

Crank Storyboard Suite

Crank Storyboard Suite

Copyright © 2016 All Rights Reserved. For more information email info@cranksoftware.com or visit <http://www.cranksoftware.com>

Table of Contents

I. Crank Storyboard Suite	1
1. Storyboard Suite Overview	8
Introduction	8
Compatibility with Previous Versions	8
Storyboard Architecture	9
Graphical Composition Elements	9
Events and Actions	12
Variables, Dynamic Content and the Data Manager	15
Maintaining State and Reacting to Changes	20
Execution Pipeline	21
Execution Environment	24
Animations	24
Scripting	26
External Communication (Storyboard IO)	26
Performance Considerations	27
2. Storyboard Designer	28
Introduction	28
Designer Environment	28
Storyboard Designer Workbench	28
Anatomy of a Storyboard Designer Project	29
Storyboard Simulator	31
Storyboard Designer Editor	31
Editing Content	31
Editor Toolbar	32
Direct Editing	33
Storyboard Designer Views	33
Actions View	33
Application Model View	35
Animation Timeline View	37
Images View	42
Layers View	44
Navigator View	45
Outline View	46
Problems View	47
Properties View	47
Templates View	49
Variables View	49
Notes View	52
Creating a Storyboard Designer Project	53
New Storyboard Application	53
Photoshop PSD File Import	55
How Photoshop Content Will Import to Storyboard	58
Storyboard Embedded Engine Import	60
Existing Project Import	61
Storyboard Designer Development	62
Simulating and Exporting an Application	62
Translation and Internationalization	69
OpenGL ES 2.0 Custom Shader, 3D Model and Compressed Texture Support	73
Working With Templates	76
Working with Multiple Application Design Files	77
Circles and Arcs	81

9-Patch	82
Groups	83
Scrolling Layers	84
Target Configuration	85
User Defined Actions	87
User Defined Render Extensions	87
Photoshop Re-Import Feature	88
Storyboard Designer Utilities	91
Design Notes	91
GoTo Dialog	93
Storyboard Search Dialog	94
Resize Storyboard Application	95
Resource Clean Up Wizard	98
Consolidate Images Wizard	98
Trim Images Wizard	99
Split Images Wizard	100
Merge Control Images	101
Collaboration and Team Development	101
Revision Control System Integration	102
Comparing and Merging Model Files	102
Comparing and Merging Projects	104
3. Storyboard Engine	105
Introduction	105
Target Configuration	105
Application Files	105
Setting up Storyboard Engine	106
Running Storyboard Engine	106
Command line Options	106
Plugins	111
Custom Shader Support	118
Font Environment Variable	119
System Specific Requirements	119
Linux FBDEV x86, armle	119
Microsoft WinCE, Compact7 win32, armle	120
Yocto Jethro Linux kernel (3.14) OpenGL, FBDEV, armle	120
4. Storyboard Media	121
Introduction	121
Media Actions	121
gra.media.new.audio	121
gra.media.new.video	121
gra.media.volume	122
gra.media.seek	122
gra.media.stop	122
gra.media.playpause	123
Media Events	123
gre.media.exit	123
gre.media.timeupdate	123
gre.media.statechange	123
gre.media.complete	124
gre.media.error	124
Media backends	124
gstreamer-backend	124
Gstreamer pipeline	125
External render extensions	125

5. Event Definitions	126
Standard Event Definitions	126
System Events	126
Pointer Events	126
Keyboard Events	131
Screen Manager Events	132
Focus Events	133
Table Events	133
Table Scroll Events	134
Layer Scroll Events	135
Mobile Events (Android and iOS)	136
Android Events	136
Windows Embedded Compact 2013 (WEC2013) Events	137
Plugin Specific Event Definitions	137
gre.gesture.up	138
gre.gesture.down	138
gre.gesture.left	138
gre.gesture.right	138
gre.screendump.complete	138
timer.[name] Timer Events	138
gre.animate.complete.[name] Animation Events	138
gre.rendermgr.error	139
6. Action Definitions	140
Built-in Action Definitions	140
gra.screen	140
gra.screen.fade	140
gra.screen.hold	140
gra.screen.release	140
gra.sendevent	140
gra.datachange	141
gra.screen.focus.set	141
gra.screen.focus.next	141
gra.screen.focus.prev	141
gra.screen.focus.direction	141
gra.table.scroll	142
gra.table.resize	142
gra.table.navigate	143
gra.log	143
gra.resource.dump_def	143
gra.playback	144
Plugin Action Definitions	144
gra.lua	144
gra.animate	145
gra.animate.stop	145
gra.audio	145
gra.greio	145
gra.perf_state	146
gra.redirect	146
gra.screen.path	146
gra.screen.scale	147
gra.screen.glswitch	147
gra.screen.glrotate	148
gra.screen.glflip	148
gra.screen.gldoors	149

gra.screen.gltip	149
gra.screen.glcube	150
gra.screendump	151
gra.timer	151
7. Render Extension Definitions	152
Common Render Extension Options	152
Render Extension Alignment	152
Fill	153
Fill Render Extension Options	153
Polygon	154
Polygon Render Extension Options	154
Rectangle	154
Rectangle Render Extension Options	154
Image	154
Image Render Extension Options	154
Image Alignment	155
Text	155
Text Render Extension Options	156
External	156
External Render Extension Options	156
3D Model	156
3D Model Render Extension Options	157
8. Scripting with Lua	159
Lua Function Parameters	159
Passing Extra Parameters to Functions	160
Storyboard Lua Integration	160
Lua Execution Environment	161
Asynchronous Lua Support	161
Lua Debugger	162
Introduction	162
Configuration	162
Debugging	164
Storyboard Lua API	165
gre.SCRIPT_ROOT	165
gre.set_data	166
gre.get_data	166
gre.set_value	167
gre.get_value	168
gre.send_event	168
gre.send_event_target	169
gre.send_event_data	170
gre.receive_event	171
gre.greio_disconnect	172
gre.touch	172
gre.key_up	173
gre.key_down	173
gre.key_repeat	173
gre.redraw	174
gre.quit	174
gre.move_layer	175
gre.move_control	175
gre.clone_control	176
gre.delete_control	176
gre.get_control_attrs	177

gre.set_control_attrs	178
gre.get_layer_attrs	179
gre.set_layer_attrs	179
gre.set_layer_attrs_global	180
gre.get_table_attrs	181
gre.set_table_attrs	182
gre.get_table_cell_attrs	182
gre.get_string_size	183
gre.poly_string	184
gre.resolve_data_key	185
gre.load_resource	185
gre.load_image	186
gre.dump_resource	187
gre.walk_pool	187
gre.timer_set_timeout	188
gre.timer_set_interval	188
gre.timer_clear_timeout	189
gre.timer_clear_interval	190
gre.thread_create	191
gre.vfs_open	192
gre.mstime	192
gre.env	193
gre.animation_create	194
gre.animation_add_step	195
gre.animation_destroy	195
gre.animation_trigger	196
Storyboard Lua DOM Module	197
gredom.get_application	197
gredom.get_object	197
DOMOBJECT:get_name	198
DOMOBJECT:get_type	198
DOMOBJECT:get_parents	198
DOMOBJECT:get_children	198
DOMOBJECT:get_variables	199
Lua DOM Samples	199
Storyboard Lua Android Integration	200
Storyboard Lua Android Integration	200
9. Storyboard IO	206
Connecting to a Storyboard Application	206
Sending Events to a Storyboard Application	207
Setting Application Data	208
Receiving Events from a Storyboard Application	209
Storyboard IO Utilities	210
iogen	210
iorcv	211
Storyboard IO API	211
gre_io_add_mdata	211
gre_io_close	212
gre_io_free_buffer	212
gre_io_grow_buffer	213
gre_io_open	213
gre_io_receive	214
gre_io_send	214
gre_io_send_mdata	215

gre_io_serialize	215
gre_io_size_buffer	216
gre_io_unserialize	216
gre_io_zero_buffer	217
10. Storyboard 3D Support	218
3D Rendering Fundamentals	218
The Scene Graph and Transformations	218
Material Support	219
Animation Support	220
Discussion on mapping FBX Animation data into meaningful structures	220
Support for Animation Takes	221
11. Optimizing Your Storyboard Application	222
Choosing the Right Image Format(s)	222
Frames Per Second	222
Scaling Images	222
Reducing Output Verbosity	222
Adjusting Engine Rendering Options	222
Memory	223
Measuring Performance	223
12. Storyboard Software Updates	225
Automatic Updates	225
II. Storyboard Design Tutorials	226
13. Creating a Storyboard Project from a Sample	228
Creating a New Application using the Storyboard Samples	228
Import	228
Import Sample	228
New Sample Project	229
14. Working with Multiple Application Design Files	230
Creating a Project	230
Resolving Conflicts	234
15. Creating a 3D Model Application	235
16. Creating a Multi-Touch Application	241
17. Adding Extra Libraries for Android	244
18. Adding Extra Libraries for iOS	246
19. Crank Public SVN	247
20. Exporting a Storyboard Project	255
21. Importing a Storyboard Project	257
III. Storyboard Target Tutorials	260
22. Linux	262
TI AM355 Starter Kit	262
Step 1: Importing A Storyboard Sample	262
Step 2: Exporting A Storyboard Application	264
Step 3: Selecting The Storyboard Embedded Engine	266
Step 4: Configuring The Target Platform	267
Step 5: Running The Storyboard Application	267
IV. Release Notes	268
23. Release Notes 4.0	270
Introduction	270
Storyboard Designer	270
Changes	270
Known Issues	270
Storyboard Engine	271
Changes	271
Known Issues	272

24. Release Notes 4.1	273
Introduction	273
Storyboard Designer	273
Changes	273
Known Issues	274
Storyboard Engine	274
Changes	274
Known Issues	274
25. Release Notes 4.2	276
Introduction	276
Storyboard Designer	276
Changes	276
Known Issues	277
Storyboard Embedded Engine	277
Changes	277
Known Issues	278
26. Release Notes 4.2.1	279
Introduction	279
Storyboard Designer	279
Changes	279
Known Issues	279
Storyboard Embedded Engine	280
Changes	280
Known Issues	280
V. Licensing	281
27. END-USER LICENSE AGREEMENT	283
28. Crank Software Third Party License Guide	293
Introduction	293
Storyboard Designer	293
Lightweight Java Game Library	293
Storyboard Engine	294
Lua	294
SOIL	294
Option Parsing	295
XML Parsing	295
Imagination OpenGL libraries	296
FreeType library	296
Scanline edge-flag algorithm for antialiasing	298
General IFF format	299
GNU LESSER GENERAL PUBLIC LICENSE	299
Storyboard Engine Platform Specific Dependencies	301
All Simple Direct Media Layer (SDL) renderers	302
All Simple Direct Media Layer (SDL), OpenGL ES 2.0, and Fujitsu Jade renderers.	302
Fonts	302
Bitstream Vera	302
Bitstream Deja Vu	304
Liberation	305
Roboto	307
Lato	307

List of Tables

3.1. Options	106
3.2. Action Manager Options	107
3.3. Data Manager Options	107
3.4. IO Manager Options	107
3.5. Model Manager Options	107
3.6. Render Manager Options: Windows, win32, OpenGL ES 2.0, x86	107
3.7. Render Manager Options: Linux, sdl, x86	107
3.8. Render Manager Options: Linux, fbdev, x86, armle	108
3.9. Render Manager Options: Linux, directfb, x86, armle	108
3.10. Render Manager Options: Linux, Windows CE, Windows Compact 7, Mac OSX, Neutrino 6.5, OpenGL ES 2.0, armle (Beagleboard)	108
3.11. Render Manager Options: QNX Neutrino 6.5, Linux, Fujitsu Jade, armle	110
3.12. Render Manager Options: WinCE 6.0, Windows Compact 7, win32, armle	110
3.13. Resource Manager Options	110
3.14. Screen Manager Options	110
3.15. 3D model rendering: libgre-plugin-model3d.so	111
3.16. Capture/Playback: libgre-plugin-capture-playback.so	111
3.17. Gesture: libgre-plugin-gesture.so	111
3.18. Linux Input Support: libgre-plugin-dev-input.so	112
3.19. Lua Scripting: libgre-plugin-lua.so	112
3.20. Linux Multi-touch Protocol: libgre-plugin-mtdev.so	112
3.21. Linux Touchscreen Support: libgre-plugin-tslib.so	113
3.22. Logger: libgre-plugin-logger.so	113
3.23. QNX input support: libgre-plugin-gfi-input.so	115
3.24. Storyboard IO: libgre-plugin-greio.so	116

Part I. Crank Storyboard Suite

Table of Contents

1. Storyboard Suite Overview	8
Introduction	8
Compatibility with Previous Versions	8
Storyboard Architecture	9
Graphical Composition Elements	9
Events and Actions	12
Variables, Dynamic Content and the Data Manager	15
Maintaining State and Reacting to Changes	20
Execution Pipeline	21
Execution Environment	24
Animations	24
Scripting	26
External Communication (Storyboard IO)	26
Performance Considerations	27
2. Storyboard Designer	28
Introduction	28
Designer Environment	28
Storyboard Designer Workbench	28
Anatomy of a Storyboard Designer Project	29
Storyboard Simulator	31
Storyboard Designer Editor	31
Editing Content	31
Editor Toolbar	32
Direct Editing	33
Storyboard Designer Views	33
Actions View	33
Application Model View	35
Animation Timeline View	37
Images View	42
Layers View	44
Navigator View	45
Outline View	46
Problems View	47
Properties View	47
Templates View	49
Variables View	49
Notes View	52
Creating a Storyboard Designer Project	53
New Storyboard Application	53
Photoshop PSD File Import	55
How Photoshop Content Will Import to Storyboard	58
Storyboard Embedded Engine Import	60
Existing Project Import	61
Storyboard Designer Development	62
Simulating and Exporting an Application	62
Translation and Internationalization	69
OpenGL ES 2.0 Custom Shader, 3D Model and Compressed Texture Support	73
Working With Templates	76
Working with Multiple Application Design Files	77
Circles and Arcs	81
9-Patch	82

Groups	83
Scrolling Layers	84
Target Configuration	85
User Defined Actions	87
User Defined Render Extensions	87
Photoshop Re-Import Feature	88
Storyboard Designer Utilities	91
Design Notes	91
GoTo Dialog	93
Storyboard Search Dialog	94
Resize Storyboard Application	95
Resource Clean Up Wizard	98
Consolidate Images Wizard	98
Trim Images Wizard	99
Split Images Wizard	100
Merge Control Images	101
Collaboration and Team Development	101
Revision Control System Integration	102
Comparing and Merging Model Files	102
Comparing and Merging Projects	104
3. Storyboard Engine	105
Introduction	105
Target Configuration	105
Application Files	105
Setting up Storyboard Engine	106
Running Storyboard Engine	106
Command line Options	106
Plugins	111
Custom Shader Support	118
Font Environment Variable	119
System Specific Requirements	119
Linux FBDEV x86, armle	119
Microsoft WinCE, Compact7 win32, armle	120
Yocto Jethro Linux kernel (3.14) OpenGL, FBDEV, armle	120
4. Storyboard Media	121
Introduction	121
Media Actions	121
gra.media.new.audio	121
gra.media.new.video	121
gra.media.volume	122
gra.media.seek	122
gra.media.stop	122
gra.media.playpause	123
Media Events	123
gre.media.exit	123
gre.media.timeupdate	123
gre.media.statechange	123
gre.media.complete	124
gre.media.error	124
Media backends	124
gstreamer-backend	124
Gstreamer pipeline	125
External render extensions	125
5. Event Definitions	126

