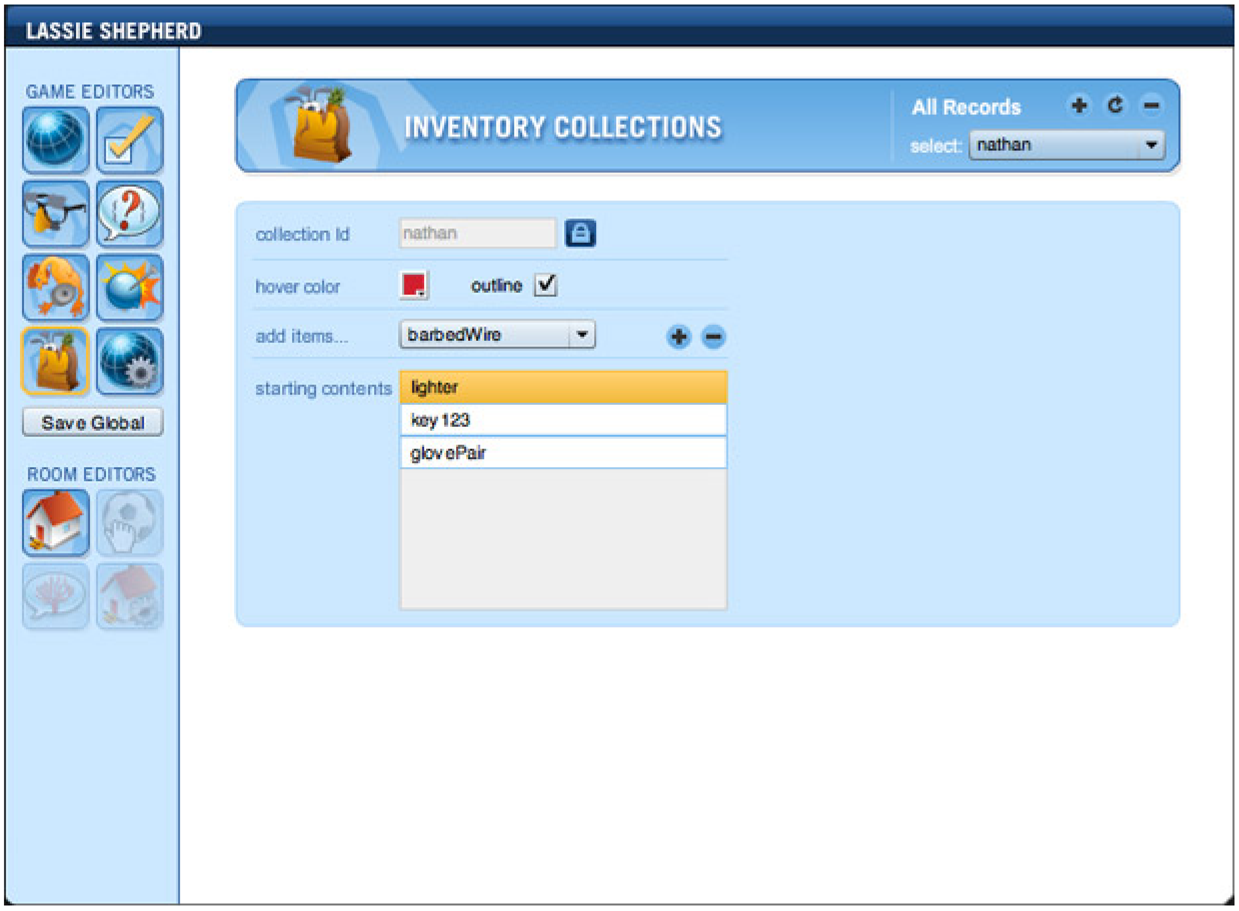
Scripts: cursor

```
<dialogue play=""/>
```

This panel defines sets of default dialogue responses that an actor will recite upon interacting with an object that has no custom response. Default response sets are assigned to actors. This allows each actor to use a unique set of default responses.

Default responses are single-line statements. If there are multiple statements defined for a single default interaction response, then a single statement will be selected at ransom from the interaction's response list.

The Inventory Collection Panel



select: nathan

collection Id: nathan

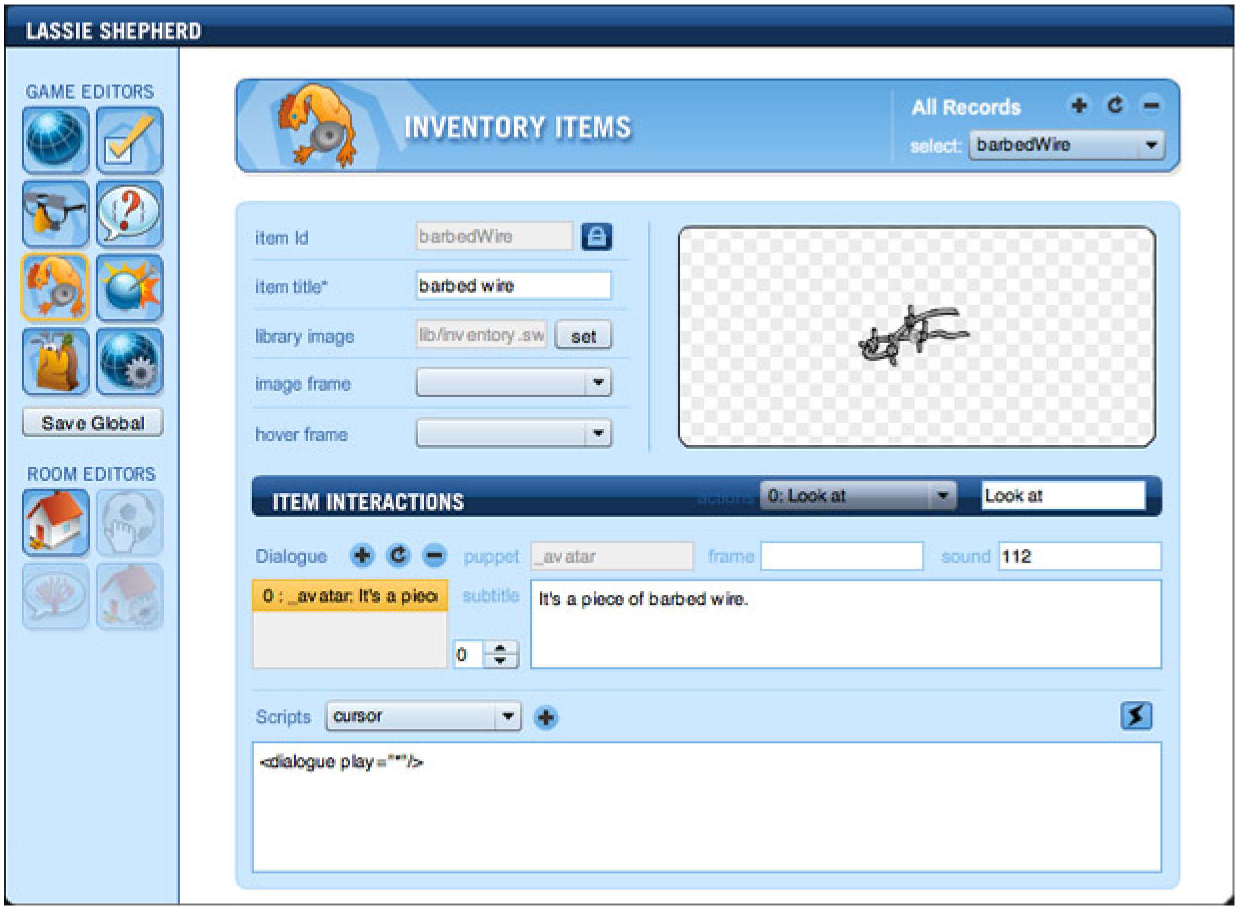
hover color: 🟥 outline ✓

add items...: barbedWire

starting contents:
- lighter
- key 123
- glov ePair

This control panel allows you to define inventories and their contents. Multiple inventory collections can be created, each containing their own unique set of inventory items. An inventory collection is assigned to an actor. This allows each actor to maintain their own unique collection of inventory items.

The Inventory Items Panel



select: barbedWire

item Id barbedWire

item title* barbed wire

library image lib/inventory.sw set

image frame

hover frame

actions 0: Look at Look at

puppet _avatar frame sound 112

0 : _avatar: It's a piece

subtitle It's a piece of barbed wire.

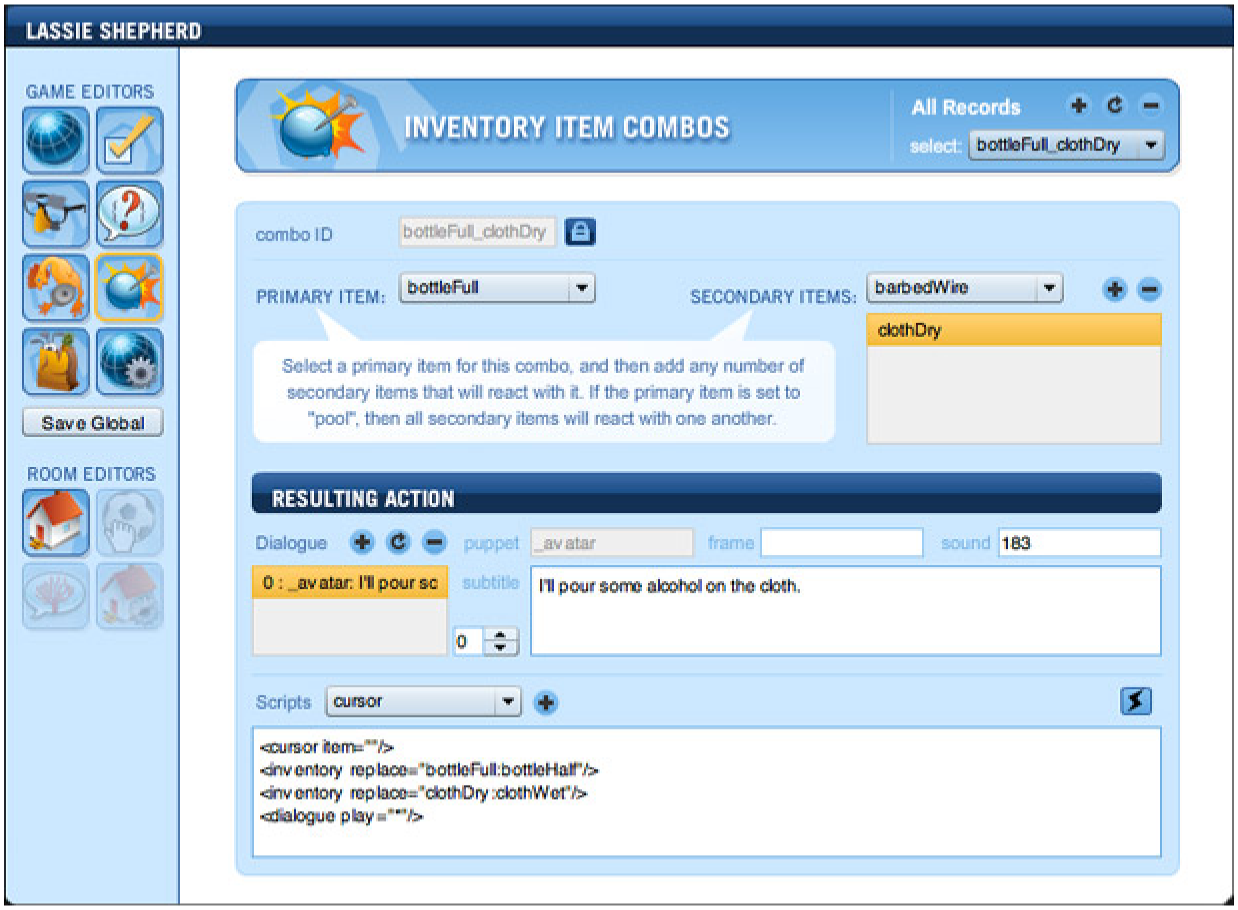
0

Scripts cursor

```
<dialogue play=""/>
```

This panel defines individual inventory items which can be inserted into inventory collections. Like actors, inventory items can only utilize artwork contained within a globally-linked media library. Specify global media libraries to import within the Setup panel.

The Item Combo Panel



select: bottleFull_clothDry

combo ID: bottleFull_clothDry

PRIMARY ITEM: bottleFull

SECONDARY ITEMS: barbedWire

clothDry

Select a primary item for this combo, and then add any number of secondary items that will react with it. If the primary item is set to "pool", then all secondary items will react with one another.

Dialogue: 0 : _avatar: I'll pour sc

puppet: _avatar

frame:

sound: 183

subtitle: 0

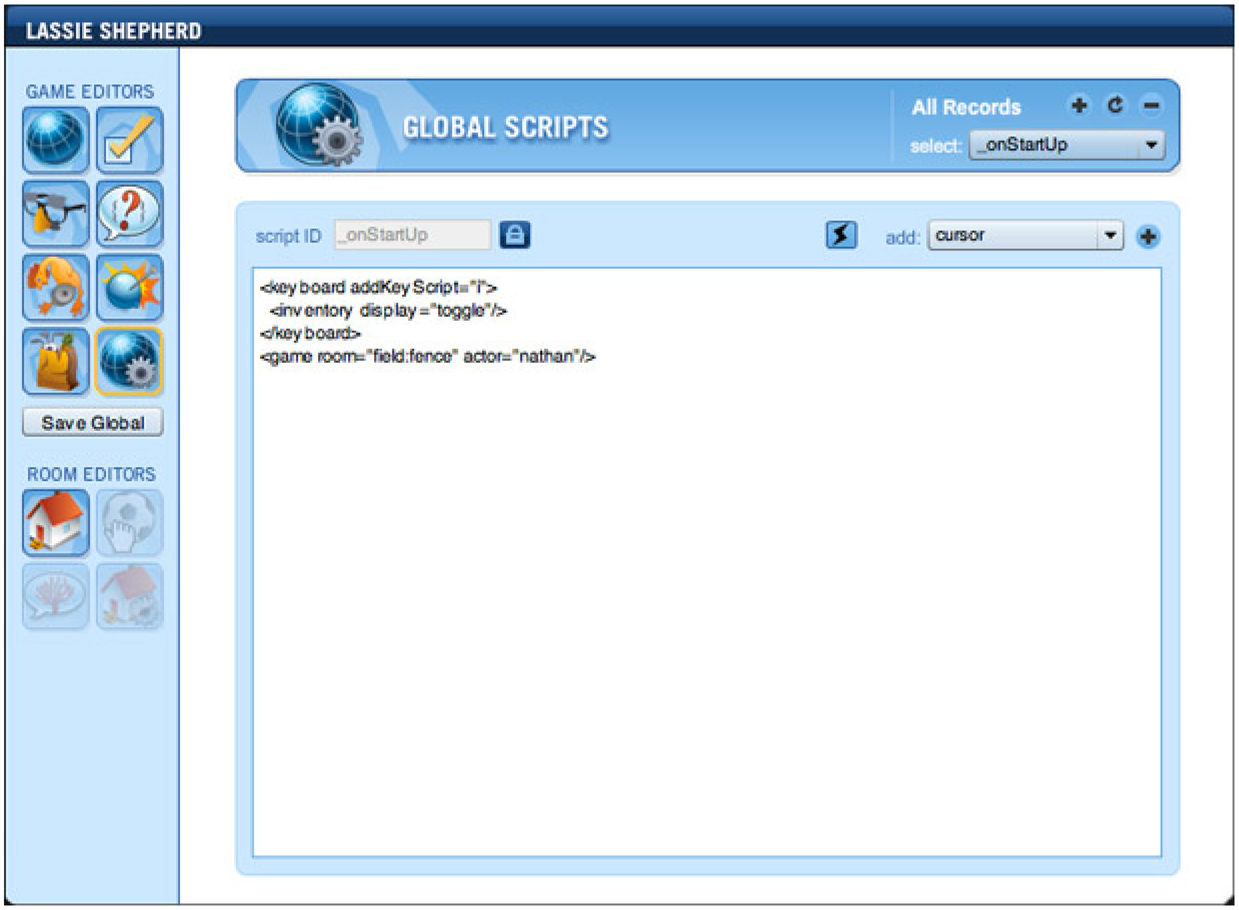
I'll pour some alcohol on the cloth.

Scripts: cursor

```
<cursor item=""/>
<inventory replace="bottleFull:bottleHalf"/>
<inventory replace="clothDry:clothWet"/>
<dialogue play=""/>
```

This panel lets you define inventory item combinations that will produce a result when used together. Individual inventory items are defined within the Inventory Item panel. Within the Item Combo panel, a single primary item and multiple secondary items can be specified. A primary item will react with any item within the pool of secondary items. When the primary item is set to "pool", then all secondary pool items will react to one another.

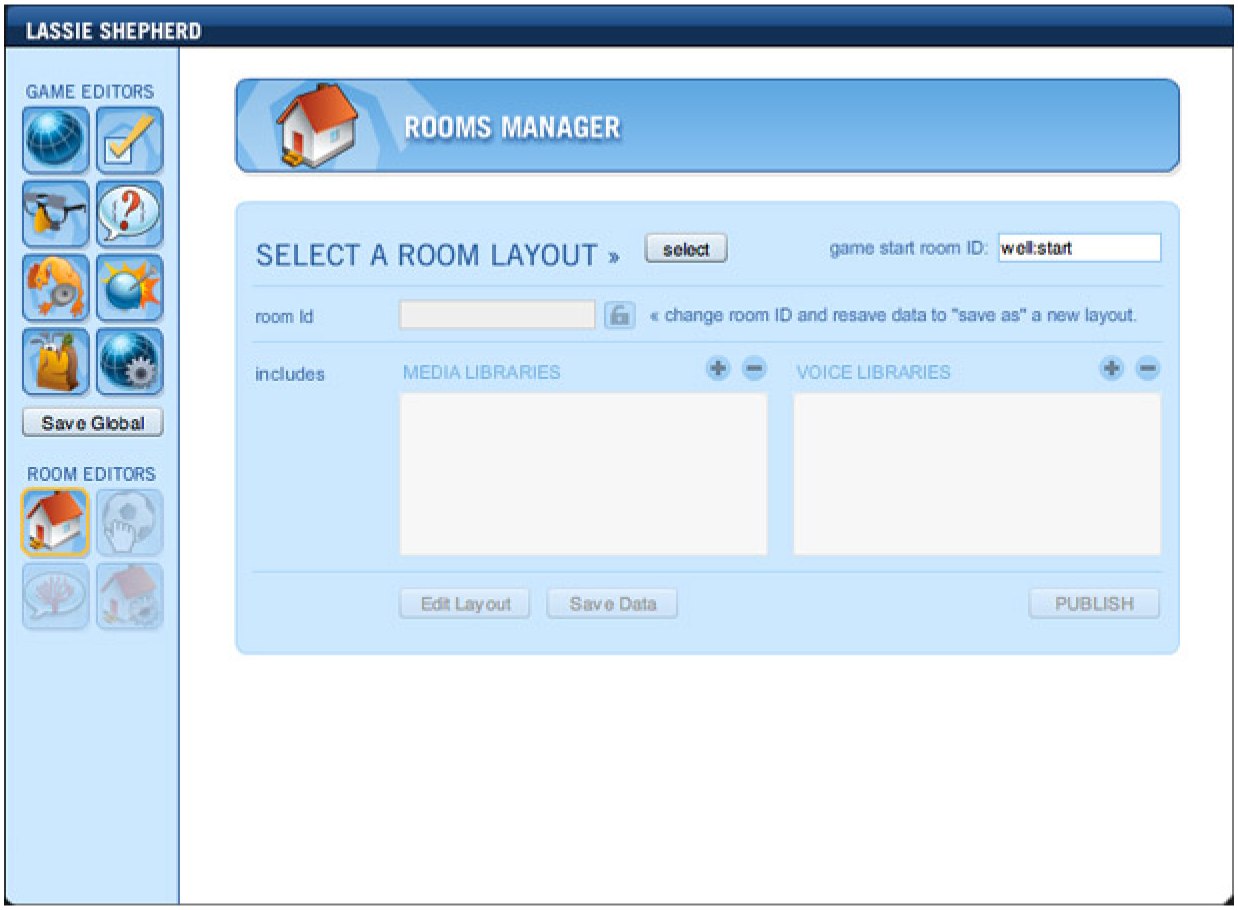
The Global Script Panel



```
<keyboard addKeyScript="i">
  <inventory display="toggle"/>
</keyboard>
<game room="field:fence" actor="nathan"/>
```