Standard Event Definitions	126
System Events	126
Pointer Events	126
Keyboard Events	131
Screen Manager Events	132
Focus Events	133
Table Events	133
Table Scroll Events	134
Layer Scroll Events	135
Mobile Events (Android and iOS)	136
Android Events	136
Windows Embedded Compact 2013 (WEC2013) Events	137
Plugin Specific Event Definitions	137
gre.gesture.up	138
gre.gesture.down	138
gre.gesture.left	138
gre.gesture.right	138
gre.screendump.complete	138
timer.[name] Timer Events	138
gre.animate.complete.[name] Animation Events	138
gre.rendermgr.error	139
6. Action Definitions	140
Built-in Action Definitions	140
gra.screen	140
gra.screen.fade	140
gra.screen.hold	140
gra.screen.release	140
gra.sendevent	140
gra.datachange	141
gra.screen.focus.set	141
gra.screen.focus.next	141
gra.screen.focus.prev	141
gra.screen.focus.direction	141
gra.table.scroll	142
gra.table.resize	142
gra.table.navigate	143
gra.log	143
gra.resource.dump_def	143
gra.playback	144
Plugin Action Definitions	144
gra.lua	144
gra.animate	145
gra.animate.stop	145
gra.audio	145
gra.greio	145
gra.perf_state	146
gra.redirect	146
gra.screen.path	146
gra.screen.scale	147
gra.screen.glswitch	147
gra.screen.glrotate	148
gra.screen.glflip	148
gra.screen.gldoors	149
gra.screen.gltip	149

gra.screen.glcube	150
gra.screendump	151
gra.timer	151
7. Render Extension Definitions	152
Common Render Extension Options	152
Render Extension Alignment	152
Fill	153
Fill Render Extension Options	153
Polygon	154
Polygon Render Extension Options	154
Rectangle	154
Rectangle Render Extension Options	154
Image	154
Image Render Extension Options	154
Image Alignment	155
Text	155
Text Render Extension Options	156
External	156
External Render Extension Options	156
3D Model	156
3D Model Render Extension Options	157
8. Scripting with Lua	159
Lua Function Parameters	159
Passing Extra Parameters to Functions	160
Storyboard Lua Integration	160
Lua Execution Environment	161
Asynchronous Lua Support	161
Lua Debugger	162
Introduction	162
Configuration	162
Debugging	164
Storyboard Lua API	165
gre.SCRIPT_ROOT	165
gre.set_data	166
gre.get_data	166
gre.set_value	167
gre.get_value	168
gre.send_event	168
gre.send_event_target	169
gre.send_event_data	170
gre.receive_event	171
gre.greio_disconnect	172
gre.touch	172
gre.key_up	173
gre.key_down	173
gre.key_repeat	173
gre.redraw	174
gre.quit	174
gre.move_layer	175
gre.move_control	175
gre.clone_control	176
gre.delete_control	176
gre.get_control_attrs	177
gre.set_control_attrs	178

gre.get_layer_attrs	179
gre.set_layer_attrs	179
gre.set_layer_attrs_global	180
gre.get_table_attrs	181
gre.set_table_attrs	182
gre.get_table_cell_attrs	182
gre.get_string_size	183
gre.poly_string	184
gre.resolve_data_key	185
gre.load_resource	185
gre.load_image	186
gre.dump_resource	187
gre.walk_pool	187
gre.timer_set_timeout	188
gre.timer_set_interval	188
gre.timer_clear_timeout	189
gre.timer_clear_interval	190
gre.thread_create	191
gre.vfs_open	192
gre.mstime	192
gre.env	193
gre.animation_create	194
gre.animation_add_step	195
gre.animation_destroy	195
gre.animation_trigger	196
Storyboard Lua DOM Module	197
gredom.get_application	197
gredom.get_object	197
DOMOBJECT:get_name	198
DOMOBJECT:get_type	198
DOMOBJECT:get_parents	198
DOMOBJECT:get_children	198
DOMOBJECT:get_variables	199
Lua DOM Samples	199
Storyboard Lua Android Integration	200
Storyboard Lua Android Integration	200
9. Storyboard IO	206
Connecting to a Storyboard Application	206
Sending Events to a Storyboard Application	207
Setting Application Data	208
Receiving Events from a Storyboard Application	209
Storyboard IO Utilities	210
iogen	210
iorcv	211
Storyboard IO API	211
gre_io_add_mdata	211
gre_io_close	212
gre_io_free_buffer	212
gre_io_grow_buffer	213
gre_io_open	213
gre_io_receive	214
gre_io_send	214
gre_io_send_mdata	215
gre_io_serialize	215

gre_io_size_buffer	216
gre_io_unserialize	216
gre_io_zero_buffer	217
10. Storyboard 3D Support	218
3D Rendering Fundamentals	218
The Scene Graph and Transformations	218
Material Support	219
Animation Support	220
Discussion on mapping FBX Animation data into meaningful structures	220
Support for Animation Takes	221
11. Optimizing Your Storyboard Application	222
Choosing the Right Image Format(s)	222
Frames Per Second	222
Scaling Images	222
Reducing Output Verbosity	222
Adjusting Engine Rendering Options	222
Memory	223
Measuring Performance	223
12. Storyboard Software Updates	225
Automatic Updates	225

Chapter 1. Storyboard Suite Overview

Introduction

Crank Storyboard Suite is composed of two components, Storyboard Designer and Storyboard Engine, that work together to provide a complete graphical user interface development and deployment environment targeted specifically at the needs of embedded user interface developers.



Storyboard Designer provides a desktop-based (Windows/Mac/Linux) graphical development environment that provides a drag and drop approach to building the user interface using elements familiar to graphic designers, Photoshop PSD content, PNG and JPG image files, and standard TrueType fonts.

When a design is ready for testing, it can be simulated in the desktop environment or easily (really, really easily!) deployed to an embedded target device. Storyboard Designer does not produce code, but produces a data model that is independent of the operating system or processor that will execute the UI. This data model is interpreted by Storyboard Engine.

Storyboard Engine is an embedded target specific rendering engine that is highly portable and optimized for a specific combination of CPU, operating system, and rendering technology.

These two technologies, Storyboard Designer and Storyboard Engine, provide a highly efficient way of transforming user interface prototypes created by graphic designers, using tools such as Photoshop, into working applications ready for deployment.

Compatibility with Previous Versions

Workspaces that were created with older versions of Storyboard Designer can be used directly with newer versions of Storyboard Designer but may require additional import steps to be followed to update to interim versions. Storyboard 4 will import project content from Storyboard 1.x, 2.x and 3.x projects.

Once a model file (*.gde) has been saved with a newer version of Storyboard Designer, the model file will be automatically converted to that version of Storyboard Designer, making it incompatible with older

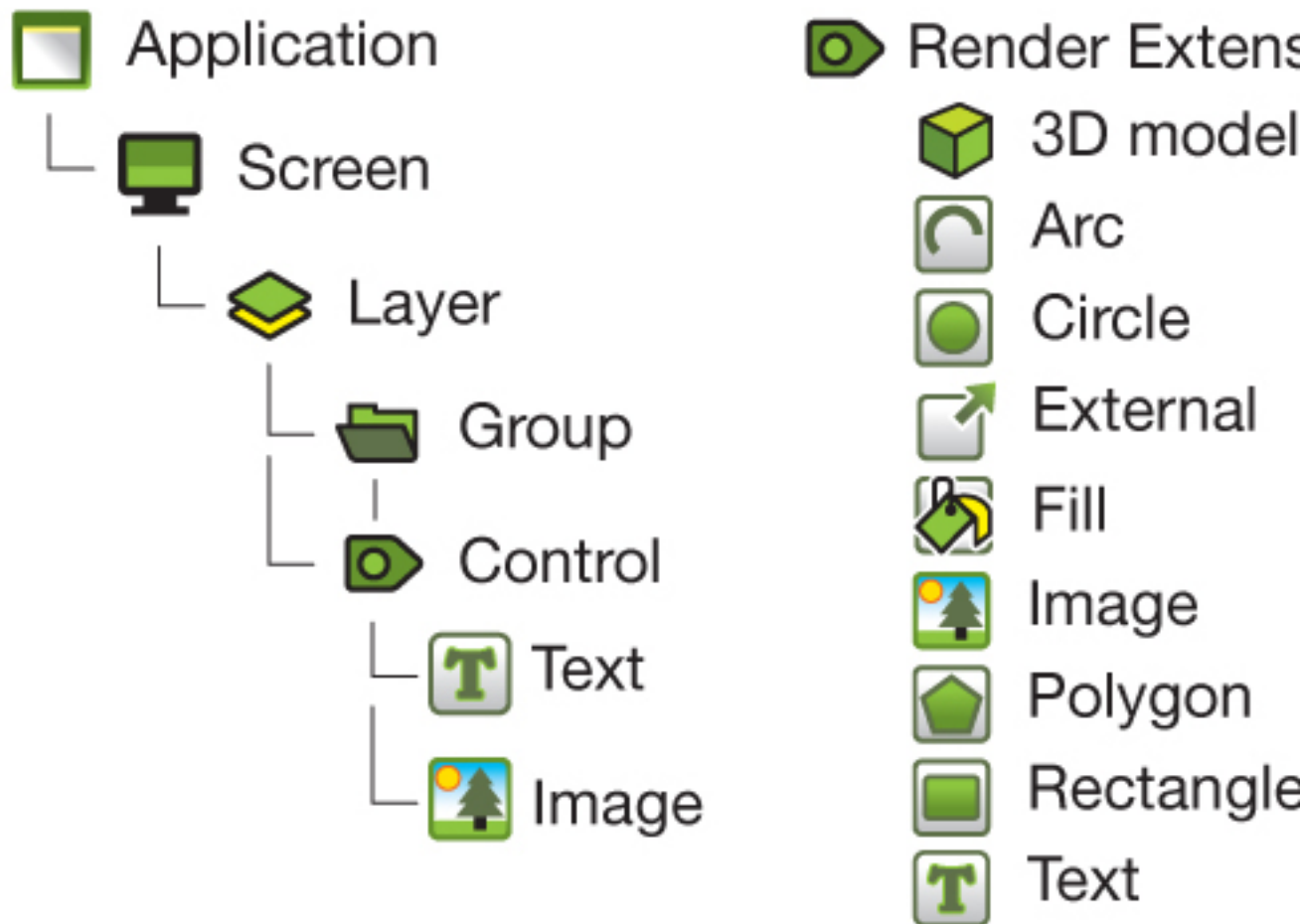
versions of Storyboard Designer. Users migrating to new versions of Storyboard Suite should take care to save and archive a snapshot of their project if they plan on updating it with an older Storyboard Designer environment.

If a new model file is required to be used in an older Storyboard Designer environment, the best approach to 'back migrate' is to export the application to a Storyboard Engine deployment from the newer version of Designer and then in the older version of Designer, use the import Storyboard Embedded Engine option to create a new Designer project. This export/import will cause certain items to be lost, such as notes and unused layers, and incompatible features will not be able to be transferred to the old model.

Storyboard Engine deployment files are also versioned with each release as the file structure changes, but Storyboard Designer maintains import support support for all versions of the engine.

Storyboard Architecture

Storyboard is designed to be used to develop and run fullscreen application user interfaces. Developing a Storyboard application is straightforward and simple, but relies on understanding a few concepts and terms and how they relate to one another within the Storyboard framework.



Graphical Composition Elements

A Storyboard application is composed of a hierarchy of *model elements*: Screens, layers, groups and controls.

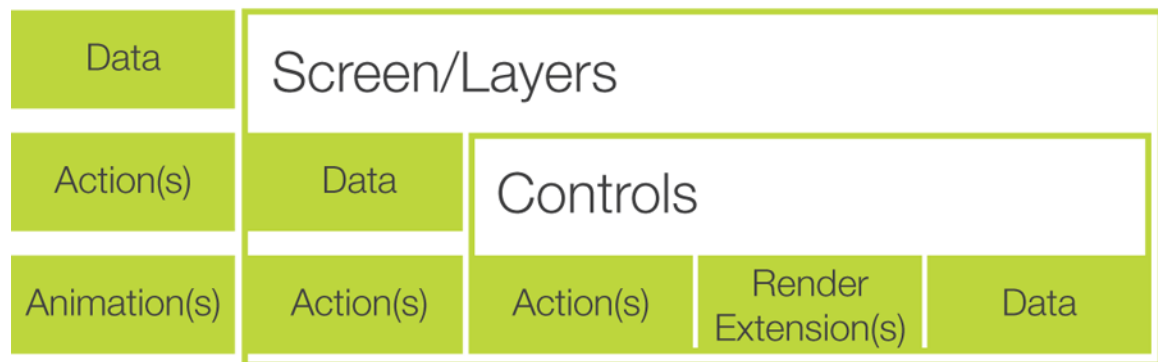
An application can be composed of multiple screens, however only a single screen is ever visible at one time.

Each screen is composed of one or more layers. The screen controls how multiple layers have their content composited together to form final display output. Where possible, layers may be mapped directly to layers in the graphic hardware. Multiple screens can reference the same layer, effectively sharing the visual content of the layer and creating multiple layer instances. All layer instances share the same visual content, but the position and visibility of the layer is specific to the screen context where it is being used. When a screen is painted, the z-order of the layer instances is used to determine which areas of a layer are visible to the user.

Layers contain controls and groups. Controls are the containers for renderable content such as images and text. Groups provide organizational structure that allow controls to be associated together in a common named container. A control is sized and positioned relative to its parent, layer or group, and contains zero or more render extensions that describe the type of rendering to occur when the control is damaged and redrawn. Controls are the only model elements in a Storyboard application that track input focus.

Each of the screen, layer, groups and control model elements are named items. Screen and layer names must not collide with one another and must be uniquely named. Control and group names must be unique within the scope of their parent container.

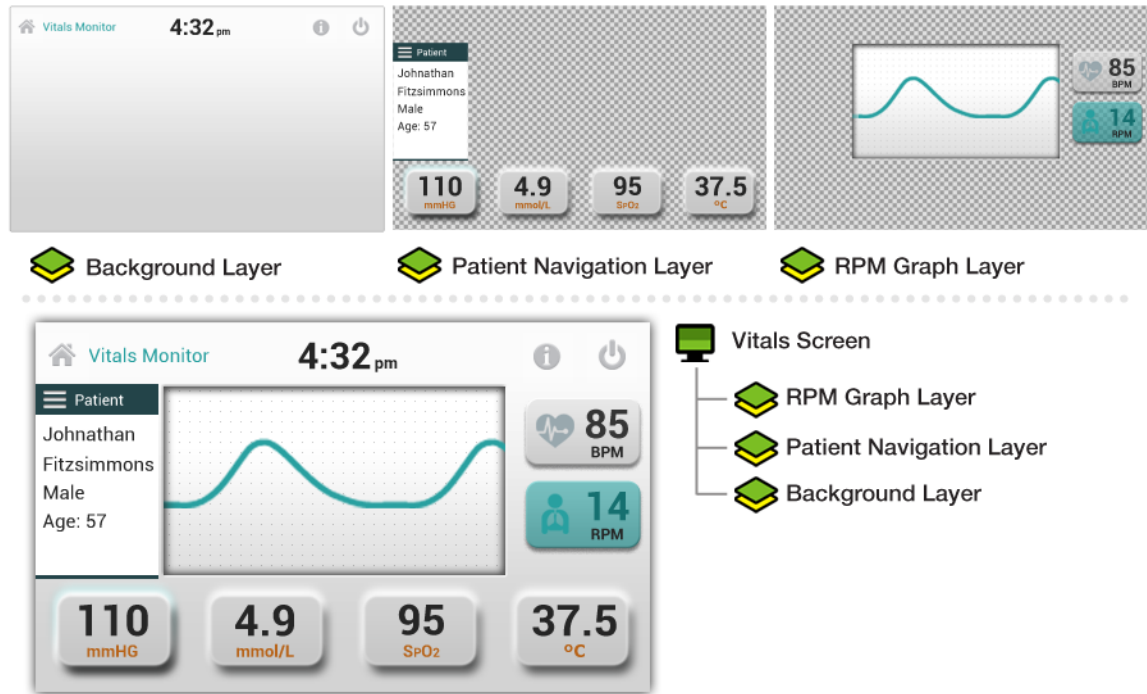
Application



Screens

A screen represents the user's view at any given moment. A screen is composed of one or more layers. These layers are composited to the screen in a z-order (back to front) in order to combine them to form the desired output. Each screen has a name and by using screen transition actions a user can traverse the user interface's many screens.

Here is an example of a screen containing three layers containing controls with images and text that combine to form the final screen:



Layers

Layers are the containers for controls and groups. A screen in the user interface is generally composed of multiple layers. A single layer can be reused and shared across multiple screens to allow common content to be displayed and manipulated. If the control or group content of a layer is changed, that change will propagate to all screens where that layer is being displayed. Certain layer attributes, such as a layer's position or its visibility are maintained on a per-screen instance basis. This allows designers to hide, show and move content on a screen by screen basis without affecting other screens where the layer may be used.

Layers can receive events and respond with actions as well as contain variables that can be referenced by other model elements. These events and variables are common to all places the layer is used.

Groups

Groups are containers for controls. Groups provide a mechanism to organize content and to place common UI elements together along with variables that may be shared among the controls. Groups have a position within a layer and a size that is dictated by the groups children. All controls within the group are laid out relative to the group's origin.