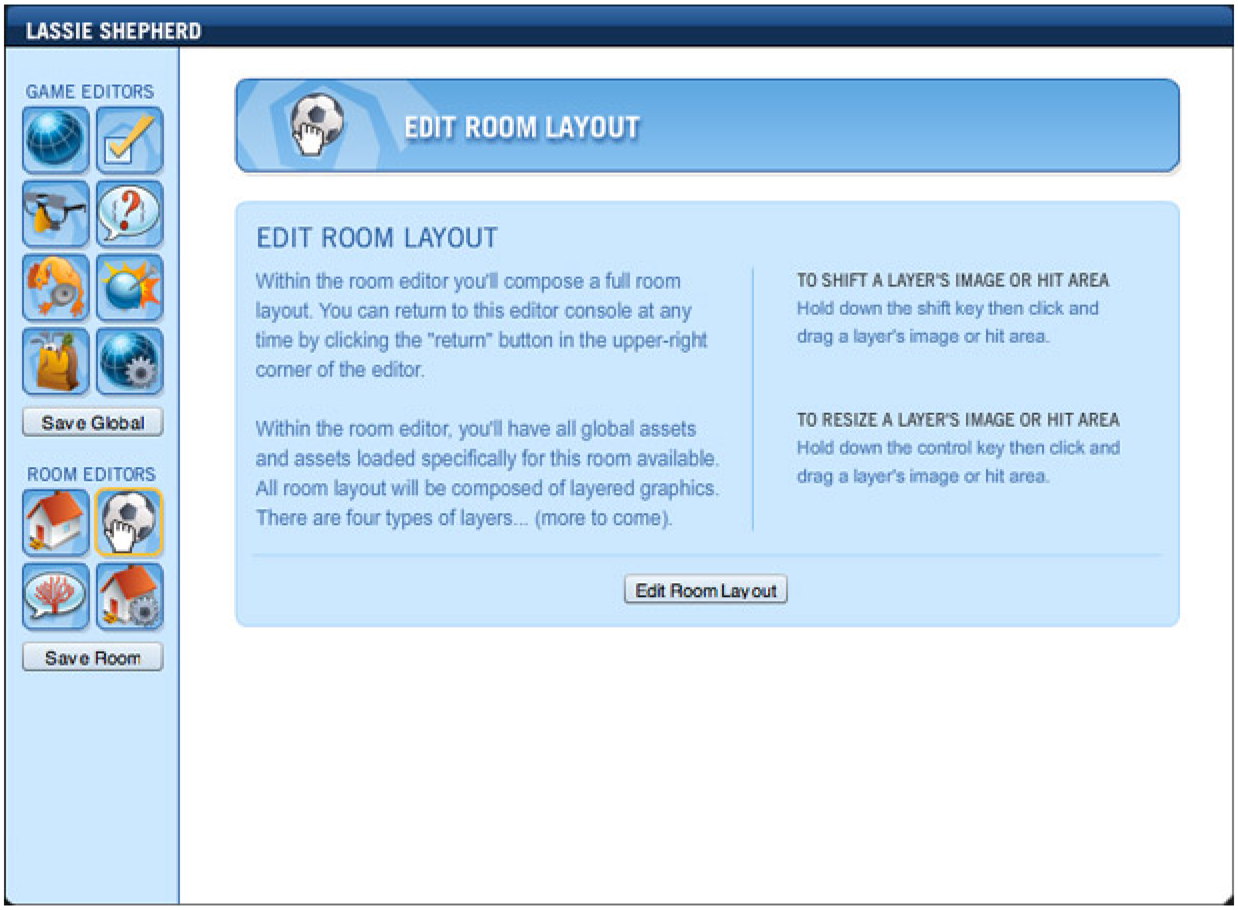
This panel allows you to compose XML scripts that can be called from anywhere within your game. A global script is referenced by ID and will perform all actions specified within its XML list. Global scripts are intended to perform universal actions, so should NOT include any scripts that are written for a specific room. Use the Room Script panel to compose room-specific scripts. When composing XML scripts, use the language reference documentation in the Shepherd XML Script document. Be sure to click the validate button (labeled with a lightening bolt) to check your XML syntax each time you edit a script.

The Rooms Panel



This panel allows you to create, save, publish, and rename room layouts. Once you've created a room, use the library list to import external SWF libraries for use within this layout. These assets will be loaded during game play upon the player first reaching this room layout. For instructions on creating external media SWFs, see the Library Tutorial document. The Rooms Panel will also allow you to define the starting room of your game (enter a room ID). And of course, always remember to publish your changes. The Lassie Player application will not be able to load a room layout until it has been specifically published as XML.

The Room Editor Info Panel



EDIT ROOM LAYOUT

Within the room editor you'll compose a full room layout. You can return to this editor console at any time by clicking the "return" button in the upper-right corner of the editor.

Within the room editor, you'll have all global assets and assets loaded specifically for this room available. All room layout will be composed of layered graphics. There are four types of layers... (more to come).

TO SHIFT A LAYER'S IMAGE OR HIT AREA
Hold down the shift key then click and drag a layer's image or hit area.

TO RESIZE A LAYER'S IMAGE OR HIT AREA
Hold down the control key then click and drag a layer's image or hit area.

This panel just provides some basic instruction on how to use some of the hidden room editor controls, then allows you to launch the room editor. Be sure to review this panel once, then refer back to it if you ever forget what keyboard/mouse combination offer advanced editor controls.

The Dialogue Tree Panel

This control panel allows you to build an interactive dialogue tree. The tree is saved as a component of the room, and must reference room-native layers as the interactive layer targets. In their most basic form, dialogue trees are composed of two structures: **topics** and **tiers**.

Topics are individual dialogue exchanges between puppets. Topics are each a standard Lassie action composed of a dialogue list and a script.

Tiers are groups of topics which compose the tree hierarchy, or in short – a tier is a single dialogue menu, where each nested topic is an individual menu item. Tiers are chained together to compose the tree of menus.

Working With Topics

Topics are the primary content of a dialogue tree, and are individually targeted and controlled within the Lassie Player through ID reference. That said, the ID name assigned to each topic is extremely important, and it is essential that no two topics within a tree share the same ID. Tier groupings are not recognized while indexing topic IDs, so two topics with the same ID will conflict even if they are in different tiers. Therefore, it is extremely important to make sure that all topics within a tree maintain a unique reference ID; but don't panic... this is done automatically for you.

By default, all topics are created with a unique ID generated by an incrementing counter. Assuming that you never change the default ID that was initially assigned to the topic, then you're guaranteed to never encounter a naming conflict. Therefore, it is suggested that you NOT change a topic's ID unless you plan to specifically target that ID with a custom behavior (at which time, assigning your own meaningful ID value would be extremely useful). When selecting your own ID value for a topic, just make sure to pick a value that is not used anywhere else in the entire tree. Note that a topic's ID has nothing to do with its indexing order. While a topic's default ID does include a number, that number has nothing to do with the topic's position within the parent tier's topic list.

Topic Status Actions

Each topic is assigned a set of "status actions" which instruct the tree what to do after playing the topic. There are two status actions assigned to each topic: one to govern topic configuration, and the other to govern tier configuration.

Topic Actions:

- **No Change** : just line the name implies, the trigger topic is left unchanged will remain visible within its tier menu and will be available for replay.
- **Hide After Play** : hides the trigger topic after it has finished playing by setting the its "enabled" status to false. The trigger topic will no longer appear within its parent tier menu. To reactive a hidden topic, restore the topic's enabled status to true using the <tree> command.

- **Reveal** : leaves the trigger topic unchanged and reveals one or more new topics by setting their "enabled" status to true. This assumes that the new topics were disabled by default. When targeting topics to reveal, cite the new topic IDs within a comma-seperated list ("new1,new2,new3").
- **Hide then Reveal** : hides the trigger topic, and then reveals one or more new topics by setting their "enabled" status to true. This assumes that the new topics were disabled by default. When targeting topics to reveal, cite the new topic IDs within a comma-seperated list ("new1,new2,new3").

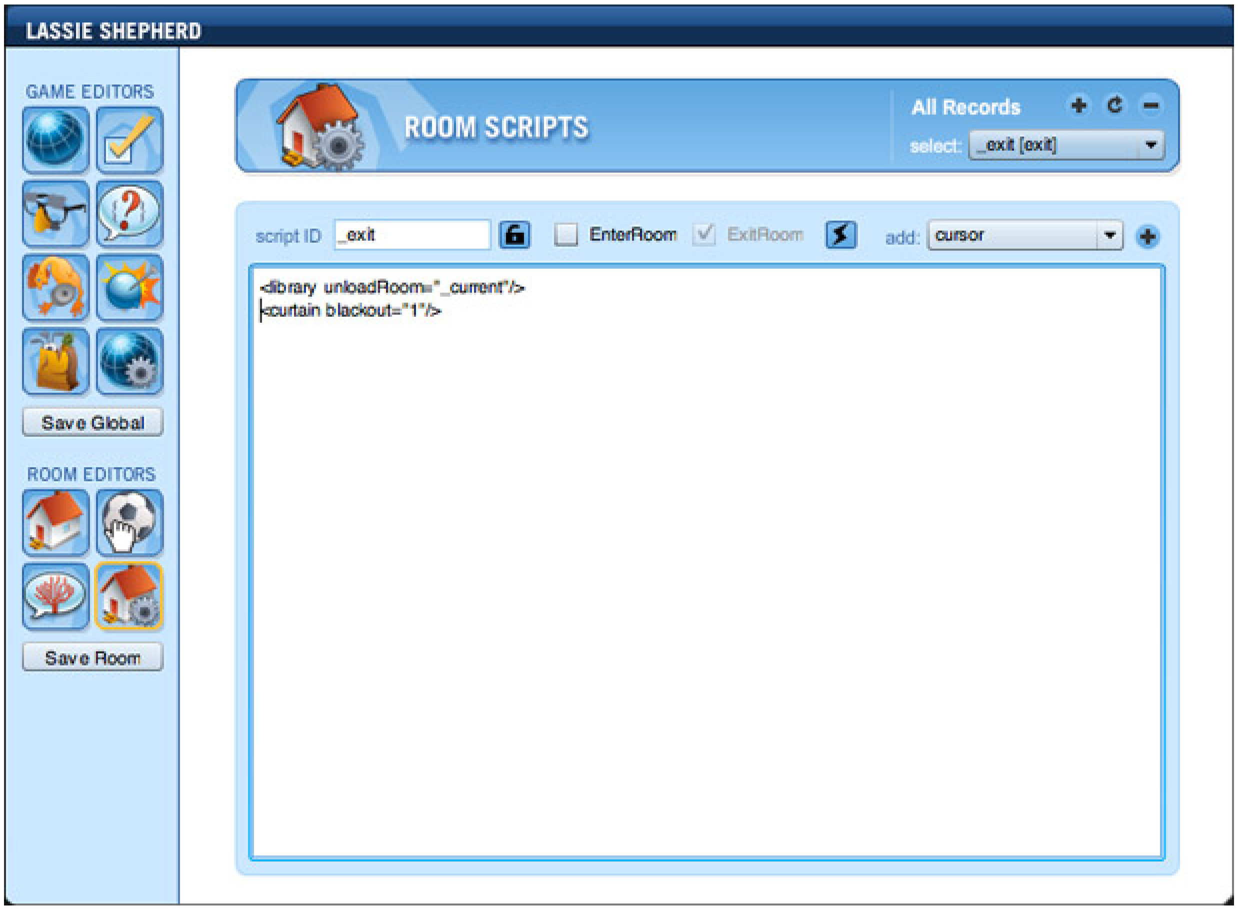
Tier Actions:

- **No Change**: just as the name implies, no change is made to the current tier. The same tree menu will be introduced after the trigger topic has finished playing.
- **Next** : steps ahead in the tree to the next tier that branches from the trigger topic. Each topic may spawn a single tier, so there is unlimited forward potential while building a tree structure.
- **Previous** : steps back to the previous tier from which the trigger topic extended. All topics may step back to their parent tier save for topics within the root tier, which have no parent. Attempting to step back from a root topic will exit the conversation.
- **Exit** : exits the dialogue tree and restores normal Lassie Player gameplay.
- **Jump** : jump is an advanced tier action where the trigger topic can jump the conversation to an unrelated tier in a separate branch of conversation. When specifying a jump, you'll need to assign the target tier a tier ID (that is the only time tier ID's are needed), then reference that tier ID as the jump target.

Working with Tiers

By contrast to topics, tiers are extremely basic and don't do much within a dialogue aside from defining topic groups. While tiers may also be assigned an ID name, the ID is not needed unless you plan to target the tier with a "jump" status action (see "status actions" section above). Otherwise, the tier ID has no purpose and may be left blank.

The Room Script Panel



script ID: _exit

```
<library unloadRoom="_current"/>
<curtain blackout="1"/>
```

This panel is exactly like the Global Script panel, except that it manages scripts that are native to the currently loaded room layout. You may only call room scripts from within the room itself, and only while the room is active within the game. The room script panel has two native scripts included for "enter room" and "exit room". These two scripts are called upon load and unload of the room layout. By default, there are enter and exit scripts configured for each room which contain screen fade in/out commands.

Introduction to Creating Room Layouts

Before diving into the core controls of the room layout editor, let's establish a high-level understanding of the basic concepts at work within a room layout. The following are some of the core concepts you'll need when getting into Shepherd room layouts.

Layers

Layers are the basis for all room layouts. Layers are simply graphical objects arranged and stacked in such a way to compose a 2D environment with foreground, mid-ground, background, and interactive elements. Within the Shepherd Editor, the term "layer" refers loosely to any graphical object placed within a room layout.

A layer is composed of two elements: an image and a hit area. A layer's image is a custom MovieClip symbol that you select for it. The layer's hit area is a rectangle or ellipse which specifies its interactive bounds. Some layers may be configured to use only an image or only a hit area. When a layer has only a hit area, it will exist on screen as an invisible interactive hotspot.

States

Layers include an additional dimension known as "states" which each define an appearance and set of behaviors for the layer. Take a door for example -- a single "door" layer would include two states: an opened state and a closed state, each with their own graphics and interactivity. A single layer may have an unlimited number of states. A layer's state may be changed at any time within the game runtime to evolve the game environment.

Floating Layers

Floating layers, or "Floats", are an important concept within Shepherd's layering theory. Often known as "walk-behinds", floating layers are Lassie's system for allowing a layer to pass both in front of and behind another layer. Think of a lamp post standing in the center of a room. Your character may pass in front of the lamp post, or walk further back in the room's implied depth to then pass behind it.

Floating layers are not locked at a fixed position within a room's layering stack. Instead, floating layers may push down in the layer stack to draw below other layers. This change in layering order is triggered by the two floating layers' spatial proximity.

Unlike the classic "z-index" system, floating layers will only shift downward in the layer stack from their native depth (which is established within the Shepherd Editor). Floating layers will arrange themselves within the layering stack based on their relative Y-positions. A floating layer with a lesser Y-position (higher on screen) will draw below another floating layer with a greater Y-position (lower on screen) at a lower native depth. The layer with a lower native depth will never shift upwards in the stack to draw above the layer with a higher native depth. This float mechanism allows for easy creation of a complex dynamic layering scheme within a room. A room's avatar layer automatically has floating enabled, so just activate floating on a few objects behind it in the Shepherd Editor's layer stack, then wander around the interactive room and watch how the avatar moves up and down within the layering stack! Note: all relative layer positions are calculated using each layer's registration point, so make sure to position a layer's registration in an appropriate place for the layer image.

Also note, advanced float capabilities are possible using the <layerSprite> XML command. For more information, see the Lassie XML Scripting documentation.

Matrix Filters

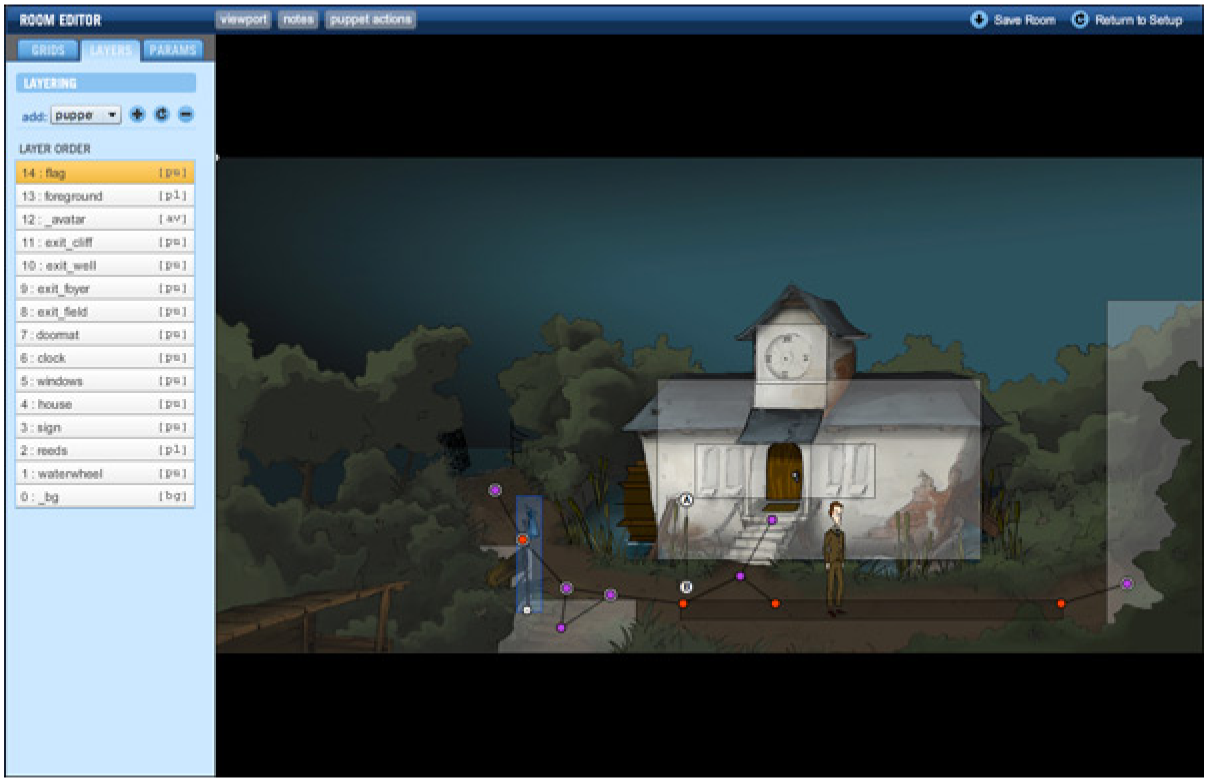
Matrix filters are a quick and easy way to apply grid-based visual effects to one or more layers. Matrix filters can be used to shift a layer's scale, color tint, rate of movement, or blur across a grid. This feature is most commonly used to set up implied depth scaling, where a character may appear very small off in the distance, then grow as they move toward the camera in implied 2D space.

The Shepherd Room Editor Controls

Shepherd's room layout editor provides a graphical interface for composing room layouts. There are many detailed controls within the room layout editor, most of which will pop a help balloon describing their function when you roll over the control's label text. Refer to those help labels for additional information beyond the documentation included here.

Layering Panel

This panel provides a list of all layers currently in the room, arranged by their graphical stacking order. You may add, remove, and rearrange layers within this list as needed. To rearrange layering order, just click and drag an item within the layers to shift its layering position. By default, all rooms start with a background layer that cannot be deleted. A room layout may add ONE avatar layer, and an unlimited number of puppet and plane layers. These four layer types are discussed in detail below.



Layers are the basis for all room layouts. Layers are simply graphical objects arranged and stacked in such a way to compose a 2D environment with foreground, mid-ground, background, and interactive elements. Within the Shepherd Editor, the term "layer" refers loosely to any graphical object placed within a room layout. Legacy Lassie developer will be familiar with the concept of layers a "sprites".

A layer is composed of two elements: an image and a hit area. A layer's image is a custom MovieClip symbol that you select for it. The layer's hit area is a rectangle or ellipse which specifies its interactive bounds. In the case of puppet and plane layers (see below), a layer may be assigned to use only an image or a hit area. When a layer has only an image, then the image's actual vector bounding is used as the hit area. When a layer has only a hit area, it will exist on screen as an invisible interactive hotspot.

While getting the feel for Shepherd layers, keep in mind the metaphor of a puppet show. Shepherd has been deliberately modeled after after this concept, because –in essence– that's what an adventure game is: digital puppeteering. That said, there several types of layers within the Shepherd infrastructure which represent different pieces within a puppet show, including the backdrop, the scenery, and the puppets themselves. All told, there are four unique type of layers which compose a room.

Background Layer

The background layer acts as a backdrop behind the room layout. This generally displays the room's primary artwork plate, and defines the scrolling area of the room. Rooms automatically include a background layer that cannot be deleted. Additional backgrounds may not be added.

Plane Layers

These are non-interactive layers. These are best used as layers of scenery within a room layout. A plane layer does not support mouse actions, making it lighter and more efficient than fully interactive (puppet) layers. In addition, plane layers have the ability to perform parallax scrolling, which is a special effect where the plane layer scrolls at an asynchronous rate against the background to create the illusion of depth. Unlimited plane layers are allowed within a room layout.

Puppet Layers

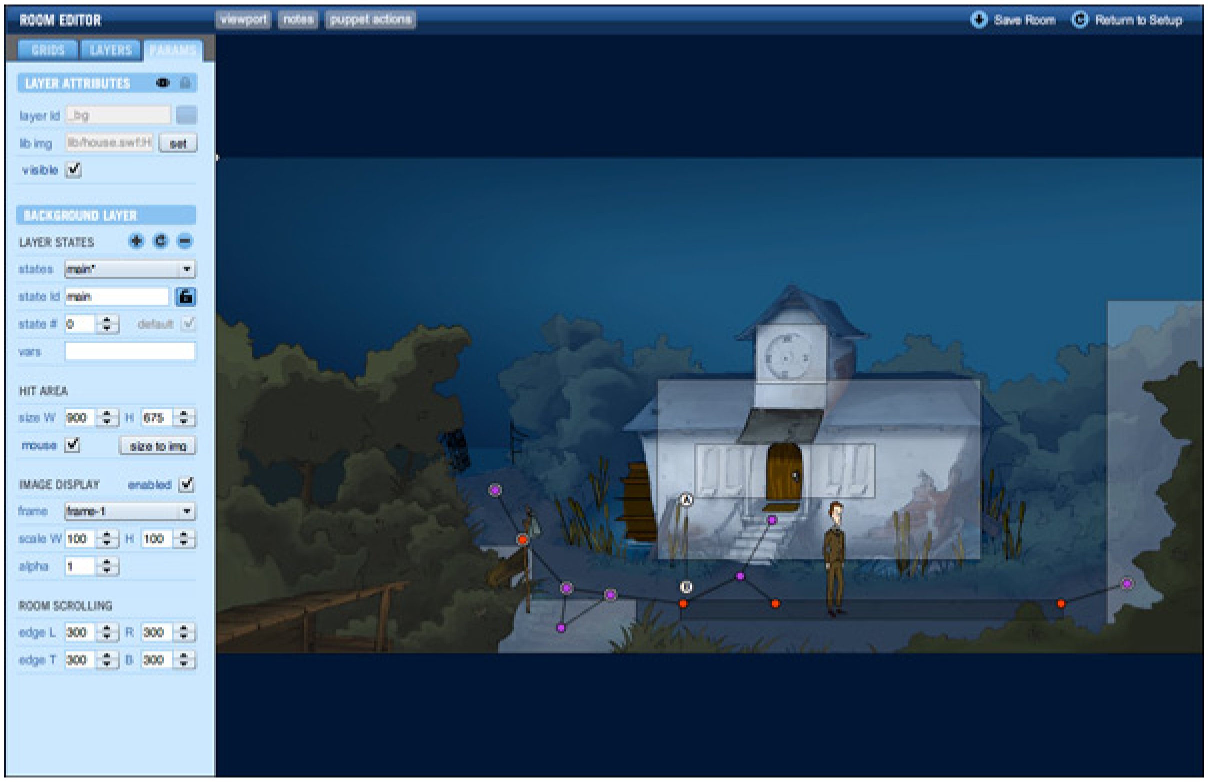
A puppet layer is a fully interactive room layer. These can be used as NPC (non-player character) elements within a room, or as basic interactive objects. A puppet layer can be targeted with mouse actions, dynamically animated using Lassie's puppet motion controls, utilize filter matrices, and define custom interactions. Puppet layers make up the core of a room's interactivity. Unlimited puppet layers are allowed within a room layout.