Groups can receive events and respond with actions as well as contain variables that can be referenced by other model elements, in particular the child controls of the group.

Controls

A user interface is made up of many controls. A control is a rectangular area of the screen that can render content and react to events. Each control can be made up of many render extensions and react to any number of events. Controls can be shown or hidden and active/inactive as the system requires. A special type of control, called a table, is available to create row/column-based layout of information. Each column in a table can have a distinct template for rendering. This allows for efficient and easy creation of list-style information.

Render Extensions

A render extension is the most basic piece of the user interface. It is responsible for rendering a specific type of content. Render extensions must be bound to a specific control and are not permitted to draw beyond the boundaries of the control to which they are bound. It is possible to include multiple render extensions on a control, and the order in which they are declared in the deployment bundle will define the order in which they perform their rendering.

Some render extensions included with the standard Storyboard release include:

- Render an image (PNG, GIF, JPEG)
- Render a text string
- Render a filled polygon, arc, circle or rectangle
- Render a 3D model (OBJ, GLES)

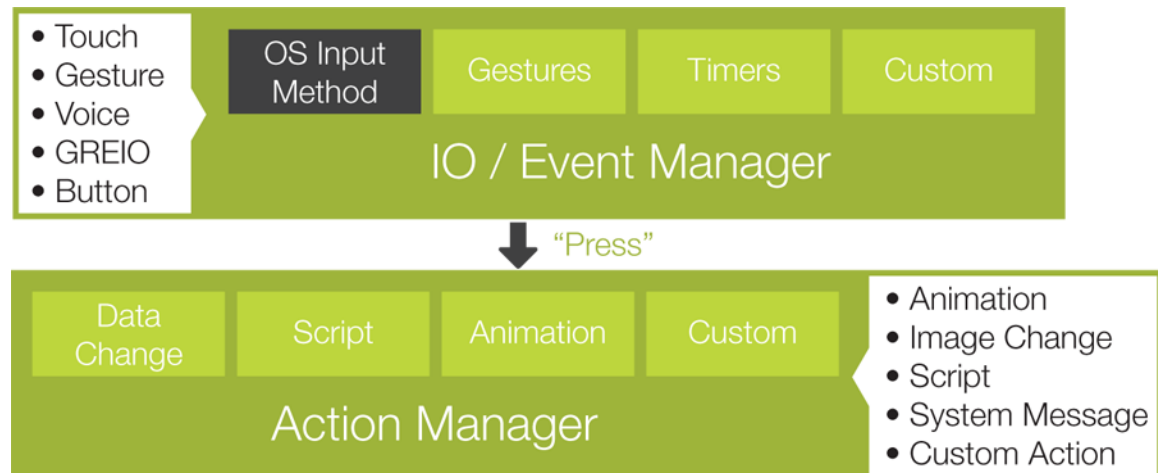
When designing a user interface, it is often possible to achieve the same look and feel by using multiple render extensions on a control or by using multiple controls with overlapping behaviour. The general rule of thumb is that multiple render extensions on a single control should be used when there is a common event for action binding that would occur for all of the visual components being displayed.

Events and Actions

Events

Events are the basic communication mechanism for triggering activity and passing data in a Storyboard application.

Events contain a name and a binary data payload. The content of the data payload is described using a format string that allows clients to generically decode the data payload. For example, the Lua action performs this decoding to allow access to the event data using string keys.



Events may be received from multiple input sources, but are processed serially by the Storyboard Engine. Clients can quickly and easily define their own custom events to enhance and drive their application, and to use these in conjunction with the standard event definitions provided by Storyboard.

A complete list of standard events can be found in the events definition section of this document.

Event Naming Conventions

New events can be readily defined and are not required to contain a data payload. In this case their format string and data payload will be empty values. When creating new events, it is appropriate to namespace the event definitions so that the names of events do not collide. For example, the Storyboard framework reserves the name prefix of `gre.` for user interface events, and the timer functions all generate events that are prefixed with `timer.`

The use of events is closely coupled with the declaration and operation of actions. An action can only be invoked when an event matching the action definition is received. This results in a common design pattern where an action will perform sophisticated logic in an external script or program and then signal a completion action to run once the script work is complete.

Event > Action (script) > Work > Trigger Event > Action (completion)

Event Format String

The format string on the event is used to assist with decoding event data. This decoding is used in the action bindings, to allow a minimal amount of additional logic to be placed into the event matching code, and also by certain advanced actions, such as Lua scripts, that can symbolically access the event data in the context of the script.

The format string provides a description of how to interpret the memory block (typically a data structure) that is associated with the event as its data payload. The format string is relatively straightforward to create and uses blocks that are formatted as `[numbytes][signed/unsigned][numelements] [][name]`. For the standard C data types the number formatting would look like:

```
1s0      --> Special null terminated string
1s1|1u1  --> 1 byte integer (int8_t uint8_t)
2s1|2u1  --> 2 byte integer (int16_t uint16_t)
4s1|4u1  --> 4 byte integer (int32_t uint32_t)
4f1      --> 4 byte floating point (IEEE754 float)
8s1|8u1  --> 8 byte integer (int64_t uint64_t)
```

So, if you were transmitting a structure such as:

```
struct {
    int32_t    a;
    uint16_t   b;
};
```

Which is assumed to have a bitwise memory layout of:

```
[a|a|a|a|b|b]
```

You would use a format string of `4s1 a 2u1 b` to describe the event.

The event data field descriptions `a` and `b` are optional. They are used to give the data symbolic representation for clients and do not have to be named to match the C structure field values. The descriptions are used by clients of the event data to symbolically access the event data. The Lua script plugin, for example, will use the symbolic name information as keys to a table that stores the event data content, appropriately decoded based on the format string definition.

The format string provided describes the memory layout of the data associated with the event. Consequently it is important that the format string compensate for any alignment or compiler padding provided by inserting appropriate additional format entries to skip over unused bytes. This may require slight adjustments to the format string based on the CPU architecture or compiler settings regarding C structure alignment.

Actions

Events trigger actions. Actions perform tasks; manipulate data, interact with the system, log messages, generate more events, etc.

Actions can be associated with any model object, but they are frequently associated with controls. When an event is received it is matched first against the actions associated with the currently focused control. After the control's actions, the processing passes to the actions associated with the visible layers on the screen, followed by the screen and application actions. This cascade of action handling provides an opportunity for the action execution to take place in the model context (application, screen, layer, control) that is most appropriate.

Actions are always invoked in response to an event to perform some sort of activity, and they are always executed in the context of the main execution thread. The trigger event might be a Storyboard standard event or a user event, and it might come from within the application, from a Lua script action, or be generated from an external program and injected using the Storyboard IO library.

More than one action might be invoked when an event is received. In this case the order in which the actions are declared within the design is the sequence that they will be invoked when an input event is matched. This action ordering can be used to ensure an order of operations execution. For example, changing the value of a data variable has to happen before executing a script that references that data variable.

The following demonstrates how the `gre.press` event triggers a `gra.datachange` action:



For more information on the Storyboard Engine execution pipeline, refer to the Execution Pipeline section of this document.

Actions are extensible components in the Storyboard framework and are usually implemented as plugins to be included in the runtime environment. Plugins have the ability to hook into the Storyboard Engine

managers and can provide a bridge between a Storyboard operation and the embedded system outside of the application.

A complete list of standard actions can be found in the Action Definitions section of this document.

Action Naming Conventions

Actions follow the same naming conventions as data variables, which are described in the Data Variables section of this document, with the following addition:

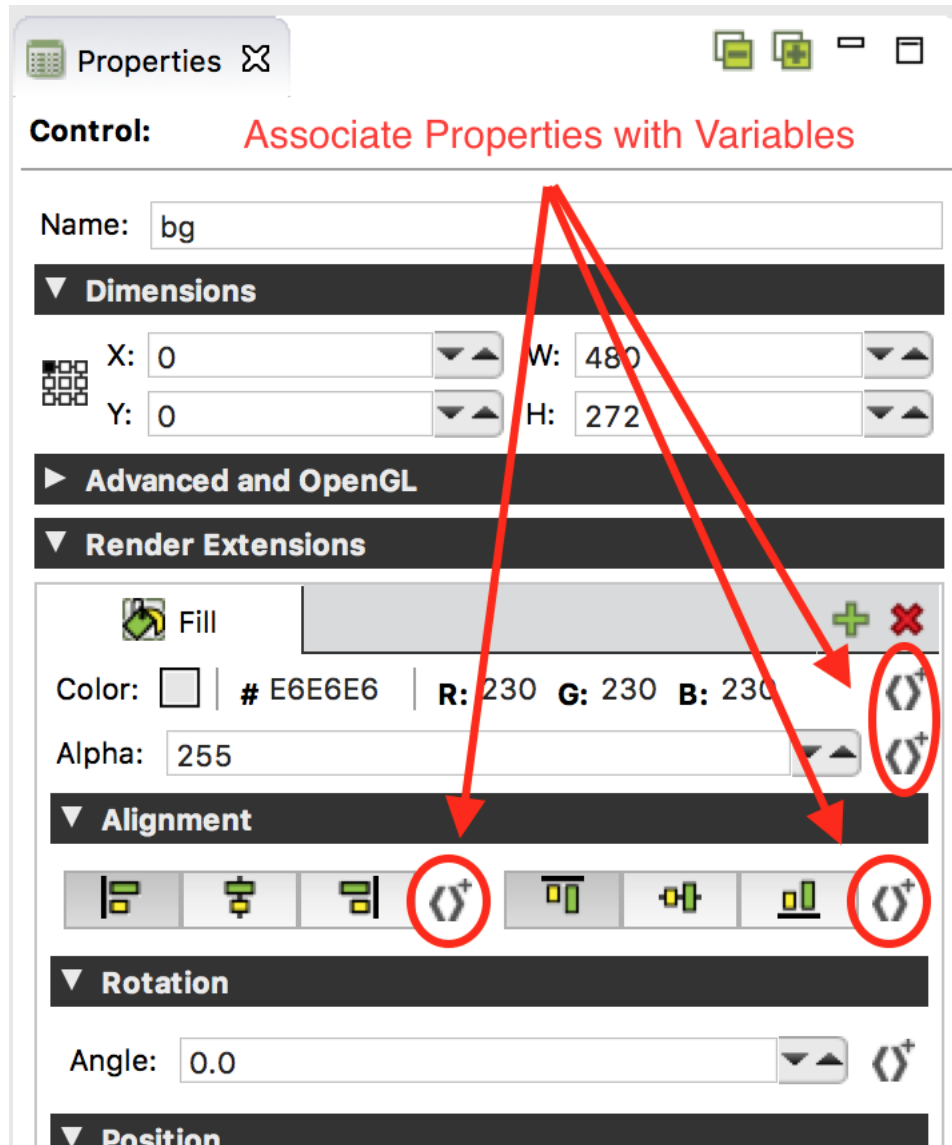
The namespace `gra.` is reserved for Storyboard internal actions.

Variables, Dynamic Content and the Data Manager

Initially all render extension properties and action arguments contain a set of fixed, static, values that are configured by the user as the Storyboard application is being created. Most applications however require a number of dynamic visual elements that are populated or changed during the runtime execution. In a Storyboard application this dynamic behaviour is provided by associating render extension properties and action arguments to data variables.

Data variables are associated with a particular model object that represents an appropriate scope for the use of variable. For example a control that behaves as a button may use a variable to contain the image to display in the button state. That variable would be defined on the control and associated to the image render extension 'Image' property. In a different scenario, an application may wish to have a uniform set of fonts used. In this situation an application level variable would be defined to contain the name of the font for a particular style (i.e. header) and then text render extensions would associate their 'Font' property to that application variable.

Variables are generally created at the point of use, for example within a render extension's property panel, by selecting the variable binding button located next to the property or argument value. Once a property or argument has been associated with a variable, the visual display will update to indicate the referenced variable. Similarly if a property is not required to be dynamic, then it can be decoupled from the variable and returned to a fixed value setting.



All of the variables defined in a Storyboard application are maintained by the Storyboard data manager database. As the value of a variable changes, the data manager will notify model elements that they should update their visual displays accordingly.

The data manager stores entries as key/value pairs where the key is a fully qualified model path (string) and the value is a set of bytes with a corresponding format string describing how the data payload should be interpreted.

The Storyboard model is hierarchical, so the construction of a fully qualified model path is straightforward process of joining model element name segments with a . (dot) in between them. The following list demonstrates how the fully qualified model name is formed for a variable, varname, associated with different contexts in the model.

varname	This identifies a variable, varname, as being an application level variable
screen_name.varname	This identifies a variable, varname, as being associated with the screen screen_name

<code>layer_name.varname</code>	This identifies a variable, <code>varname</code> , as being associated with the layer <code>layer_name</code>
<code>screen_name.layer_name.varname</code>	This identifies a variable, <code>varname</code> , as being associated with the layer instance <code>layer_name</code> associated with the screen <code>screen_name</code>
Most variables are not defined as layer instance variables, but rather as layer variables.	
<code>layer_name.control_name.varname</code>	This identifies a variable, <code>varname</code> , as being associated with the control <code>control_name</code> that is located on the layer <code>layer_name</code> .
Groups variables can also be addressed in this same manner as groups are also children of layers. The variable, <code>layer_name.group_name.varname</code> identifies a variable <code>varname</code> within the group <code>group_name</code> .	
<code>layer_name.group_name.control_name.varname</code>	This identifies a variable, <code>varname</code> , as being associated with the control <code>control_name</code> that is located in the group <code>group_name</code> on the layer <code>layer_name</code> .

Note

There is some overlap in the Storyboard namespace that could lead to ambiguous resolution. To maintain a clear name resolution, layers and screens may not have the same names and within a container such as a layer or a group, all of the model element names must be unique. This restriction is enforced by Storyboard Designer.

Using the fully qualified model paths can be cumbersome and impose extra maintenance effort as a project evolves or changes. Storyboard defines several variable shortcuts that will expand their value based on the current model context in which they are being resolved.

<code>\${app:varname}</code>	Refers to the application variable <code>varname</code> .
<code>\${screen:varname}</code>	Refers to the current screen's variable <code>varname</code>
<code>\${layer:varname}</code>	Refers to the current layer's variable <code>varname</code>
<code>\${group:varname}</code>	Refers to the current group's variable <code>varname</code>
<code>\${control:varname}</code>	Refers to the current control's variable <code>varname</code>

For example, assuming a Storyboard model of:

```
Application
+ MainScreen
+ FirstLayer
+ AGroup
+ AControl
+ SecondLayer
+ AnotherControl
```

where the current focus is associated with the control `AControl`, then a reference to a variable `varname` would resolve to a fully qualified path as follows:

<code>\${app:varname}</code>	<code>varname</code>
<code>\${screen:varname}</code>	<code>MainScreen.varname</code>
<code>\${layer:varname}</code>	<code>FirstLayer.varname</code>
<code>\${group:varname}</code>	<code>FirstLayer.AGroup.AControl.varname</code>
<code>\${control:varname}</code>	<code>FirstLayer.AGroup.AControl.varname</code>

Often these variables are used within render extensions and actions to define access to a dynamic value that will be adjusted at runtime. Alternatively, application variables can be used to provide an application-wide setting for a particular value. For example, if all text render extensions associated their font names with an application variable `${app:heading}`, then changing the value of the `heading` variable would change all the fonts in the application.

Storyboard Naming Conventions

Valid data variables and control/layer/screen names must follow the following rules:

- A valid name matches the following `[a-zA-Z][a-zA-Z0-9_]*`
 - Starts with a character `a-z/A-Z` not a digit or special character
 - Only `'_'` is supported as a special character, no spaces in names
- Screens and layers must be uniquely named from one another

Controls and groups must have unique names within their parent container

- Declared variables on a container model element (such as a layer or a group) cannot collide with the name of any of the child model elements of that container (such as a control or group)
- The prefix of `grd` is reserved for Storyboard internal variables
- The character `.` (dot) is reserved as a namespace separator

Layer, Group and Control Data Variables

Layers, groups and controls can be manipulated at runtime by modifying their internal data variables. These variables are created automatically for each layer, control and group in the model and do not have to be created by users. These variables use the reserved `grd_` variable namespace.

These variables are generally accessed using `${model_object:varname}`, for example `${control:grd_x}` to indicate the x position of the current control

Layer variables

The following values can be queried and changed through the normal data management channels. The position variables are relative to the screen.