Avatar Layer

The avatar layer is a special instance of a puppet layer which represents the room's playable character. Lassie will dynamically configure the avatar layer at game runtime to reflect the game's current actor. The avatar layer shares all the same features as a standard puppet layer, and therefore may be targeted by any XML scripts designed to work with puppet layers. A room may have up to one avatar layer. Note that the presence of an avatar layer is NOT required. You'll generally want to omit the avatar layer for rooms with full screen content, like mini-games or cutscenes.

Background Layer Params Panel

This panel is accessed when you select the background layer within your room layout, then open the "PARAM" editor tab.



layer id _bg
lib img lib/house.swf:H
visible ✓
states main*
state id main
state # 0 default ✓
vars
size W 900 H 675
mouse ✓
enabled ✓
frame frame-1
scale W 100 H 100
alpha 1
edge L 300 R 300
edge T 300 B 300

Aside from just displaying the base graphics of a scene, the background layer's hit area rectangle defines the bounds of the room's scrolling. Other layers that extend beyond the bounds of the background layer's hit area rectangle will be cropped. Also, the background layer is unique in that it does not support embedded animations. If you attempt to include an animation within your background layer, it will be stopped during game play. This is an intentional feature of the Lassie Player built in for optimization purposes. Rather than forcing the entire bounds of the room to redraw each frame, the background layer is cached as a flat bitmap and all animated content must be placed as a layer on top of it.

States

Like most layer types, the background layer may define multiple "states". A state is a set of visual and interactive properties that the background may assume. By changing the layer's current state within the game runtime, the layer's appearance and behavior may be changed.

All layer settings are specific to a state. This allows almost any attribute of a layer to be customized on a state-by-state basis.

State - Selector

This selector lists all states defined for the currently selected layer. All layers must have at least one state. Additionally, this selector defines the order in which states are arranged on the layer. State order may be important if you plan to take advantage of the <puppet> XML script's "nextState" and "prevState" features.

State - Id

This is a unique identifier name by which the state is referenced. No two states on a single puppet may share an ID. Once you pick a state identifier, it's suggested that you lock the state Id field so that you don't inadvertently change the state ID later on. Remember, if you ever change an ID within Lassie, you will need to go through all of your XML scripts and update any references to the changed ID.

State - Index (#)

This field defines the state's numeric index within the layer's state list. When you change a state's index, you'll see its new position reflected within the state selector listing (A). State order may be important if you plan to take advantage of the <puppet> XML script's "nextState" and "prevState" features.

Variables

Planned field. Currently unused.

Hit Area

The background layer's hit area is extremely important because it defines the scrolling bounds of the room layout. Regardless of the size of the background artwork or the position of layers within the room, the room layout will be cropped to the background's hit area.

Hit Area - Size

Adjusts the width and height of the background's hit area rectangle. The rectangle's registration is locked in the upper-left corner of the screen at coordinates x-0, y-0. The shape, position, and registration of the background's hit area cannot be changed.

Hit Area - Mouse

Specifies if mouse clicks on the background layer should trigger proximity target mapping. When enabled, a random click on the room background will attempt to move the avatar as close to the click as possible while staying within valid grid bounds. When the background's mouse is disabled, the avatar will only walk to specific interactive objects.

Hit Area - Size to Image

Convenience method for automatically setting the background layer's hit area size to match the size of its artwork. After setting a background image, you can click this button to immediately match the hit area dimensions to those of the image.

Image Display - Frame

Specifies a frame within the background layer's MovieClip asset to display.

Image Display - Scale

Specifies custom scale percentages for the background image's width and height.

Image Display - Alpha

Specifies the alpha (opacity) of the background layer's artwork. Note that additional layers ARE permitted to be placed behind the background layer.

Room Scrolling - Edge Left & Right

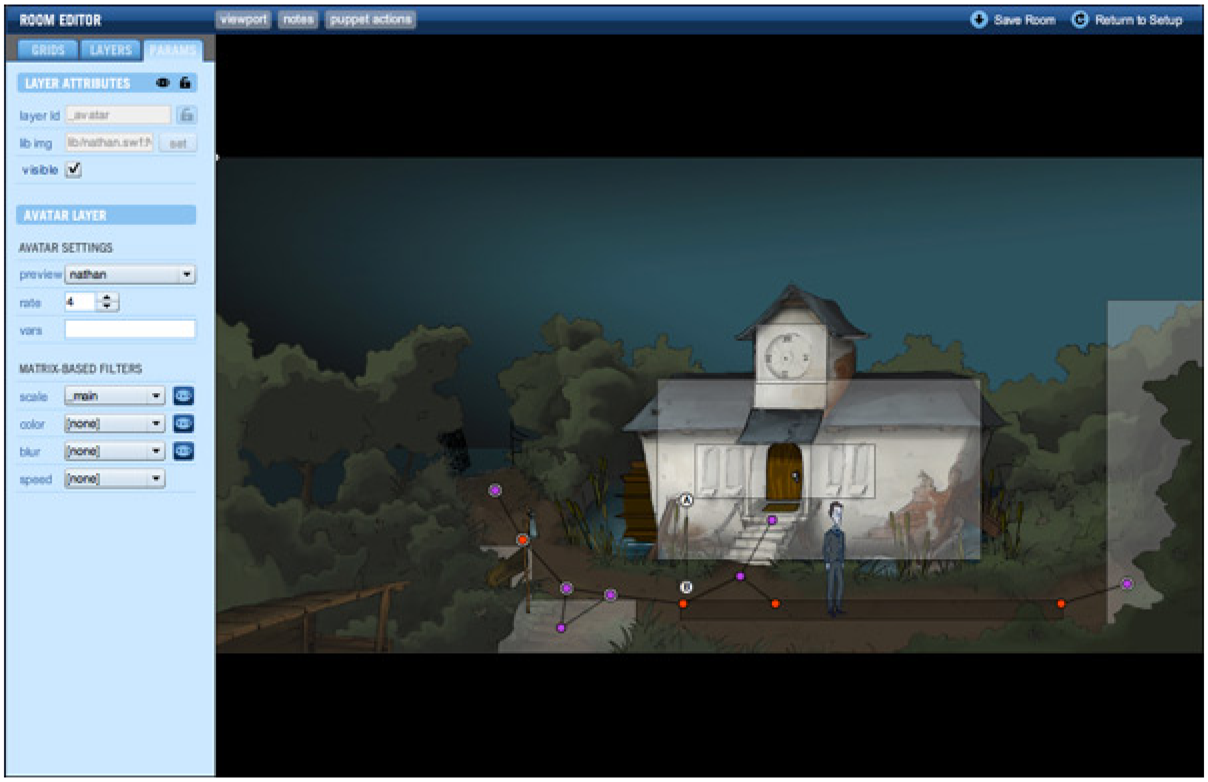
Specifies the minimum horizontal space (in pixels) allowed between the avatar layer and the left and right edges of the screen before horizontal room scrolling is invoked.

Room Scrolling - Edge Top & Bottom

Specifies the minimum vertical space (in pixels) allowed between the avatar layer and the top and bottom edges of the screen before vertical room scrolling is invoked.

Avatar Layer Parameters Panel

This panel is accessed when you select the avatar layer within your room layout, then open the "PARAM" editor tab.



layer id: _avatar
lib img: lib/nathan.swf
visible: ✓
preview: nathan
rate: 4
vars:
scale: _main
color: [none]
blur: [none]
speed: [none]

While the avatar layer is technically just a puppet layer within a room layout, it has a limited subset of properties because it's mostly configured by the Lassie player. However, you can still customize the avatar layer at game runtime using XML scripts. The avatar layer is targeted just like any other room layer, referenced with the ID of "_avatar". The avatar layer has one single state called "main" that is defined by Lassie. You may not add, remove, or rename states applied to the avatar layer.

Actor

Use this selector to apply an actor skin to the avatar layer. Actors are configured within the global editor window's "Actors" panel. An actor skin is applied to a room's avatar layer for preview purposes only. An actor profile is NOT saved as part of the room's avatar layer configuration. Instead, the avatar layer will be configured at game runtime with whatever actor is set as the current game actor.

Rate

Specifies the avatar's base rate of movement (in pixels per frame) that the avatar will tween around the room with. This rate of movement may be modified using a speed-based matrix filter.

Variables

Planned field. Currently unused.

Matrix-based filters

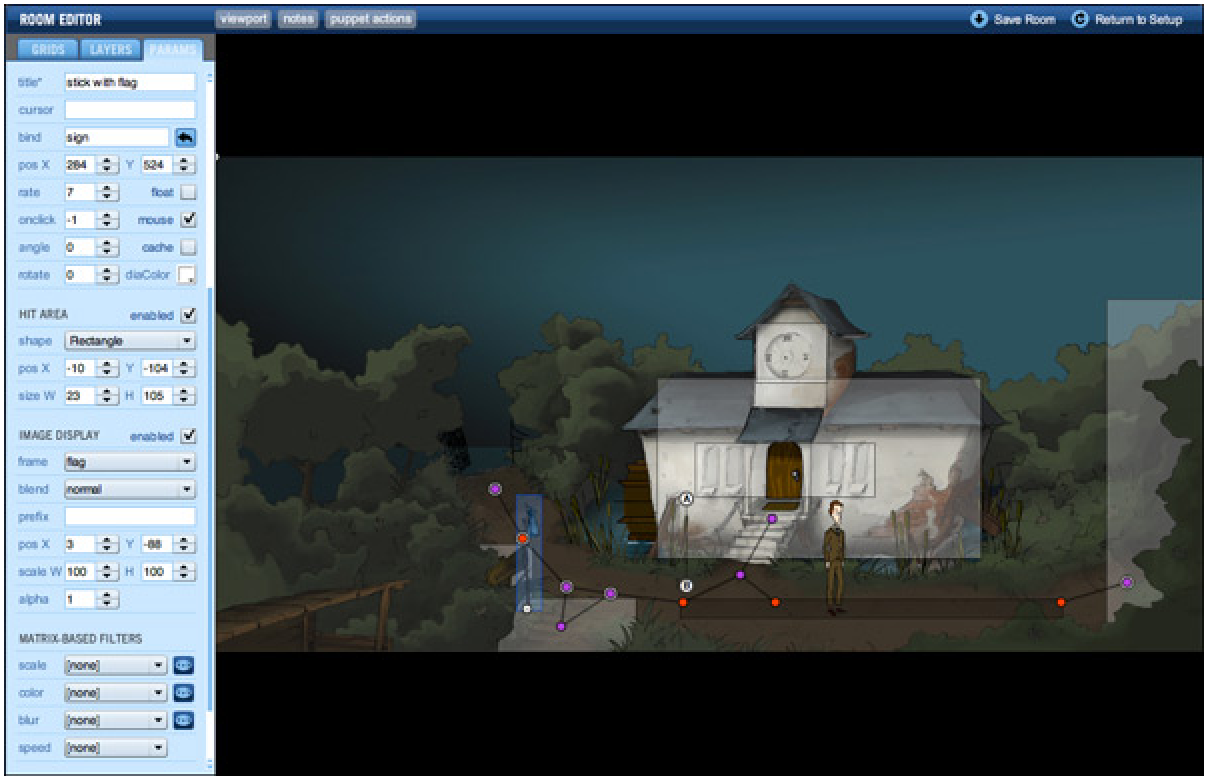
These options let you subscribe the layer to one matrix-based filter for each of the four available filter effects: scale, speed, color, and blur. Matrix filters are configured within the "GRIDS" tab of the room editor. After configuring the positions and values for a matrix filter, you may then subscribe layers to the filter. The filter's effects will be applied to subscribed layers based on their registered position with the filter's grid.

Plane Layer Parameters

Coming soon.

Puppet Parameters

This panel is accessed when you select a puppet layer within your room layout, then open the "PARAM" editor tab.



title* stick with flag
cursor
bind sign
pos X 284 Y 524
rate 7 float ☐
onclick -1 mouse ✓
angle 0 cache ☐
rotate 0 diaColor

HIT AREA enabled ✓
shape Rectangle
pos X -10 Y -104
size W 23 H 105

IMAGE DISPLAY enabled ✓
frame flag
blend normal
prefix
pos X 3 Y -88
scale W 100 H 100
alpha 1

MATRIX-BASED FILTERS
scale [none]
color [none]
blur [none]
speed [none]

States

Like most layer types, a puppet layer may define multiple "states". A state is a set of visual and interactive properties that the puppet may assume. By changing the layer's current state within the game runtime, the layer's appearance and behavior may be changed.

A good example of a layer with multiple states would be a door. The door's layer would define two states: "opened" and "closed". The two states would be given different graphics to visually depict the position of the door. Also, the behavior of the door would change between it's opened and closed state – as in: you may open the closed door, then pass through the opened door.

Almost all puppet layer settings are specific to a state. This allows almost any attribute of a layer to be customized on a state-by-state basis.

State - Selector

This selector lists all states defined for the currently selected layer. All layers must have at least one state. Additionally, this selector defines the order in which states are arranged on the layer. State order may be important if you plan to take advantage of the <puppet> XML script's "nextState" and "prevState" features.

State - Id

This is a unique identifier name by which the state is referenced. No two states on a single puppet may share an ID. Once you pick a state identifier, it's suggested that you lock the state Id field so that you don't inadvertently change the state ID later on. Remember, if you ever change an ID within Lassie, you will need to go through all of your XML scripts and update any references to the changed ID.

State - Index (#)

This field defines the state's numeric index within the layer's state list. When you change a state's index, you'll see its new position reflected within the state selector listing (A). State order may be important if you plan to take advantage of the <puppet> XML script's "nextState" and "prevState" features.

State - Default

Check this box to set the selected state as the layer's default state.

Variables

Planned field, currently unused.

Title

This field defines the title text to display when the player mouses-over or interacts with the layer. This is a player-facing text field, and therefore accepts language-specific XML tagging. Multiple language texts must each be placed within their own XML blocks (wrapping in CDATA is suggested). For example:

```
<en><![CDATA[door]]></en><es><![CDATA[puerta]]></es>
```

During development, it is suggested that you develop your game in a single language, and DO NOT include any XML within the title field. When XML is not provided, Shepherd will automatically export the title text as XML keyed to the application's default language. Once your game is complete, you may use Shepherd's localization tools to export all player-facing texts for translation, then import all the localized XML back into Shepherd.

Cursor

Specifies a custom cursor frame to display when the layer is moused-over. This value MUST correspond to a frame label within the global UI's cursor MovieClip (found within "ui.fla").

Bind

Specifies a labeled grid node to bind the layer's map point to. A layer's "map point" is a position within the room layout that the avatar layer will walk to when the layer is interacted with. By default, all layers have a draggable map-point control to define this map position. However, it often times makes more sense to utilize an existing node within the room's walkable grid as a map position (particularly handy when multiple layers want to share a common map point). In these cases, you may select the desired walkable grid node (make sure it has a name), then click the arrow button next to the binding field. This will bind the layer's map point to the selected grid node. To unbind a grid node, just clear the binding field's value.

Also note, grid nodes are bound by name reference. If you ever change a grid node's name, you will need to redefine all binding references to it.

Position

The layer's registered position within the room layout. This position is illustrated with a white anchor point in the editor window. Note that the layer's image and hit area can both be shifted within the local coordinates of the layer (allowing you to register the layer at a specific point in its artwork). Registration is an important consideration when you have floating (walk-behind) layers or NPC layers that will move around the walkable grid. Floating layers are stacked based on their registration positions, and puppet layers will anchor to the walkable grid at their registration point. Generally it's best to configure artwork so that a layer's registration point falls at the implied position of where the layer touches the ground (ex: a character's registration point should be positioned at their feet).

Rate

The rate (pixels per frame) at which the layer will move around the room layout when tweened using Lassie's puppet layer motion controls. This setting is generally only needed for NPC layers that you plan to animate as additional characters walking around the room.

Float

Toggle's the layer's float status, or "walk behind" status. When a layer is set to float, it will trigger dynamic layer stacking within the room layout so that the avatar may move both in front of and behind a single layer.

Onclick

Specifies the index of a puppet interaction to call immediately upon clicking the layer. Interactions are defined within the puppet interaction window, accessible from the top toolbar. Within the puppet interaction window, there is a numbered list of available interactions. Cite the numeric index of one of these interactions to have that interaction called immediately upon layer click. Specify negative one (-1) to disable a layer's onclick action.

Mouse enabled

Specifies if mouse actions are enabled on the layer. Disable this option if you have a puppet that will change states or perform actions, but is NOT interactive with the mouse.

Angle

Specifies the puppet's default turn-view setting (angle to camera). Turn views are specified as a number 1-8, where 1 faces away from the camera, and each sequential view turns 45 degrees clockwise (ex: 2 = back-quarter-right, 3 = right, 4 = front-quarter-right, 5 = front, etc). If you have an NPC sprite with fully-configured directional views, then you'll need to set it's turn view value to match its graphics. If you do not match the turn view setting to the layer graphics, then you may see a jump in turn direction when Lassie switches the NPC between animated states (ex: rest, speak, move, etc).

Cache

Specifies if the layer should be cached as a bitmap by Flash. Enable this option for any COMPLETELY static graphics, such as room foreground layers. Any timeline animations found within cached layers will be stopped. Caching layers will increase game performance, especially in scrolling rooms.

Rotation

Specifies the rotation of the layer around its registration point.

Dialogue color

Specifies a color for dialogue subtitles associated with the layer.

Hit Area

A layer's hit area is a geometric shape that defines the layer's mouse response region. When a layer has its hit area enabled, then the layer's image artwork bounding will be ignored and all mouse behaviors will apply to the hit area. If the layer's hit area is NOT enabled, then all mouse behaviors will respond to the layer artwork's positive (non-transparent) image area.

A layer hit area's shape may be drawn as a rectangle or an oval. To quickly scale a hit area within the layout editor, hold the SHIFT key and click the layer's hit area, then drag the hit area's bounds out to your desired dimensions.

A layer's hit area may be shifted around within the local coordinates of the layer. This allows you to adjust the hit area's position relative to the layer's registration point. To quickly set the position of a hit area within the layer's coordinate space, hold the CONTROL (Mac) or COMMAND (Windows) key then drag the hit area shape around within the layer.

Image Display

A layer's image will display a MovieClip library asset as part of the layer. When a layer's image is disabled, the the layer will just exist as an invisible hit area within the room. If the layer's hit area is disabled, then all mouse activity will respond to the image's positive (non-transparent) pixels. It is generally better to use layer hit areas for mouse interactivity (which will provide uniform geometric shapes) rather than to rely on image pixels (which are generally irregularly shaped and therefore not intuitive for a player).

Blend

A blend mode for the image. Try them out! Blend modes do hurt performance, so render color effects into your source artwork if possible.

Prefix

Frame prefixes offer a platform for advanced character animation. If you're new to Lassie and just trying to build a simple game, just leave this field blank and don't worry about it. You can build a very robust system of game animations without it. For advanced users, keep reading…

The frame prefix allows you to create multiple groupings of animations for a single layer, then switch which set is used. A common animation grouping is the Lassie character defaults of "rest", "speak", and "move". If you configure these three animations for a puppet layer, then Lassie can utilize those frames to turn your puppet layer into a walking, talking NPC. However, this only allows your puppet to have one talk animation, one rest animation, and one move animation. What if you want to have multiple versions of these animations – for example, one where the character is wearing red, and another where they are wearing blue? In this case, we'll need an animation grouping for each costume, and a way to select which consume to display. Here's where the image prefix comes in…

Within our example character asset, let's say we create some animations with the following frame labels:

- red_rest
- red_speak
- red_move
- red_dance
- blue_rest

- blue_speak
- blue_move
- blue_dance

Notice the structure of the frame groupings? There are two frame prefixes: "red_" and "blue_" (the underscore IS considered part of the prefix). Within each prefixed grouping is a "rest", "speak", "move", and "dance" frame. Now, we can specify a prefix for the layer's image display that will tell which grouping to use when playing animations. Let's set the prefix to "red_". If we were to then talk to our NPC within the game player, the player would use the "speak" frame from the "red_" grouping, and revert to a "rest" state, also from the "red_" grouping. If we were to set the prefix to "blue_", then we would see the blue costume animations used within the Lassie player.

So, this covers how Lassie's default animation states (rest, speak, and move) work with prefixes. But what about your own animations? Animations that you plan to trigger using the <layerSprite> XML command? Well, these work too. Notice above that we defined a "dance" animation frame for each of the prefixed groupings. If we were to call that from the XML API, we'd use the following:

```
<layerSprite target="myRoom:myLayer" animFrame="dance" waitForComplete="1"/>
```

Note that in the above command line, we've referenced the animation to play WITHOUT its prefix. This is very important. When a group prefix is defined for a layer, the Lassie player will automatically prepend the prefix onto any animation requests. In the above command, we're requesting the "dance" animation regardless of what group prefix is currently defined for the layer.

Finally, a layer's prefix may be changed during game playback in one of two ways… you may either define multiple states for the layer, each with their own prefix value, or you could dynamically change the current state's prefix setting using the following:

```
<layerState target="myRoom:myLayer:myState" framePrefix="blue_"/>
```

Frame

All image displays pull MovieClips from a Lassie asset library, and therefor allow you to select a labeled frame from your asset MovieClip's timeline. The "frame" selector will show all frame labels currently available on your asset MovieClip's root timeline. NOTE: there is a bug in Shepherd where frame selection is recorded by frame number rather than by frame label. As a result, your frame selection will change if you ever rearrange the frame labels on your timeline. Until this is noted as fixed, be sure to set up your whole timeline of frame labels up front, or else always add new asset frames to the END of its timeline where none of the existing labels will be affected.

Position

A layer's image display may be shifted around within the local coordinates of the layer. This allows you to adjust the image display's position relative to the layer's registration point. To quickly set the position of an image display within the layer's coordinate space, hold the

CONTROL (Mac) or COMMAND (Windows) key then drag the image asset around within the layer. You'll generally want to position your image asset so that it's implied contact point with the ground lines up with the layer registration. For example: you'd want to position an NPC layer's artwork so that the layer registration falls at the character's feet.