<code>grd_x</code> (format = 4s1)	The layer instance's x position relative to the screen
<code>grd_y</code> (format = 4s1)	The layer instance's y position relative to the screen

<code>grd_xoffset</code> (format = 4s1)	The x pixel offset that will be used to determine the origin of the layer instance
---	--

<code>grd_yoffset</code> (format = 4s1)	The y pixel offset that will be used to determine the origin of the layer instance
---	--

<code>grd_width</code> (format = 4s1)	The layer's width
---------------------------------------	-------------------

Note

Any change to this value affects all layers.

<code>grd_height</code> (format = 4s1)	The layer's height
--	--------------------

Note

Any change to this value affects all layers.

<code>grd_alpha</code> (format = 1u1)	The layer's transparency value. The values range from 255 (opaque) to 0 (transparent)
---------------------------------------	---

<code>grd_hidden</code> (format = 1u1)	The layer's visibility. A value of 0 states that the layer and all of its controls are visible and a value of 1 hides the layer and all of its controls
--	---

Group variables

The following values can be queried and changed through the normal data management channels.

<code>grd_x</code> (format = 4s1)	The group's x position relative to its layer
-----------------------------------	--

<code>grd_y</code> (format = 4s1)	The group's y position relative to its layer
-----------------------------------	--

<code>grd_hidden</code> (format = 1u1)	The group's visibility. A value of 0 indicates that the control is visible and 1 that it is hidden
--	--

Control variables

The following values can be queried and changed through the normal data management channels.

<code>grd_x</code> (format = 4s1)	The control's x position relative to its layer
-----------------------------------	--

<code>grd_y</code> (format = 4s1)	The control's y position relative to its layer
-----------------------------------	--

<code>grd_width</code> (format = 4s1)	The control's width
---------------------------------------	---------------------

<code>grd_height</code> (format = 4s1)	The control's height
--	----------------------

<code>grd_zindex</code> (format = 4s1)	The control's z-index position. This sets the stacking order of controls within its layer where 0 is at the back (furthest from the eye).
--	---

<code>grd_hidden</code> (format = 1u1)	The control's visibility. A value of 0 indicates that the control is visible and 1 that it is hidden
--	--

<code>grd_active</code> (format = 1u1)	A value of 1 states that the control is active (can receive and react to events) and 0 for an inactive control (cannot receive or react to events)
<code>grd_opaque</code> (format = 1u1)	Indicates if the control is opaque to events. If opaque (1), the control will block events from being handled by other controls. If the value is 0, the events flow through the control to ones behind it.
<code>grd_findex</code> (format = 4s1)	<p>The control's focus index. This sets the focus on a control in a navigation sequence, where 1 sets the focus on the first control, 2 sets the second, etc. A value of 0 indicates that the control is not focusable.</p> <p>In order for a control's focus index to be changed dynamically at runtime, the focus value must be initially set to a non-zero value in Storyboard Designer.</p>

Table variables

A table contains all of the control variables and also a set of table specific variables. These table specific variables can be queried but not dynamically changed. In order to change these values in a table, actions are provided: `gra.table.resize`, `gra.table.scroll`. The variables are as follows.

<code>grd_rows</code> (format = 4s1)	The number of rows in the table
<code>grd_cols</code> (format = 4s1)	The number of columns in the table
<code>grd_visible_rows</code> (format = 4s1)	The number of visible rows in the table
<code>grd_visible_cols</code> (format = 4s1)	The number of visible columns in the table
<code>grd_active_row</code> (format = 4s1)	The row index of the currently active cell
<code>grd_active_col</code> (format = 4s1)	The column index of the currently active cell
<code>grd_row</code> (format = 4s1)	The table's current top left row
<code>grd_col</code> (format = 4s1)	The table's current top left column
<code>grd_xoffset</code> (format = 4s1)	The x pixel offset that will be used to determine the origin of the 1,1 table cell
<code>grd_yoffset</code> (format = 4s1)	The y pixel offset that will be used to determine the origin of the 1,1 table cell

Maintaining State and Reacting to Changes

In addition to providing a framework for defining the visual display of an application, the Engine framework contains a data manager that is responsible for maintaining the state information that controls the behaviour of the application, such as which screens to display and what content to render inside of a

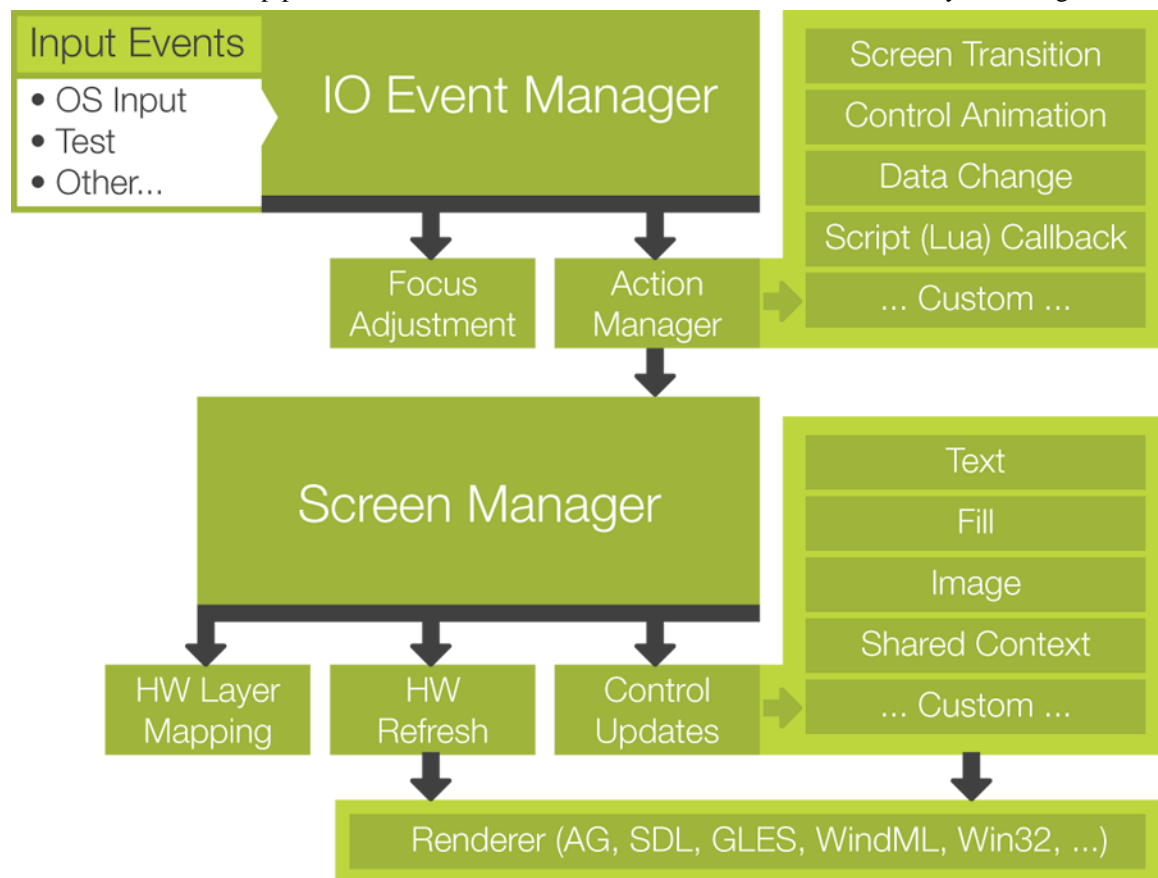
control. The Data Manager contains variables that are user-defined (string keys) and typed (string, integer etc), and can be readily modified.

Most often the Data Manager variables are modified by the application itself as it responds to events in the system. Events are asynchronous triggers, originating internally or externally to the Engine, that are in turn mapped to actions that perform work in the system.

Actions are where all of the real ‘execution’ is performed within an Engine. Actions are managed as extensions to the Engine and are flexible. Some of the built-in actions include changing content in the data manager, manipulating timers, logging messages or triggering additional events both inside and outside of the Storyboard framework.

Execution Pipeline

Below is the execution pipeline that is associated with the internal execution of the Storyboard Engine.



Execution begins with the arrival of an input event to the Storyboard Engine's IO Manager. That event is matched to all of the available actions that are *in context* by the Action Manager and executed in sequence. Changes in state will result in a notification to the Screen Manager which will manage the update and refresh of the visual display by invoking the appropriate rendering modules based on the current context.

Trigger Event

While the Engine defines a number of standard user interface types of events, there is no limit to the number of new events that can be created to control custom logic in an application. There can be multiple event input event sources concurrently generating events, however the events are queued. The delivery of the events, and potential execution of actions as a result, is serialized by the main application thread. This serialization is discussed further in the Execution Environment section of this document.

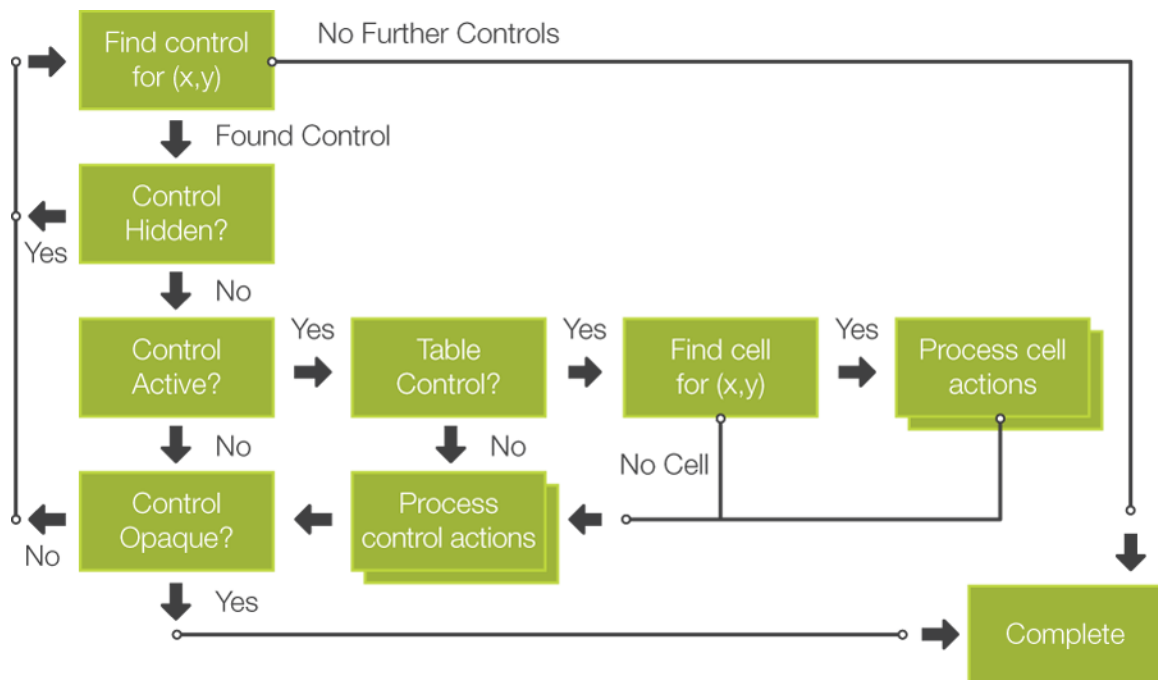
Action Execution

The application changes its internal state in response to events via actions bound to particular Storyboard model objects.

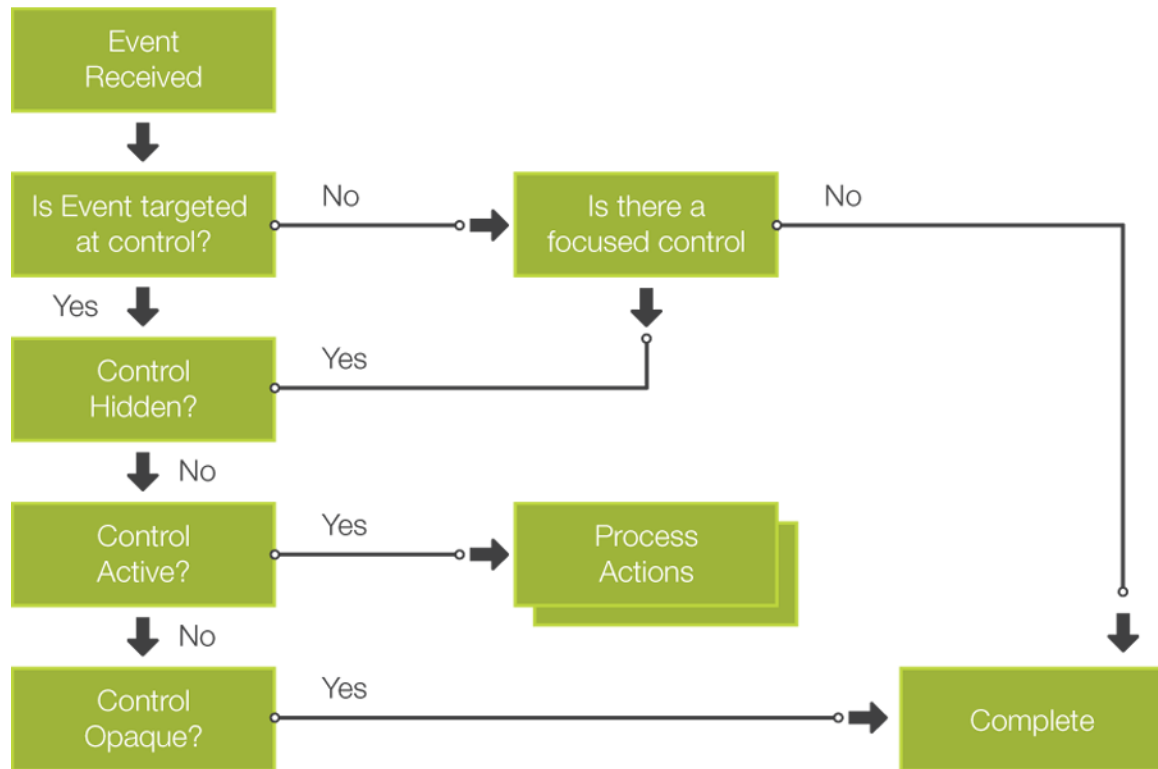
As an event is processed, it is matched against what events the action handlers available in a particular screen context are expecting. When there is a match between the received event and the expected event, the action handler is invoked. The action handler is invoked with arguments set in the deployment bundle, as well as the *context* of the invocation. The context includes the current screen, the current and focused controls as well as the event and its payload.

Multiple actions can be bound to a single event. These actions will be serially executed based on the order in which they were declared in the Design environment. The first declared action is the first action executed.

Actions can be declared on controls, layers, screens and applications. The event to action matching is performed first for controls based on screen display order (where applicable) and moves sequentially up the stack to process layer, screen, and the application actions. The execution of actions does not stop when one action handler is executed, with the exception that if a control is marked as opaque, then no additional controls will have their action handlers invoked. If an event is directed (an event which has a position) the only layer which this control is part of will have action handlers invoked, otherwise the actions for all layers will be invoked. Once control and layer actions have been invoked the following continues to the active screen and application context. The following diagram illustrates the affects of control flags on action handlers for a directed event (an event which has a position):



The following diagram illustrates the flow for an event which does not have positional information. These events may be targeted at a specific control or the focused control.



Focus

When an event is received the event is delivered to the currently focused control and this control's layer, and application. This is true for all events except for directed events such as mouse/touchscreen events which contain positional data.

The focus control for a screen is determined using the focus index value assigned to a control. A focus index is an integer value which places the control in the screen focus queue. A focus index must be unique across a screen. Focus can be navigated via the focus actions which allow for setting focus and moving through the focus queue (next/prev). The focus actions are described in more detail in the Action Definitions section of this document.

Data Change

The data change is a meta-stage in the execution pipeline. It is one of many potential outcomes of executing an action handler in response to an event. However, since most actions cause a change of state, and most state information and variables are maintained in the data manager, this topic deserves special discussion.

The data manager provides clients with the ability to be notified of data content changes through data change listener callback functions. Since it is possible for multiple action handlers to be invoked as a result of a single event, these callback functions should be restricted to a bare minimum of functionality, otherwise they may introduce undesired delay to the graphical rendering operations.

Data change listener notification/processing occurs after all of the actions have been processed for a particular event, allowing data changes resulting from multiple actions to be more efficiently processed.

Display Render

The display render action is also a meta-stage in the execution pipeline. As actions are executed, if any of these actions cause changes to data or state which is relevant to the rendering of content on the current

screen, then the display will be refreshed with the updated content. If there is no change in data content that would affect the current screen, then there is no display update required.

Execution Environment

A Storyboard application may be multi-threaded, however the execution pipeline is single threaded. This serialization is provided by the servicing of the event queue as each event is processed through the execution pipeline in sequence.

Internally, within the Engine framework, multiple threads are used to simplify control logic for things such as supporting multiple input sources, or timed event activities.

The threads that are run as part of the Storyboard framework are generally not signal handling, and mask off all signals. In certain situations there may be a requirement to handle specific signals, but in those situations the signal handling behaviour will be documented as part of the component API.

Animations

Animation Action

The Engine supports user defined animations using the animation action, `gra.animate`. This action starts executing an animation immediately, monitors the animation, and applies the specified changes as they have been defined by the user in Designer.

An animation is a named block of operations that will perform changes on Storyboard data values at a pre-determined frame rate. The individual data changes that occur within an animation are referred to as animation steps.

An animation step contains the following information:

key	The key is a reference to the data object that is going to be changed over the course of the animation. In general, keys are numeric items such as x or y position, width, height or transparency (alpha) values. However, it is possible with 0 duration animation steps to apply a change to any variable at a point in the animation. This includes text or images.
offset	This is the time in milliseconds from the time that the animation was started that this particular change will start to occur.
duration	This is the time in milliseconds over which the change will occur. This value may be 0 for changes that are not numeric (ie text or image values) or if the animation step is defined to occur at the start or end of the animation block.
rate	For non-zero duration animation steps, this is the change curve that will be applied to the numeric value from its start value to the end value. Example rates include linear, ease in (easein), ease out (easeout), ease in out (easeinout) or bounce.
starting value	This represents the starting value of the animation. The starting value can be either a specific value or variable reference, or it can be specified as the current value of the animation key at the time that the animation starts. Using the current value is good for animations that need to work generically to achieve some end value.
end value	This represents the end value of the animation. The end value can be either a specific value or variable reference or it can be specified as an offset from the starting value rather than as an absolute value. Using an offset (or delta) in an animation makes it easy to perform incremental animations on objects.

Animation steps are all synchronized within an animation block so that their data changes will occur in a synchronized manner. While it is possible to specify arbitrary time offsets and durations, these values will be mapped onto the nearest synchronized frame slot. The frame slots are dictated by the frame rate of the animation block.

Animation instances can be labelled with a string `id` which is an identifier used to provide exclusive execution. It is possible for many animations to run concurrently, however if two animations have the same `id` value, then only the last one invoked will actually run. For example, if you have an animation to shrink and grow a control, then you only want one of either the shrink or grow operations to occur at one time. This can be achieved by having the shrink and grow animation actions share the same identifier.

Animations may be stopped at any time by using the animation stop action, `gra.animate.stop`.

Timer Keyframe Animations

Animations can also be created in the more traditional method of setting a timer and operating on data on every timer firing. By manipulating data in the timer callback, clients can cause any number of custom behaviours to occur, as the data changes on variables will automatically be reflected through to the user interface as would be done at any other time.

The timer callback allows non-traditional data change rates to be applied, as well as flip-book style animations where a sequence of images is pre-defined and changed on each timer iteration. This could be achieved through a series of 0 duration animations as of Storyboard 3.0.

Screen Transition Animations

Animations are frequently used during screen transitions. A screen transition is a way to move from the visible screen to a new screen which may or may not have common layers. By default, screen transitions can be invoked by using one of the following actions:

<code>gra.screen</code>	Transition to a new screen immediately
<code>gra.screen.fade</code>	Fade the new screen into the current screen over time
<code>gra.screen.path</code>	Slide the new screen in and the old screen out over time, from one of the following directions <ul style="list-style-type: none">• Left• Right• Top• Bottom
<code>gra.screen.scale</code>	Grow the new screen over the current screen

All transition actions, which are time-based, take similar arguments that control the duration of the transition, the rate at which the transitions will occur, the orientation of the transition, and the number of frames that should be used. Using these arguments, the designer can control the user experience (i.e. duration and effects) as well as the overhead incurred on the system (frequency of frame updates).

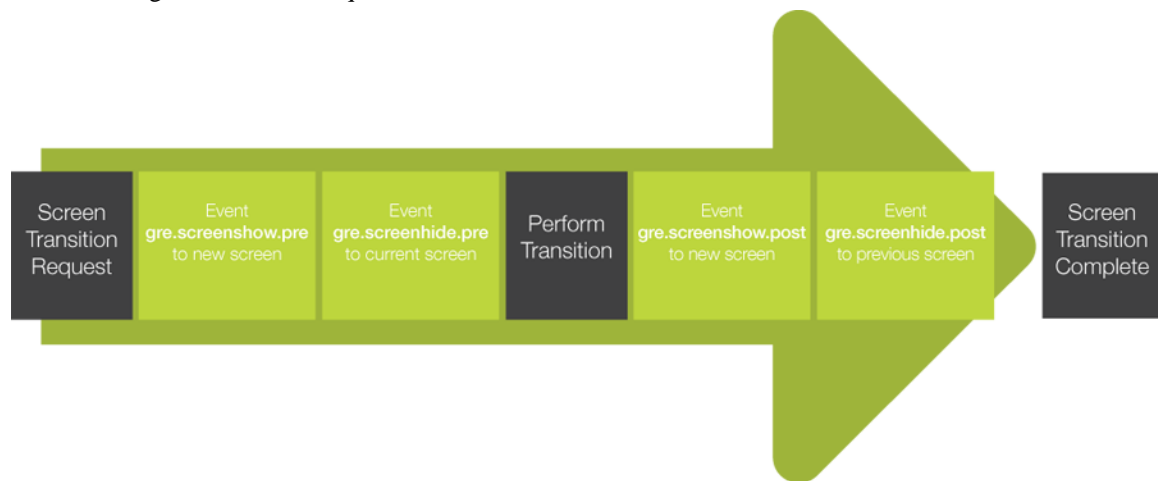
During a screen transition, four events will be generated to notify the system of the current state. These events are:

<code>gre.screenshow.pre</code>	This event is generated for the new screen being shown. The event will be generated before the transition starts. This event gives the user a chance
---------------------------------	--

to change data via the `gra.datachange` action or Lua before the transition content is updated.

<code>gre.screenhide.pre</code>	This event is generated for the previous screen being hidden. The event will be generated before the transition starts.
<code>gre.screenshow.post</code>	This event is generated for the new screen being shown. The event will be generated after the transition has completed.
<code>gre.screenhide.post</code>	This event is generated for the previous screen being hidden. The event will be generated after the transition has completed.

The following illustrates the sequence of events:



The transitions are written such that if graphics hardware layer support is available, then these layers, assuming they are available for use, will be leveraged to lower the processing overhead for the system during the transition period. Experience has demonstrated that it is possible to achieve smooth transitions at almost no CPU cost when the hardware capabilities can be properly leveraged.

Scripting

The Storyboard action operation behaviour, based on events and actions, provides a limited amount of logic capabilities. When more sophisticated glue logic is needed to control behaviour, a scripting language can be used to interact with the Storyboard environment.

Scripting support is provided through action plugins and there are no limits to which languages may be used. The default scripting language that is provided with the Storyboard framework is Lua (www.lua.org).

Lua was selected for its small footprint, high performance, and its ability to be quickly integrated with custom extensions through a well-defined C module programming interface.

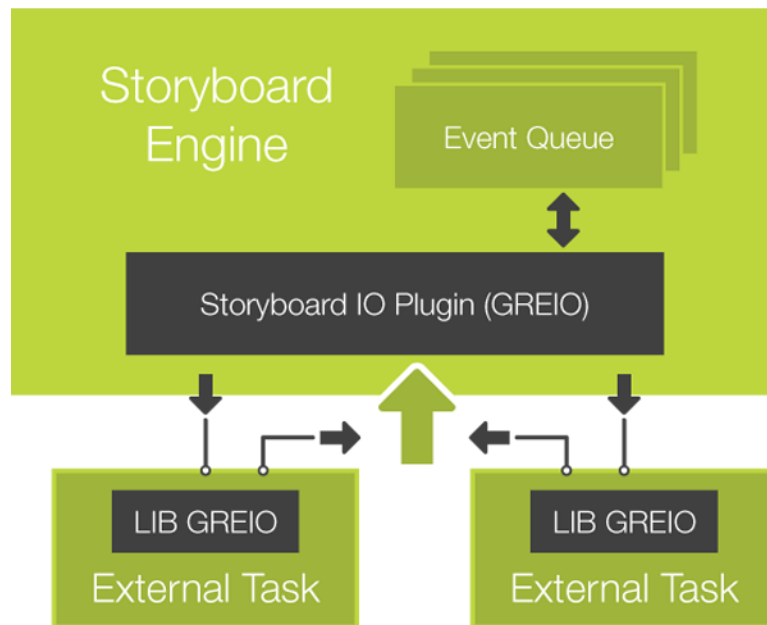
For more information about the Storyboard Lua integration, refer to the Scripting with Lua section of this document.

External Communication (Storyboard IO)

Communication with external processes in the embedded system can be accomplished in several ways. One approach that provides a strong API while maintaining a loose coupling for the implementation is to use Storyboard IO.

Storyboard IO, historically known as GREIO, is provided as a plugin for the Storyboard Engine, as well as a C API and library for external applications to use.

When the Storyboard IO plugin is loaded a channel is created in order for processes to inject events into the system. A single event queue is used to serialize the events and therefore any events sent via Storyboard IO will be placed in the queue with standard Storyboard system events. If the external application wishes to receive events, it can create its own Storyboard IO channel which can have events sent through. Applications can have multiple receive channels and the Engine has a single input channel. The following diagram illustrates an application which can send events to the Engine and receive events on a named channel.



For more details about the client Storyboard IO API, refer to the Storyboard IO API section of this document. For more details about the Storyboard IO action, refer to the Action Definitions section of this document.

Performance Considerations

All actions are executed within the context of an event delivery and as such their execution will have an impact on the overall throughput and responsiveness of the system. In particular with Lua scripts, it is important to limit the length of time that functions take to perform their work or to separate lengthy operations into separate tasks, threads or processes depending on the operating environment being used.

The screen manager listens for data changes and checks the state of controls to determine when the display needs to be refreshed. If the data for controls is changing rapidly this may cause thrashing of the display and possible flicker if not using double buffering. When changing data values, moving controls, or generating events which would cause the display to be updated, it is advisable to hold the screen manager updates until all changes have been made. Once modifications are complete the screen manager can be released and the display updated is needed. The actions are as follows:

1. `gra.screen.hold`
2. `gra.screen.release`

Chapter 2. Storyboard Designer

Introduction

Storyboard Designer is a design and development environment for creating full-screen applications ready for deployment to embedded environments using the Storyboard Embedded Engine.

Storyboard applications are designed to be full-screen user interfaces that are designed by graphic artists and designers. Storyboard Designer incorporates graphic content directly into the application design process.

Storyboard Designer allows graphic designers to import their artwork and design files as images directly into the development tool rather than trying to skin desktop style pre-configured widgets. The imported images (gif, jpeg, png and psd formats are all supported) are used as control surfaces that application developers can bind action behaviour to, based on externally generated input events.

Designer Environment

Storyboard Designer is intended for use by both graphic designers and embedded applications developers and is based on the extensible Eclipse framework (www.eclipse.org) [www.eclipse.org].

Graphic designers have a rich set of development tools to create and manipulate images. Storyboard Designer is not meant as a replacement for these tools, but is intended to provide a binding environment where static images can be animated into multi-screen applications by allowing graphics designers to easily import their work into an application design.

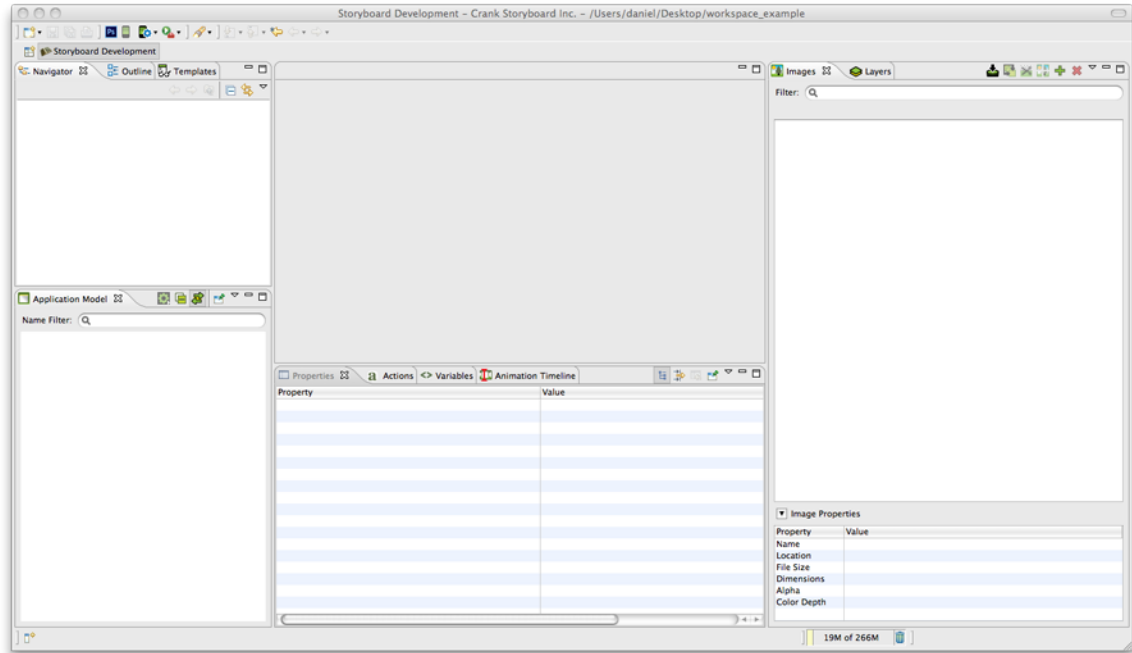
Embedded software developers typically work in C or C++ development environments. Storyboard Designer integrates into the CDT, the most common Eclipse based embedded development environment, so application user interface development can easily be done side by side with other embedded software development.

Storyboard Designer Workbench

When Storyboard Designer starts, the user is presented with an initial empty working environment for application development as shown in the following image. Storyboard Designer presents the user with a main editing area that displays a visual, WYSIWYG, representation of the application screens as they are being developed. The editor is the primary interface for development and design of the application. The editor is opened, like any other standard editor in the Eclipse environment, by double clicking on any Storyboard Designer file or right clicking and selecting **File > Open**.

The editor area is surrounded by dockable views that present editing information to the user as the application is being developed. Many of these views, such as the layer or application view, will provide information relative to the selection in the current editor. Additional views can be added into the current display by selecting **Window > Show View** and then selecting the additional views.

The selection of views and their arrangement around the editor area is called a perspective. The default Storyboard Designer perspective layout can be customized by dragging, resizing, and re-docking views in an arrangement that is convenient to individual developer or designer workflows. It is always possible to reset the layout of the perspective to its default by selecting **Window > Reset Perspective** from the main menu.



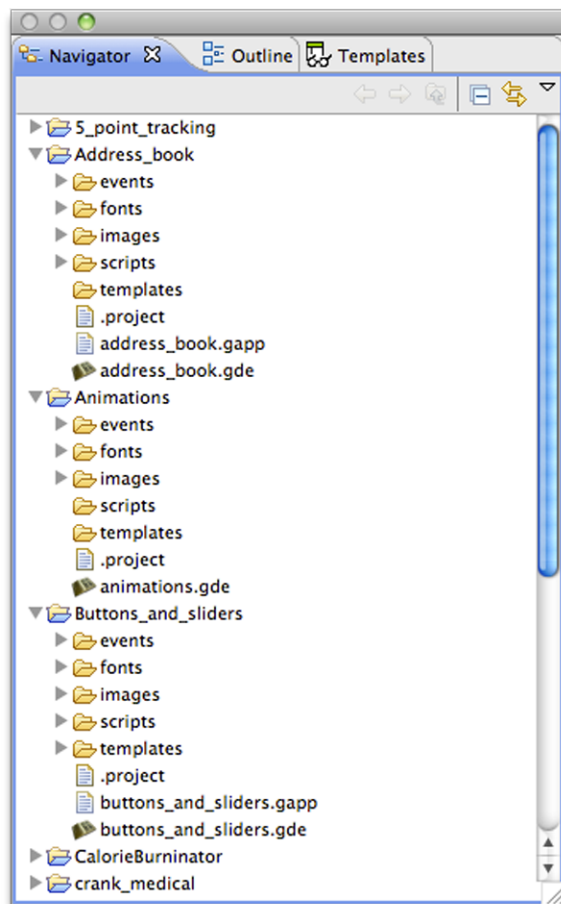
For more details on configuring the workbench refer to **Help > Help Contents > Workbench User's Guide**

Eclipse is an extensible framework with a rich set of plugins available from multiple software vendors. Among other integrations, team collaboration plugins for GIT, Mercurial, ClearCase, SVN and CVS are all readily available. The Eclipse marketplace (marketplace.eclipse.org) contains a comprehensive listing of available plugins and extensions.