Scale

Percentages for horizontal and vertical image scale.

Alpha

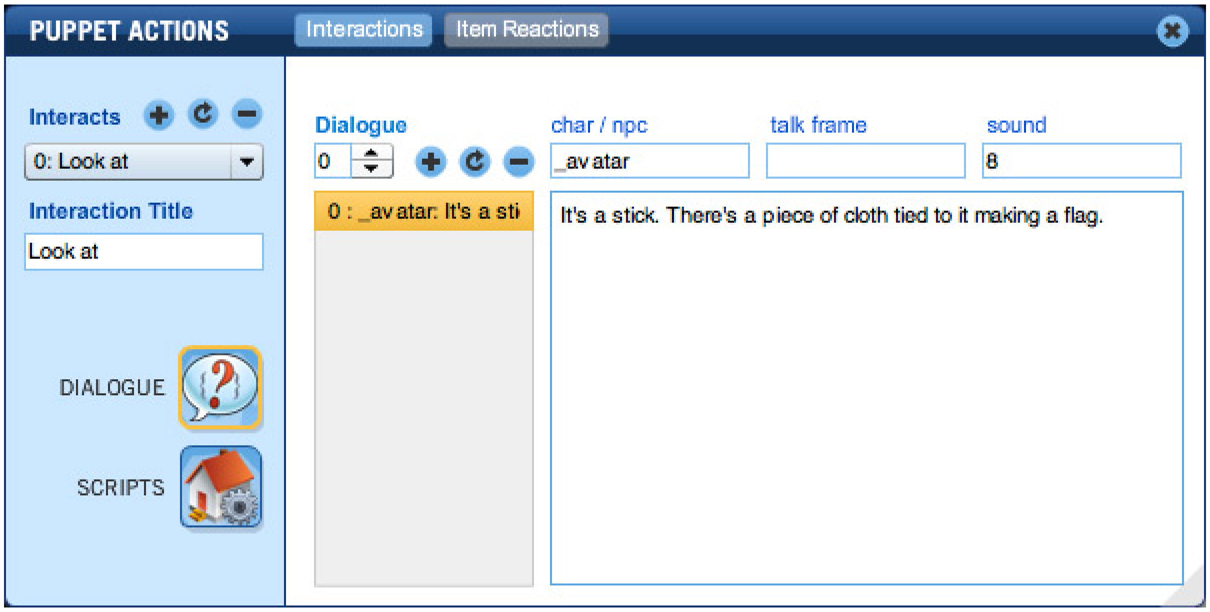
Opacity of the layer artwork.

Matrix-Based Filters

These options let you subscribe the layer to one matrix-based filter for each of the four available filter effects: scale, speed, color, and blur. Matrix filters are configured within the "GRIDS" tab of the room editor. After configuring the positions and values for a matrix filter, you may then subscribe layers to the filter. The filter's effects will be applied to subscribed layers based on their registered position with the filter's grid.

Puppet Actions

The puppet actions window is accessed by selecting a puppet layer within your layout, then clicking the "puppet actions" button in the editor application's top bar.



It's a stick. There's a piece of cloth tied to it making a flag.

Puppet layers are unique in the amount of interactivity that can be assigned to them. A puppet layer my be assigned a collection of dialogue and script triggers that are called in response to interacting with the object. To manage a puppet's interactivity, open the puppet interaction panel with the button in the upper-left corner of the interface (next to the viewport toggle). This window will show a set of interactive controls while a puppet layer is selected. This window manages both lists of puppet interactions: one for basic interactions, and the other for item interactions. To toggle between basic and item interaction controls, use the interaction-type selector buttons within the window's header bar.

When assigning actions, a list of dialogue and scripts may be assigned for each action in the list.

A handy feature introduced into Lassie Shepherd v1.1 is cross-state interaction linking. It's not uncommon to want to use a single puppet interaction for multiple states. For example, lets say that you have a multi-state puppet with a unique "use" action assigned to each state, however the puppet's "look" and "talk to" responses do not change. Rather than assigning duplicate instances of the same dialogue and scripts to each state for those two interactions, Lassie Shepherd v1.1 introduces the "@state:" compiler directive to handle this scenario.

The "@state:" compiler directive will allow a direct copy of a single interaction between states. To use the directive, just specify "@state:id" within the interaction's title field, where "id" refers to another state ID within the same layer. Then when the layer's data compiles, the "@state:" directive will instruct the compiler to reuse data compiled for the referenced state. There's only one catch: when referencing another state ID, you may ONLY reference a state with a lesser numeric index in the layer's state list than the state with the reference. In other words, a state may only reference a previous state in the layer's state list. This limitation is imposed due to the linear nature of the compiler: if you cannot reference a later state because it will not have been compiled at the time the reference is processed.

Grids Panel

The grids panel is accessed from the "GRIDS" tab in the editor application's top bar.

Room Properties

Music

Specifies a sound asset from an external media library to be used at the room's looping music track.

FX

Specifies a sound asset from an external media library to be used at the room's looping ambient effects track.

Walkable Areas

Grid State Selector

Like room layers, a room's walkable grid may be given multiple states. Grid states are useful for reconfigure a room's walkable area during the game runtime. By switching grid states, you can reconfigure the room's walkable paths. Grid states are fully supported within the game player application, however, their controls are a little buggy within the Shepherd editor. The grid layout sometimes does not refresh when you change states. This is a known issue, although a low-priority bug given that multiple grid states are not a commonly used game feature.

Grid ID

Specifies an ID name for each grid state. Grid state ID names MUST all be unique within a single room layout. It is suggested that you lock each grid state's ID to avoid inadvertently changing a name.

Active

Check this box to active the selected grid state as the room's default grid.

Grid boxes

Boxes are free-motions areas within a walkable grid, or, rectangular regions in which a puppet may move to any point. Boxes are Lassie's primary means for selecting a point that the avatar layer will move to when the background layer is randomly clicked.

Boxes may be freely added into and removed from a grid layout. Click and drag a box to set its position within the room layout. Also, you may hold the SHIFT key and click on a box to quickly resize its dimensions.

Boxes may overlap one another within your room layout, however they will NOT automatically connect to one another as a unified area. To connect box regions together, you must place one or more grid nodes within the boxes' intersection. Those nodes will provide junction points by which a puppet may pass from one boxed region to the next.

Grid Nodes

Nodes are the primary grid structure behind puppet motion within Lassie. While boxes define safe regions for background-click trapping, grid nodes define the actual structure that puppets will use to navigate between boxes and around the room.

Nodes may be freely added into and removed from a grid layout. Newly added nodes will start in the center of the editor window. Click and drag nodes to position them within the room layout. You may hold down the SHIFT key while making a selection to select multiple nodes at once. Multiple nodes may be deleted, joined, or broken.

When two nodes are joined, a line is drawn between them to illustrate their connection. This line, known as a "beam", defines the motion path that a puppet will follow between the two points. Breaking nodes apart will sever all connections between them. When two nodes fall within a single grid box, they will automatically be joined together, whether connected by beams or not.

Selected Node

Selected node features are available when there is only a single grid node selected. The selected node controls allow you to configure properties and behaviors of each individual node within the grid. Node properties are entirely optional and –in most cases– are not needed for the majority of nodes within your grid. However, you'll generally want to at least customize the starting grid positions within your room, and you may choose to implement the advanced process of grid-based filter effects (see below).

Node ID

Grid nodes may be given an ID name. Node ID names must be unique across all grid states (ie: a single node ID can only be used once per room, regardless of what grid state it appears in). Named nodes can be bound to room objects to serve as the object's walk-to map point. Also, a node ID may be specified when changing rooms to specify where in the room the avatar will start.

Turn

Specifies the a turn view (number 1-8) that the avatar layer should assume when starting within the room layout at the selected node.

Trap

Specifies if the node is eligible for grid trapping. Grid trapping is the process of selecting a point that the avatar will move to in response to a click on the background layer. When trapping a mouse click, boxes will be reviewed first for the nearest free-motion position to the click, then trapping nodes will be searched for a closer match. Ultimately, the avatar will move to the nearest boxed/trapped position found to the mouse click.

Grid-Based Filter Effects

Filters are visual transformations that can be applied to one or more puppet layers across a grid. Lassie Shepherd supports four puppet filters: scale, color, speed, and blur. The most commonly used of these filters is the scale filter, which allows a puppet to scale to different sizes at different parts within a room (creating implied 2D perspective).

Basic filter effects are done using matrix-based filters (see below). A basic matrix filter provides two points, called "poles", along a single axis. Filter settings are configured for each pole of the matrix, then one or more puppets are assigned to the filter. The puppets' visual properties will then be transformed based on their positions between the two poles.

Grid-based filter effects operate on the same principals of a matrix-based effect, except that every node within the walkable grid it treated as a filter pole. Therefore, a puppet may be transformed differently between each node in the walkable grid. Grid-based filters are more work to configure and manage than matrix-based filters, however they offer an unmatched level of control over puppet transformations.

When configuring grid-based filters, there is one important rule to keep in mind: a puppet's visual transformations will only scale when moving TO a grid node with enabled filter settings. All grid-based transitions will start with the puppet's current filter settings, and then end with the destination node's filter settings. If the destination node does not have any enabled filter settings, then the puppet will not transform along that beam of the grid. This allows you to configure far fewer nodes with filter properties (depending on how you structure your grid).

TIP: whenever possible, set up a grid-based system using filtered "zones" – a zone being a cluster of nodes that will all share the same filter settings. By connecting each zone together with a single beam that defines filter settings, you will create transition corridors between each zone. All non-transition nodes within each zone can then be left with their filter settings disabled.

Scale

The scale matrix filter value assigned to the node. If enabled, a puppet will transition to the specified scale on its approach to the node.

Speed

The speed matrix filter value assigned to the node. If enabled, a puppet will transition to the specified speed percentage on its approach to the node.

Blur

The blur matrix filter value assigned to the node. If enabled, a puppet will transition to the specified blur radius on its approach to the node.

Color

The color matrix filter value assigned to the node. If enabled, a puppet will transition to the specified color transform on its approach to the node.

Script

An XML script to be called at the time a puppet layer moves over the node while following a walkable grid path.

Matrix-Based Filters

Filters are visual transformations that can be applied to one or more puppet layers across a grid. Lassie Shepherd supports four puppet filters: scale, color, speed, and blur. The most commonly used of these filters is the scale filter, which allows a puppet to scale to different sizes at different parts within a room (creating implied 2D perspective).

Basic matrix filter effects provide two points, called "poles", along a single axis. Filter settings are configured for each pole of the matrix, then one or more puppets are assigned to the filter. The puppets' visual properties will then be transformed based on their positions between the two poles.

Lassie Grid Motion

The following overview summarizes how the components of the Lassie grid system are used to control puppet motion.

Grid Trapping

"Trapping" is the process of selecting a target point within a room layout to move a puppet layer to which falls within the valid walkable area. The classic example here is clicking on a room's ceiling, which is just drawn into the background layer. There is no custom walk-to point specified for a generic click within the room background, however the Lassie Player will still attempt to move the avatar layer as close to the clicked point as possible. However, given that we don't want the character to walk onto the ceiling, an alternate point that falls within the walkable area must be selected, or "trapped".