Anatomy of a Storyboard Designer Project

Storyboard Designer manages its projects within a filesystem directory referred to as a workspace. The workspace is used to limit the scope of file resources to just those files in the host filesystem that are relevant for the application(s) design. Storyboard Designer projects correspond to the root directories contained within the workspace directory.

When a new Storyboard project is created, using **File > New > Storyboard Application...**, it creates an initial project structure in the workspace that contains several default directories in addition to the main Storyboard application design file.



The images, fonts, scripts, templates and events directories are automatically scanned for content and that content is integrated into the application designer tools. In order to import content from the filesystem into these directories, you can use the **File > Import > General > Filesystem** option or the standard system copy and paste or drag and drop from other applications.

Each directory scans for a different type of content:

- | | |
|--------|---|
| events | This directory and its sub-directories are scanned for event definition files which are text files that have an extension of .evt. The events contained in the event definition files are then automatically included in the action trigger event list. Event definitions are generally automatically managed by the Designer framework when new events are added or removed using the New Action Wizard. |
| fonts | This directory and its sub-directories are scanned for TrueType™ font files. In general, these font files have the extension of .ttf. The fonts discovered are automatically added to the list of available fonts in the font selection dialog. OpenType™ and TrueType container formats are not supported by Storyboard at this time. |
| images | This directory and its sub-directories are scanned for image file content. Supported image file formats include gif, jpeg, bmp and png files. Photoshop™ PSD files can be imported directly as an application or as only the component images using the File > Import > Storyboard Development > Photoshop PSD File menu option. |

scripts	This directory is scanned for Lua (www.lua.org) [www.lua.org] scripts which have the extension of <code>.lua</code> . The functions that are found in these scripts are automatically added to the list of available functions presented in the Lua action argument configuration.
templates	This directory is scanned for Storyboard Designer template files. Valid templates are automatically added to the list of available templates or new actions. For more details on creating and working with templates, refer to the document sections Working With Templates and User Defined Actions.

As changes are made in the filesystem, the workspace should refresh automatically and the changes be reflected in the Storyboard Designer user interface. An automatic refresh may be delayed due to system activity and can be forced at any time by selecting a project or file in the Navigator view and selecting **Refresh** from the right click menu.

Storyboard Simulator

Included with Storyboard Designer is a simulator environment that is a host compiled version of the Storyboard Embedded Engine. The engine is included with the development environment to provide direct feedback about the non-static portions of the application such as its screen transitions and animations.

Since the simulator is in fact Storyboard Engine running in the host environment, what is presented as a simulation is the same graphical assets and data model that will execute on the embedded target.

Storyboard Designer Editor

The Storyboard Designer editor is the central location for all design activities for your application. It provides a visual representation of all of the screens of the application and allows designers to edit the screen content and get immediate feedback about what the look and feel of the application will be.

Editing Content

The default editing mode is to display and edit the entire application, showing all screens and their composited layers together.



If, instead of looking at all of the screens of an application together, you want to focus on editing and working with one particular screen, then you can right click in the editor and select **Edit > Screen**, which will open up a new editor window with just that screen's content shown.

If you want to edit just the layer contents, independent of the screens to which they are bound, then you can right click in the editor and select **Edit > Layers** and this will open up a new editor with all of the selected layers shown individually. If changes are made to a layer in this mode, the change will be reflected in all of the screens that reference the changed layer.

The right-hand side the editor contains a fly-out palette toolbar that provides the basic visual design elements for the application; screens, layers and controls. These can be selected and dropped onto the editor to start building up the application.

Additional editing functionality is available through the right-click menu while in the editor, as well as from the main menu. This is where you will find functions to manipulate control size, alignment, and z-order/front-back placement, as well as the creation of new controls, layers, and screens.

Content can be moved within its container by selecting one or more items and then using the arrow keys to move the item. By default, the movement is in 10 pixel increments when using the keyboard, but if the **SHIFT** key is held down while using the arrows the content will move one pixel at a time in the desired direction.

There are a number of keyboard shortcuts for common operations these can be displayed on-screen by selecting **Help > Key Assist...** from the main menu.

Editor Toolbar



In addition to the editing options available right click menu, the toolbar provides functionality that is context sensitive to the editor being used. When a Designer file is being edited (and the editor area has focus), then the toolbar provides short-cuts to several common operations.

Storyboard Simulator	This will export the Designer file being edited to a Storyboard Engine file and simulates it using the host based Storyboard Engine configuration.
Zoom Display	This controls the current zoom level of the display. This value can also be adjusted by CTRL+MouseScroll or COMMAND+Scroll using a wheel mouse or touchpad.
Align and Resize	These toolbar actions provide a convenient alternative to manual alignment by aligning the selected controls automatically with one another. When a single control is selected the alignment is performed relative to the screen. When multiple controls are selected, the alignment is performed relative to each other.
New Model Elements	These toolbar actions provide an alternative to the palette for the quick construction of screens, layers and controls.
Control Outline	These toolbar actions control how the control content is displayed within the editing environment. By enabling the control outline, a border will be drawn around controls and layers. By enabling the wireframe mode, no control content will be drawn, but an outline of the controls is drawn. These modes can be used to optimize the application layout to avoid unnecessary damage and redraw operations.

Wireframe

The wireframe mode turns off all of the render extension drawing within the controls and shows only an outline of the controls, layers, and screens similar to what is provided by the Control Outline functionality.

This functionality is useful to minimize the amount of control overlap that occurs so as to prevent excessive redraw damage areas.

Direct Editing

In addition to the standard model editing functionality, some render extensions have direct edit functionality. To enable the direct edit on a control with one of these render extensions, the user needs to do a slow double click on the control.

3D Model

When the direct edit mode is activated, a set of xyz axes will appear in the bottom left corner of the control. At that point the user can use their mouse to move the model. Holding the shift key allows the user to rotate the model using their mouse. As well, holding either the x, y, or z key isolates that axis when either translating or rotating the model. When the user is finished editing the model, they just need to click anywhere outside of the control to end the direct edit.

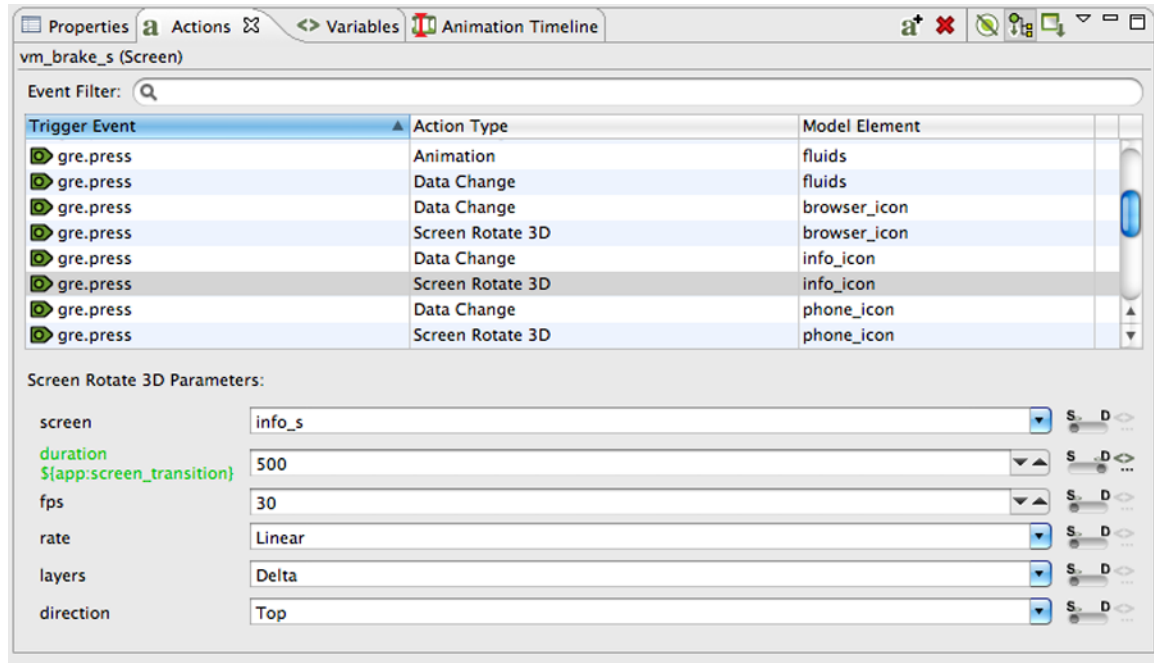
2D Polygon Editing

When direct edit is activated, the user can create and edit a polygon on the screen using hotkey toggles. Hold 'shift' and click within the control bounds to create new vertices (ideally, 3 vertices are needed to have a sufficiently visible polygon). Hold 'control' and click on a particular vertex to delete it from the polygon. Hold 'alt + shift' and click near an edge to create a new vertex splitting that particular edge. Vertices can be moved by simple click and drag. Exit quick edit by moving mouse out of the control bounds.

Storyboard Designer Views

Actions View

The Actions view provides a display of all of the available actions that are in context for the given selection in the editor.



Actions can be added through the right-click menu in the editor, **Add > Action...** or in the Actions view directly.

The content of the action list can be sorted by selecting the action table title. When actions are sorted by their triggering event, the order in which they appear will also correspond to the order in which they will be evaluated within the same context. If two actions are bound to a `gre.press` event on the same control, for example a Data Change and then a Screen Fade, then the first action in the list will be executed (Data Change) before the next action (Screen Fade). The order of these events can be adjusted by right-clicking the event and selecting **Move Up** or **Move Down** as required.

The content of the action list is automatically populated based on the Designer model object selected in either the editor or in the Application Model view. The content of the list can be populated in several ways:

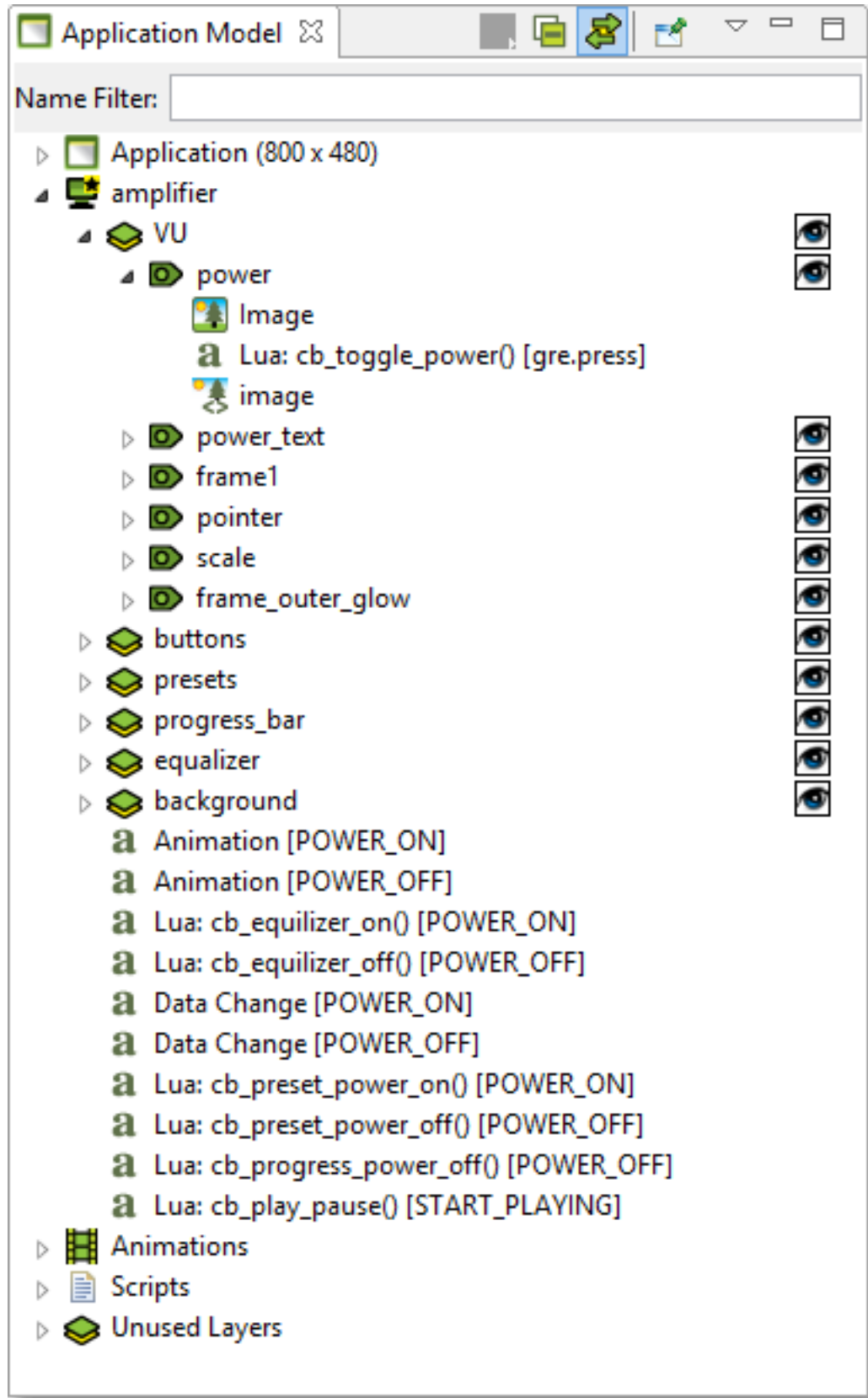
- | | |
|-----------------------|---|
| Selection Only | This will show only the actions associated directly with the selected model object. |
| Sub Hierarchy | This will show the actions of the selected model object and all of its child model objects. |
| Application Hierarchy | This will show all of the actions in the project |

In addition to controlling how the list is populated using the toolbar selections, it is also possible to use the name filter at the top of the list to match against specific event names. This is particularly useful when used in conjunction with the Application Hierarchy to search the entire project for custom events.

The triggering event, action type, and context can all be edited inline in the action table. Each action also has its own set of parameters or configuration options. These values can be changed in the lower display area of the Action view once an action selection is made. When the action types are changed, as many argument values from the original action will be migrated to the new action as long as the argument names and types match.

Application Model View

The Application Model view displays a tree representation of the model objects that make up the Storyboard application.



The tree representation aligns with the Storyboard model representation and allows editing operations to be performed on elements that may not necessarily be visible within the editor. For each of the application, screen, layer, and control objects the Application Model displays the Actions and Data Variables associated with that model object.

The visibility of the layer instances and controls can be quickly adjusted through the Application Model view by toggling the setting in the visibility column. The changes made here will be immediately reflected in the editor and will also be reflected in the Storyboard Engine runtime file as the initial setting for the layer instance or control.

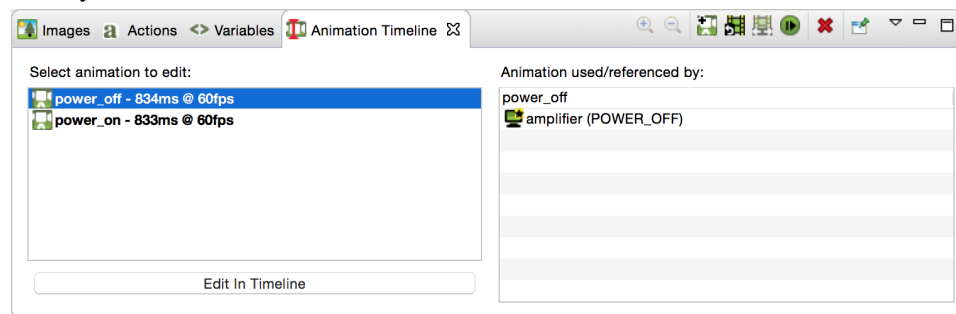
Since layers are displayed as layer instances, the tree will show layer and control content several times in the tree. If the Link with Editor toolbar option is enabled, when a model object is selected in the Application Model view the editor will automatically scroll to present the appropriate context of their selection. The same behaviour can be achieved by double-clicking on a model object if the view's content is not synchronized with the editor.

The Application Model view tree also displays all of the scripts, animations, and unused layers that are currently a part of the application design.

You can copy and paste model elements, such as screens, layers, controls, and actions, from one application to another using this view.

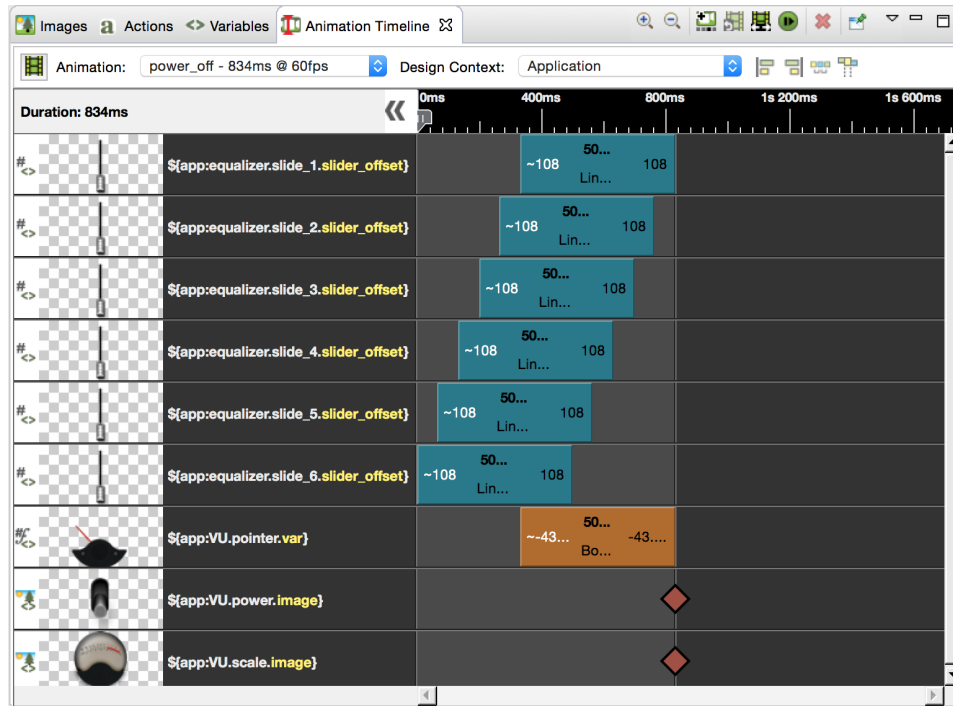
Animation Timeline View

The Animation Timeline view provides an editing environment for creating the animation blocks that are used by the animation action.



The initial Animation Timeline presentation displays a list of all of the animation blocks defined in the application. Those animations whose names are highlighted in bold represent animations that are currently referenced. The list on the right is updated, based on the selected animation, to show a list of all of the locations in the project where this animation is currently referenced.

Double-click on an animation or select the *Edit in Timeline* button to change the display mode of the Animation Timeline from a list of all animations to an timeline based editor focused on the selected animation block. To create a new animation, right-click in the list and select *Add Animation*, or select the *Add Animation* from the toolbar.



The timeline editing mode displays a list of the variables that are being modified during the animation and a timeline view indicating the starting time offset and step duration for a particular animation step relative to all of the other animation steps running in this block.

A single click on the animation step will show the properties view where values can be modified.

Property values can be edited / calculated inline.

Double clicking beside the animation step will create an exact copy of that step.

A click on the animation step will display a pop up window to show all its information when the step is too small to do so.

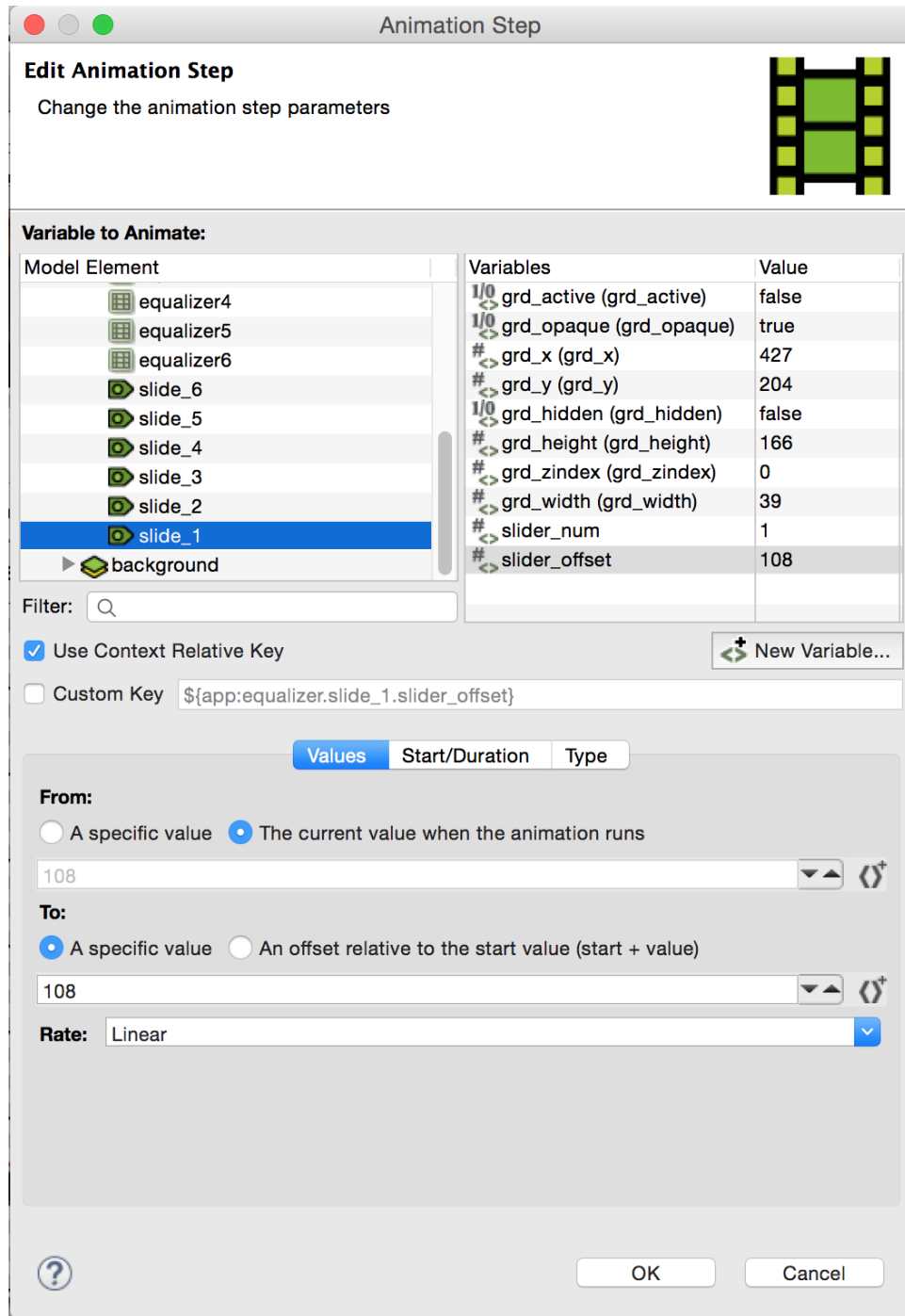
Select multiple animation steps and use the align left, align right, distribute, or pack tools to easily line up the animation steps in the timeline.

Using the 'Design Context' drop down, a context can be selected that the animation will use to resolve variables in. This context will also be used to determine what screen displays during an animation preview (see below).

Right click on an animation group to duplicate its functionality and assign it to another variable.

You can drag and drop the block within the display to adjust the starting offset or lengthen or shorten its duration by resizing the block. The start, end values, and the rate of change can be adjusted inline by double-clicking the values or by right-clicking the block. Animation steps can also be reordered by dragging and dropping them in the list. For animations with many steps, you may want to choose the compressed display option from the menu.

If you want to make a more significant change to the animation step, or to change several values all at once, select a step and click on the 'Change Variable...' button in the properties view and the **Animation Step Dialog** will open. You will also get this dialog if you right-click and select *Add Animation Step* to add a new variable to animate in this block.



The *Animation Step Dialog* allows you to select the variable that you want to animate and then fine tune all of the characteristics of the animation step including how you want the values to change and when.

Working with Animations

There are a number of ways to get started creating animations. It is always possible to manually create a new animation from the *Animation Timeline View* and add the variables to the animation one-by-one, however Storyboard provides a few shortcuts to make this process easier.

Record Animation



To create new animations, Storyboard provides a mechanism that allows you to record a series of changes to your application and have those changes automatically incorporated into a new animation. This is called **Animation Recording** mode and it can be accessed in the Storyboard Editor from the top menu bar.

The animation recorder will track all changes that are made to model objects in the application, and when the recording is finished, it will automatically create new variables for those static values that have changed. It then gathers all of the changed variables and places them into a new animation ready to replay that visual change.

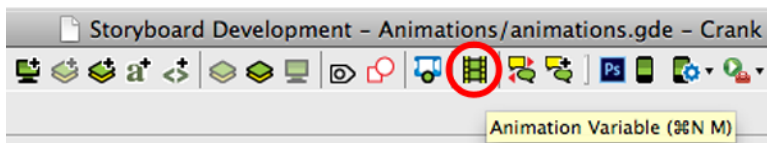
While recording, you have the ability to take snapshots of it throughout. Taking a snapshot will cause subsequent steps created by changes during the recording to be offset by the duration of the previous step. This way, you can create a sequence of consecutive animation steps rather than having all of them execute simultaneously. The snapshot action is located right next to the recording action, and will only be enabled if you are currently recording.

Note

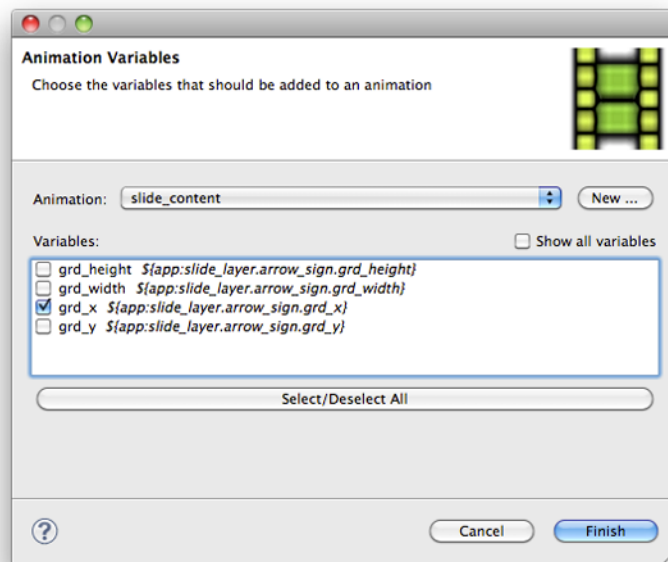
When using the animation recorder, any structural changes made to the model will be automatically reverted and lost at the end of the recording. The model recording only tracks changes to existing model attributes and variables, creating variables automatically for any static elements that change.

This means that if during the recording any controls, variables, actions, or any other model elements are created, they will not be captured in the recording and they will also not be present in the model when the recording is stopped.

Add Animation



To quickly add new variables to either a new animation or to an existing animation, you can use the **Animation Variable** command that is available from the editor toolbar or from the right-click menu on many model objects.



When this option is used, a dialog will open that displays all of the variables that are associated with the current selected object. You can quickly select those variables you are interested in, select the animation that you would like them to be applied to, and immediately start fine tuning that animation.

Preview Animation



Preview the currently selected animation by clicking on the animation preview action. This will open up a new dialog in which an un-editable version of the application will be displayed and the animation will run. The bottom bar contains actions that control the preview. Play/Pause will stop and restart the animation from the current frame. While the preview is paused, use the Fast Forward/Rewind buttons to move ahead/back one frame at a time. Interact with the progress bar to jump to any frame and click on the Replay button to reset the animation to frame 0 in preparation for another run.



By default, the preview window will guess the screen to display by looking at the variables being animated. However if the user has selected a design context for the animation in the timeline, that context will be used to determine the screen to display in the preview.

Images View

The Images view provides a thumbnail presentation of those images that are currently included in the application design.



The content for the Images view is automatically pulled from the image file content contained in the *images* directory of the Storyboard project. New content that is imported into this directory in the filesystem will, upon a workspace refresh, be automatically shown in the Images view. Supported image file formats include PNG, JPG, GIF and BMP files. In order to import Photoshop PSD file content, the images must be imported using the **File > Import > Photoshop PSD File** wizard.

You can drag and drop images from the Images view directly into an application design. By default, a new control will be created that matches the image's size and the image source is a static value pointing at the drag and dropped image.

Within the Images view it is also possible to quickly switch an image that is used in one context to another using the **Swap Image With...** right-click menu. For example, if a number of controls were using an image as a button background and a new image was available that provided an updated look, then after importing the new image one could select the existing image and choose **Swap Image With...** to quickly change all instances within the application to the new look.

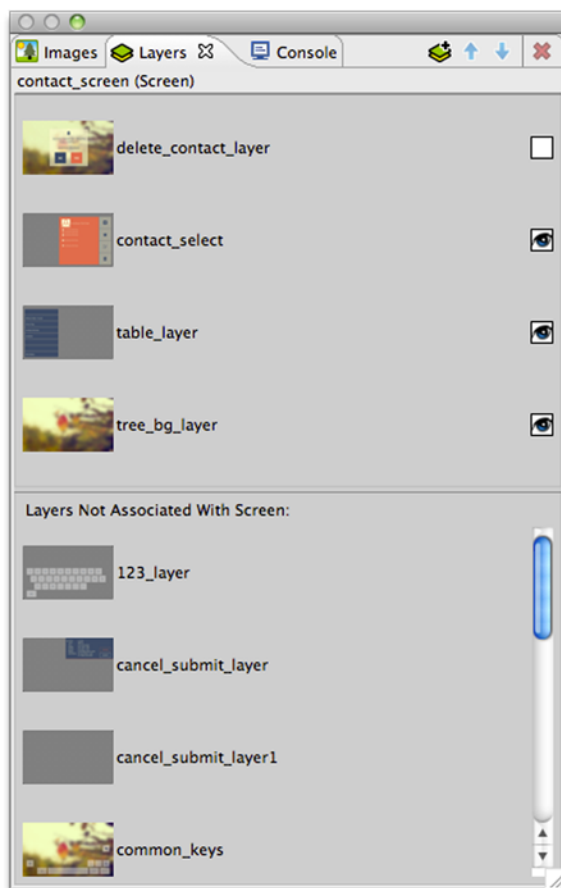
All files contained within the *images* folder will attempt to be processed as images, regardless of their file extension. If an unrecognized image file is encountered, by default it will not be displayed. This behaviour can be adjusted by de-selecting the **Show Only Images** in the view's drop-down menu. This menu also provides the ability to group images by directory which is a convenient way to classify images that are used in different parts of your user interface.

As an application evolves, it will often accumulate a number of unreferenced or duplicate resources. The Image view offers a few utilities to help manage these images.

- The **Resource Clean Up** toolbar action allows the pruning and deletion of unreferenced images from the application design.
- The **Consolidate Images** toolbar action identifies duplicate images, references all consolidated into a single resource, and removes duplicate images from the workspace.
- The **Trim Images** toolbar action works on selected images to remove all of the extraneous transparency that surrounds non-transparent content. This can significantly impact the performance of rendering on systems without hardware graphic rendering capabilities.
- The **Split Images** toolbar action works similarly to the *Trim Images* option but transforms a single image with significant areas that are completely transparent into multiple images with that transparency trimmed away.

Layers View

The Layers view provides a thumbnail display of the layers used by the currently selected screen and also displays a thumbnail of all of the other layers that are available for use but are not bound to the currently selected screen.



Layers are categorized as either being included on the current screen or not. Layers can be added or removed from the current screen by dragging and dropping them either into or out of the current screen area, or by using the delete key.