Legacy versions of the Lassie system trapped exclusively with boxes. A stray click on the background would move the avatar to the closest possible position to the target that feel within the bounds of a box area. This same process still exists, but the Lassie Player now also supports advanced trapping using grid nodes. The full process of how grid motion is covered below within two scenarios:

Scenario 1: An interactive puppet layer is clicked or interacted with.

In this scenario, a specific target object to move to is defined, so the avatar layer will automatically attempt to move to the target layer's map point (floor point). This is a blue point that appears within the Shepherd editor while the layer is selected. Only puppet layers have map points. When the move-to target is a map point, then the Lassie Player will forcibly assure that the avatar reaches that point through whatever means necessary. The following process is used:

1. The Lassie Player tests if the layer's map point falls within a box area. If so, then the target point is reached by pathing to that box, and then directly to the target point.

2. If the target point does not fall within a box area, then the Lassie Player will forcibly create a connection to the target point by joining it to the closest grid node.

Scenario 2: The room background is clicked.

In this scenario, a the mouse is clicked without providing a specific layer target to move the avatar to. As a result, the avatar will move a close to the clicked mouse point as possible without leaving the confines of the safe walkable grid area. This to location that the avatar moves to is selected as follows:

1. If the click falls within the bounds of a walkable box, the actual click location is used as the move-to target, and no further trapping is performed. In this case, the avatar will move to the target point by joining with any grid nodes that fall within the box.

2. If the click did not register within any box areas, then an operation is run to trap the nearest point on screen that falls within a box area. A second operation is then run to find the nearest trapping grid node to the click location. The results of the box trapping and node trapping operations are then compared, and the closer match is used as the avatar's move-to point.

Scripting Your Game

Here's a primer on how to script basic interactions within Lassie Shepherd.

An Overview of Scripting in Lassie Shepherd

Shepherd introduces a dramatically new system of programming to Lassie games. Where previous versions of Lassie were programmed with pseudo-ECMA style script commands, Shepherd is programmed entirely using XML snippets. For those unfamiliar with XML, fear not. Composing XML is exactly like writing HTML markup, except that XML tags use custom names and attributes. A basic Shepherd script command looks like this:

```
<game room="beach"/>
```

XML also supports nested structures, which is leveraged by some Shepherd commands for layered script controls. A basic nested structure used in a Shepherd script would look like this:

```
<keyboard addKeyScript="i">
    <inventory display="toggle"/>
</keyboard>
```

Shepherd's scripts are composed in the Global Script panel and Room Script panel, as well as in each interaction manager throughout the Shepherd editor. When composing scripts, just line up multiple XML snippits in sequence, such as:

```
<cursor visible="0"/>
    <layerClip target="_avatar" frame="reach" waitForComplete="1"/>
    <inventory add="keys"/>
<cursor visible="1"/>
```

Additionally, Lassie Shepherd has been expanded with support for conditional logic operations. Conditional logic allows game behavior to be intelligently controlled based on current conditions within the engine. Using Shepherd's "logic" command, you can test to see if specific game conditions currently apply and take action (or not) in response. A basic Shepherd logic construct looks like this:

```
<logic eval="[_currentRoom] EQ 'beach'">
    <!-- add beach-specific behaviors here -->
</logic>
```

For full documentation on Shepherd XML commands and logic controls, see the complete Lassie Shepherd XML Script Reference manual.

Commonly Used Game Scripts

Content being written...

Localization & Voice

Localization

Content being written...

Talkie/Voice Production

Lassie Shepherd has native talkie (voice-based dialogue) support. The Shepherd editor takes care of naming sound files and creating voice sound links. Using Shepherd's default voice support system, you should just need to record sound clips and name them based on Shepherd's automated naming system, drop them into a media library, then plug the media library into the game. In general, you'll probably find it easiest to allow Shepherd to take care of voice media management.

Basic voice management

1) Create an empty voice library.

All voice sounds should be stored in separate media libraries that contain ONLY voice media. This allows the heavy loading of voice media to be skipped when a game has voices disabled. Following Shepherd's default voice process, you'll first need to create an empty media library that will be used to store a room's voices.

So, create a new (empty) media library using the provided blank library template. Name the SWF file something meaningful describing the voices that it will contain. It's generally a good idea to create a separate voice library for each room to minimize the amount of data that needs to be loaded for a single library. For example, you may create a "beach_voice.swf" library file for use in a room called "beach". Don't worry about dropping sounds into this new voice library yet. This blank library is just establishing a container that Shepherd will point voice links into.

2) Link empty voice library into a Shepherd room.

Next, drop your newly-created empty voice library into your project's "voice" directory. All voice libraries MUST be linked from this "voice" directory rather than the "lib" directory used for all other game media. Now load a room within Shepherd. Notice that on the main room screen there is a media linkage panel titled as "voice libraries". Click the "plus" button on that voice media linkage control to select a voice library for the room. A room only needs a single voice library linked in, although you may include additional voice libs if want to do some advanced sound management (discussed later). If you do include multiple voice libraries, make sure the room's primary voice library is the first item in the list. Shepherd will use the first library in the list as the default container to point sound links at.

3) Load your room layout.

Load your room layout into the editor, then click around among dialogue lines. Within the dialogue editor, there should be a small field on the right of the dialogue box that contains a number. As you create additional dialogue lines, you'll see a unique number assigned to each new line of dialogue created within the room. This is Shepherd's auto-numbering at work. Generally, you shouldn't touch these numbers. Shepherd is making sure that every number assigned is unique, so entering your own numbers risks breaking the numbering scheme. If two lines of dialogue in the room will use a common sound file, you may manually change one of the numbers to match the other so that they'll link to a single sound file. Generally this will be the only reason to change a dialogue number, unless you plan to do advanced dialogue management.

4) Save and publish your room.

Now save your room and publish it, then look at your room's dialogue transcript within your project's "transcripts" directory. You should now see that each dialogue line has a unique voice sound assigned to it. The sound path will look like this:

voice/lib_name.swf:room_id000X

That voice reference is a fully-qualified path linking to a media resource that you will need to create. Note that the first part of the media path ("voice/lib_name.swf") refers to that blank media library that you linked into the room. Shepherd will not create voice media paths until there is a voice library linked into the room that it can point media references at (hence the reason that we first needed to link in a blank library). The second part of the voice path refers to sound file it will look for within that library. This sound name is constructed as the room's ID plus the sound number. As you record and cut your sound files, use this name while saving out each clip (ex: "room_id0001.mp3", "room_id0002.mp3", etc).

5) Record, cut, and save your sound files.

Here's the fun part… record all of your dialogue! The Shepherd transcripts are designed to help you with this task. There is a CSV version of the dialogue list at the bottom of the transcript that you can copy into Excel which may be a big help (hint: copy into Excel, then sort all lines by the character ID column to group all character-specific dialogue). As you record, save out each file as a WAV or MP3 using the sound's filename portion of the path (ex: "room_id0001.mp3", "room_id0002.mp3", etc).

6) Import sound files into media library.

Now open your empty voice library FLA file, then import all your recorded and named sound files into the document's library. Now you'll need to define media linkages for each sound file. This can be done quickly using a provided JSFL resource. Do the following:

Close all other FLA files currently open within Flash except your voice library document. Then, locate a run the provided "voice_linkage.jsfl" script (you should be able to doubleclick its icon to launch the script within Flash and run it). After running the JSFL linkage script, you should see that all library sounds have had linkage enabled with their respective filenames, and that their linkage definitions have been written into frame-1 actions. Running this JSFL script will configure linkage on all media that is currently found in the library, although this does not guarantee that all media expected by the room is present. To validate that your library has all expected sound files properly linked in, look at the very bottom of your dialogue transcript to find a complete list of prepared media linkages for the room. Replace all frame-1 actions with that pre-compiled list and try to compile your library again. If your library does not compile using the expected media linkage list, then you're probably missing some sounds or have something mis-named.

It's a good idea to always validate your final voice library using the pre-compiled list of expected sound links. However, if you know your library is incomplete and you just want to plug in preliminary sounds you have so far, just use the JSFL script to enable linkage on all available media. The Shepherd player will not throw an error if a linked sound file is not found; it will simply ignore the missing sound and display a subtitle for the dialogue line instead. This makes it very easy to set up sound linkages then fill in the media paths with sound files as you work through recording them.

7) Publish and update voice libraries.

Finally, drop your compiled voice libraries into the "voice" folder to replace the empty placeholder libraries. Remember that you'll need to clear your browser cache to have the Shepherd player load the newest linked voice library. Assuming voices are enabled within global game data, your game should play voice sounds in tandem with subtitles. Note that subtitle duration will match the length of it's associated voice when available, otherwise it will display for a duration based on the length of the subtitle text. You may disable subtitle display within the game menu. When subtitles are disabled, a subtitle will only display when its associated voice sound is missing or cannot be played.

Advanced voice management

In cases where multiple rooms share substantial amounts of common dialogue, it may be beneficial to share dialogue sound media between rooms. This can be done by importing a single voice library into multiple rooms, or by importing multiple voice libraries into a single room. In either case, you will need to manage your own sound references when breaking away from Shepherd's default numbering system.

In the event that you want a dialogue line to use a manually-defined sound reference rather than Shepherd's default numbering scheme, you may replace a line of dialogue's sound number (in the small field to the right of the dialogue box) with a fully-qualified sound path reference. A fully qualified sound reference is formatted as "voice/ lib_name.swf:sound_name". The media library SWF specified in the path must be included in the room's list of voice library imports. Assuming a valid sound path is provided, then the custom path will be used instead of Shepherd's default path assignment.

Manual sound reference management is a lot of work and adds a substantial margin of error. In most cases, it's probably simpler to include a sound clip multiple times with different names and absorb the file size bloat rather than manually managing references. However, also keep in mind that you only need manual path definitions when sharing sounds between multiple room layouts. If a single sound is needed multiple times in a single room layout, then you can just give those dialogue instances a common sound number to make them share a sound file within the room's voice library.

Credits

Production Credits

The following list includes all known contributors to the Lassie Shepherd engine. Note that not all third-party contributions are open source technology, therefor you (the end game developer) are responsible for reviewing all third party licensing agreements to make sure that your project is in compliance with their licensing restrictions.

Lassie Shepherd creation, design and development
by Greg MacWilliam
gmacwill77@gmail.com

Game menu text translations

- Matthias Kempke (German)
- Pietro Turri (Italian)
- [missing author name] (Russian)
- [missing author name] (Spanish)
- Marc Zulet (Catalan)

Lassie Shepherd logo illustration
by John Green

Intellectual Material

Adobe AS3 corelib
http://code.google.com/p/as3corelib/
by Adobe Systems

Bulk Loader
http://code.google.com/p/bulk-loader/
by Arthur Debert

Crypto
http://code.google.com/p/as3crypto/
by henrit

Tweener
http://code.google.com/p/tweener/
by Tweener OSS community

TweenLite*
http://blog.greensock.com/tweenliteas3/
by Jack Doyle, GreenSock

SoundObject
http://code.google.com/p/sound-skin/
by Greg MacWilliam

Stats
http://code.google.com/p/mrdoob/wiki/stats
by mrdoob

MathParser
http://www.undefined.ch/mparser/index.html
by Raphael Graf

** Has known licensing policy that is NOT open-source.*