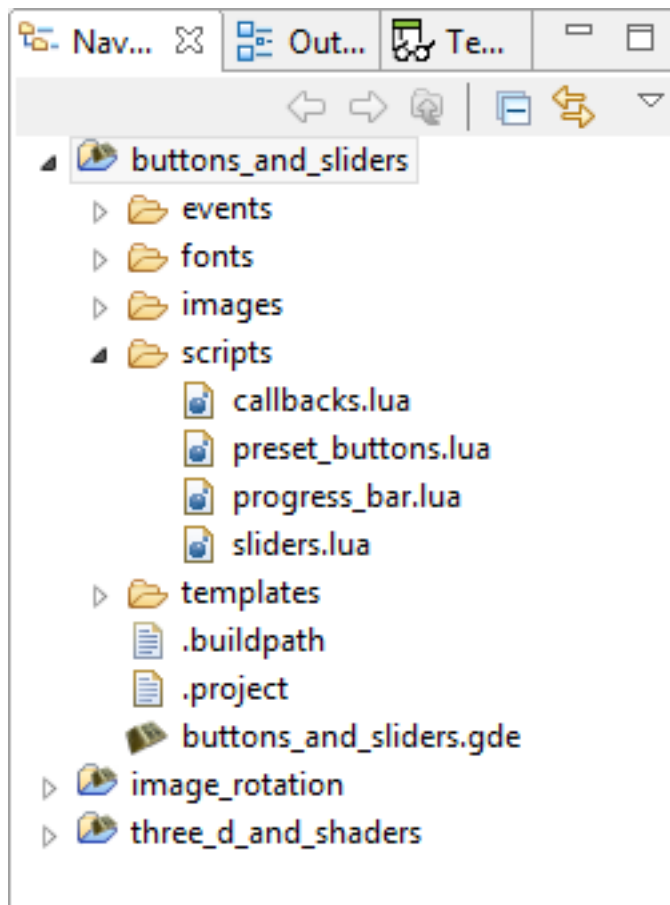
Layers are listed according to their front (top) to back (bottom) order z-order presentation. This order can be manipulated by dragging and dropping the layers within the view or using the toolbar buttons. In

addition, the Layer view provides the ability to change the visibility within the application. If the visibility or z-order are changed, the change is immediately reflected in the editor and will also be reflected in the Storyboard Engine runtime deployment.

Using the toolbar controls in the Layers view it is possible to create new layers and also to open up the Layer editor mode to work with layers independent of the screens with which they are associated.

Navigator View

The Navigator view is a standard filesystem style explorer limited to showing only content available in the Storyboard workspace.



The Navigator view only displays content that has been imported into the workspace and starts with the top level Storyboard Project directories that have been created. Since the workspace allows multiple Storyboard projects to be shown, it is possible to work on multiple projects concurrently all within the same workspace with multiple editors open targeting different Storyboard projects. Content and resources can be copied and pasted among the different editors.

The Navigator view provides a variety of filters to hide/show different file types as well as the ability to group projects together as 'working sets' and then to only display the content from those working sets. For more details on configuring the Navigator and filtering workspace content refer to the Eclipse help **Help > Help Contents > Workbench User Guide**.

Outline View

The Outline view provides an overview of the entire editor. The outline content will change to reflect an outline of the editor that currently has focus. Within the Designer environment there are two Outlines of interest, the Storyboard editor outline and the Lua editor outline.



The Storyboard editor outline displays a scaled visual presentation of the entire contents of the editor. If the editor is in Application mode then all of the screens of the application will be shown. If it is in Layer mode then all of the layers will be shown, and similarly for the other editor modes. By moving the highlighted area within the Outline view it is possible to change the viewport of the current editor.

The Lua script editor outline displays a listing of all of the identified functions in the file and supports the quick navigation to those functions by double-clicking the function name.

Problems View

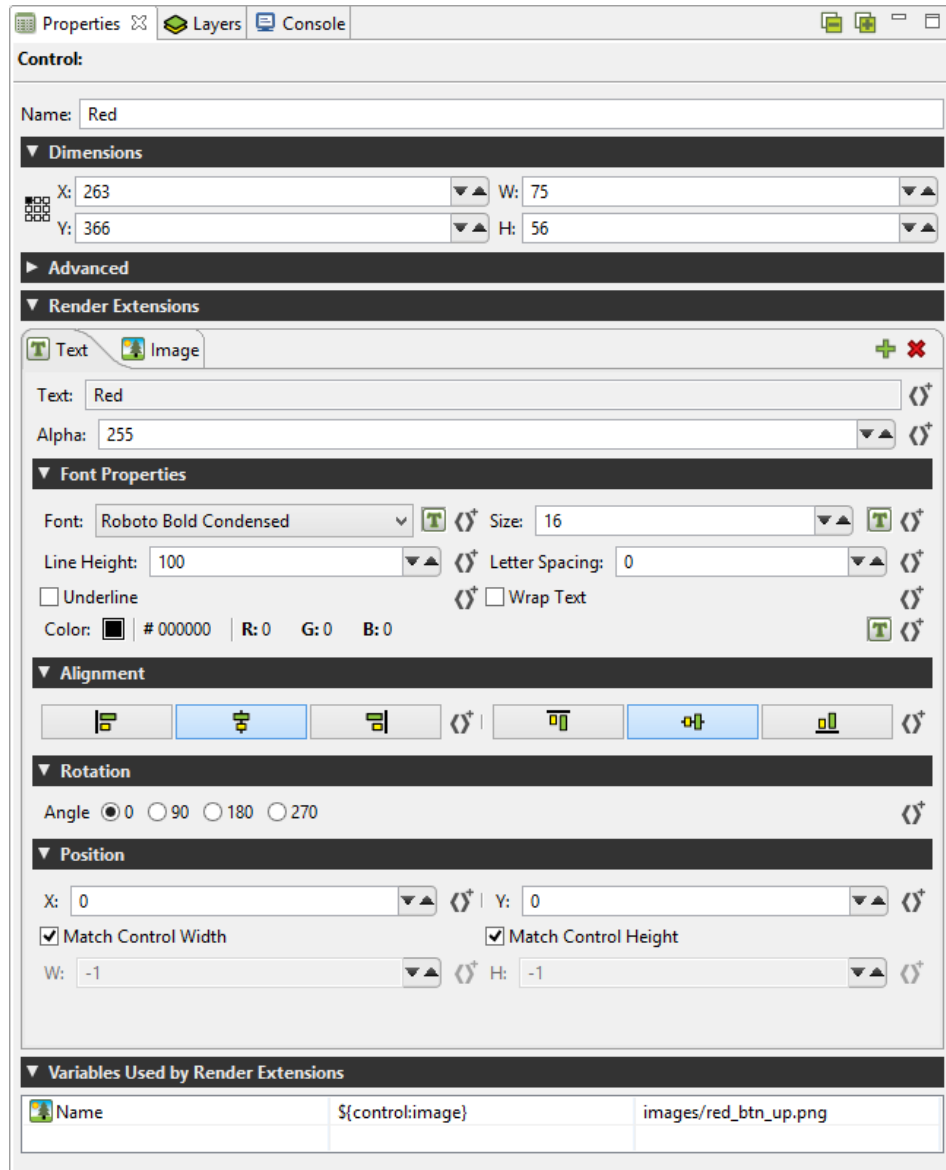
The Problems view shows a list of all problem markers that are created within the workspace. Storyboard provides a project analysis mode that runs in the background to examine Storyboard models and report on design concerns. To enable or disable this analysis mode you can change the Storyboard workspace preference setting in **Windows > Preferences > Storyboard > Enable Background Storyboard Project Analyzers**.

When the Storyboard Project Analyzers are enabled as workspace resources change the model will be scanned and analyzed and discovered issues will be reported into the Problems View. Issues that are scanned and reported include:

- Mismatched image color depths. If a 32bit color image is used with a 16bit color display there is a possibility the image will be distorted on the target
- Fill hides content. When a fill masks on top of other content, this can be an ineffective use of processing resources on target devices.
- Missing render extension content. For example, a control with an image render extension that is not defined or does not exist. This may result in additional computations being performed by the embedded target.
- Scaled or rotated static image content. Since static images do not change at runtime, the work required to scale or rotate an image could be replaced by a fixed cost of rotating or scaling the image during the design. This will reduce the amount of processing required at runtime.

Properties View

The Properties view displays information about the current selection in the editor and also provides the ability to change and adjust the properties of that selection. The Properties view is the primary editing location for fine tuning the visual presentation and adjusting data bindings of the application.



For each model element, application, screen, layer and control, there is a different property interface that provides access to those items that are most relevant to the selected context. The following is a non-exhaustive list of some of the property pages.

Application	The application properties, active when the editor background is selected, displays the application name, size, and color attributes.
Screen	The screen properties, active when a screen is selected, displays the screen name and indicates if the screen is the start screen for the application.
Layer	Layers and Layer Instances share a common set of property pages. A Layer Instance is simply a Layer that has been associated with a particular screen. Changes made to a layer's size or its controls will be propagated to all layer instances. Position, opacity, and 3D rotation attributes are properties that are not associated with the layer but are associated with a layer instance.

Group	Group properties contain the name of the group as well as information about the group's origin. In this property panel you will also find the functionality for automatically re-configuring a group's origin based on it's control content. This is very usefull when you are taking a group and then converting it for use in a more generic Storyboard Designer template.
Control	Controls contain the most sophisticated property pages, because in addition to the name, size, and position information the property page also contains all of the configuration parameters for the render extensions associated with that control. The render extensions are listed in the Z order (front to back) that they will be rendered within the control and this can be adjusted by dragging and dropping the render extensions within the list entry.
Render Extensions	Render extension property pages show the argument details of the selected render extension. This is the same information as is shown within the Control's property panel, but without all of the additional details associated with the control.
Actions	The property pages for actions show the parameters that are available for editing and the presentation changes based on action type. The content that is shown here is the same as the information presented in the Action View but in the case where multiple actions are selected, the content can be changed across the entire selection.

When multiple elements are selected, the Properties View will try and show the most suitable content possible. If all of the selected elements are the same, then the properties view will display the common properties and any changes that are made will apply across all of the selected elements.

In certain cases, such as when multiple controls with different render extensions are selected, it may not be possible to provide a completely synchronized display. In these cases the display will show a common set of attributes and hide the attributes that are not common among the selected elements.

Templates View

The Templates view provides users with a list of Storyboard Designer templates that are available for use in this project. The list of available templates is generated automatically from the contents of the project's *templates* directory.

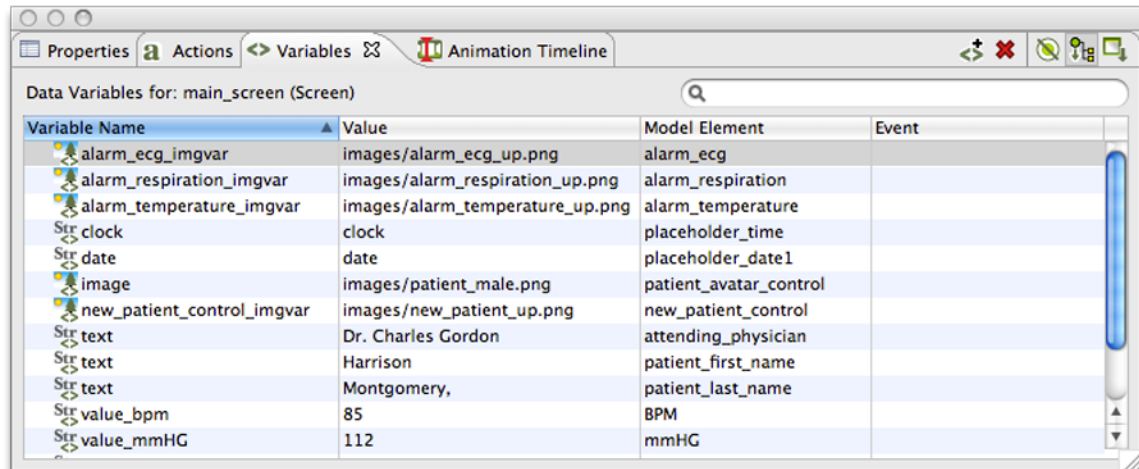
When a template is selected, its description along with a graphical preview of the template will be shown if they are available. To create a new model object (control, layer, screen etc) that is template-based, select one of the template items and drag and drop it into your application.

Not all templates create new Storyboard model objects. Some templates simply enhance the functionality of an existing object, and when this is the case, when a model object is selected you can right-click on the object and select Templates Apply to access a list of available templates.

For more information about creating templates, see the document section entitled Working With Templates.

Variables View

The Variables view is similar to the Actions view in that it displays all of the data variables that are in the context of the current selection.



Once a variable is defined and associated with a particular model object context (application, screen, layer or control) then the variable can be referenced as a parameter for actions and render extensions.

There are two different types of data variables that can be defined. A normal variable contains a name, a type (ie number, string) and the value matching the type that should be used when the variable is referenced. To facilitate working with repetitive data within a table control, a special type of variable called a table cell variable can be created. This variable contains all of the same attributes as a normal variable, but is extended to contain additional row and column information that can be used to specialize a particular value at a given table row and column.

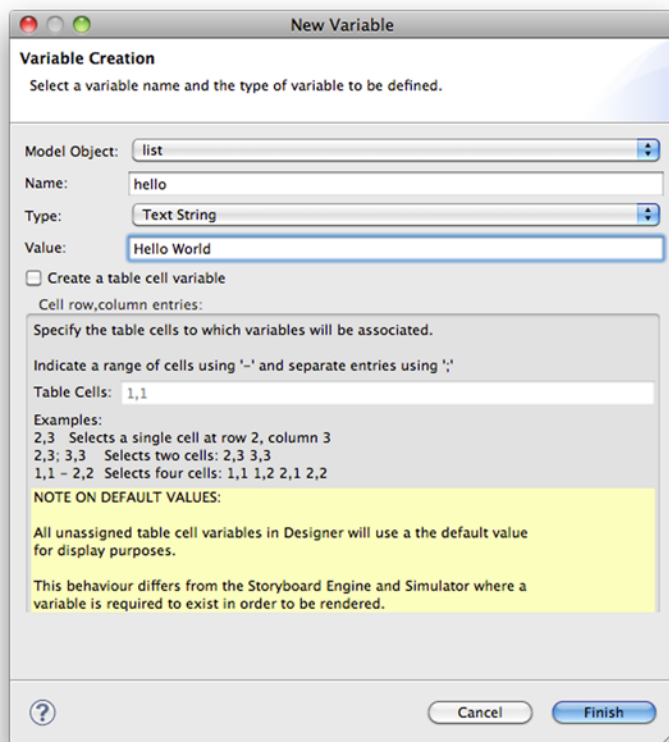
The content of the variable list can be sorted by selecting the appropriate variable table title and the variable values can be edited inline in the variable list by double-clicking on the appropriate field that you wish to change.

What variables are shown in the Variables view, similar to the Actions view, is automatically populated based on the Designer model object selected in either the editor or in the Application Model view. The content of the list can be populated in several ways:

Selection Only	This will show only the variables associated directly with the selected model object.
Sub Hierarchy	This will show the variables of the selected model object and all of its child model objects.
Application Hierarchy	This shows all of the variables in the application, regardless of what the current selection may be.

In addition to controlling how the list is populated using the toolbar selections, it is also possible to use the name filter at the top of the list to match against specific variable names. This is particularly useful when used in conjunction with the Application Hierarchy to search the entire project for a variable.

Variable Creation



New variables are frequently created at the point where they are required, for example within the property display for a render extension or the properties for an action argument. When variables are created in this context, then their types will automatically be determined from the context of use. However variables can also be created directly from within the Variables view in which case the user can select the type of the variable. Care should be taken to match the type of the variable to its intended use, for example text variables can not be used as adjustments for numeric values and vice versa.

In all cases the variable creation will open the New Variable wizard. From within this dialog you can select the name of the variable, its data type, and the value to associate with the variable. From this dialog it is also possible to create table cell variables that span a particular row/column range

By default, the variable will be created and associated with the current application, screen, layer, or control that was selected when the New Variable wizard is launched. However this association can be changed on the second (optional) page of the New Variable wizard where the variable can be explicitly assigned to a different model object.

Note

The type of a variable is important for the Actions or Render Extensions that may use them. If a variable is mistyped, such as a string variable is created but referenced in a location expecting an RGB color value, then the results are undefined. In general, it is a better idea to create variables from the Actions or Render Extensions that will be using them to ensure the proper typing occurs.

Generating Events on Variable Change

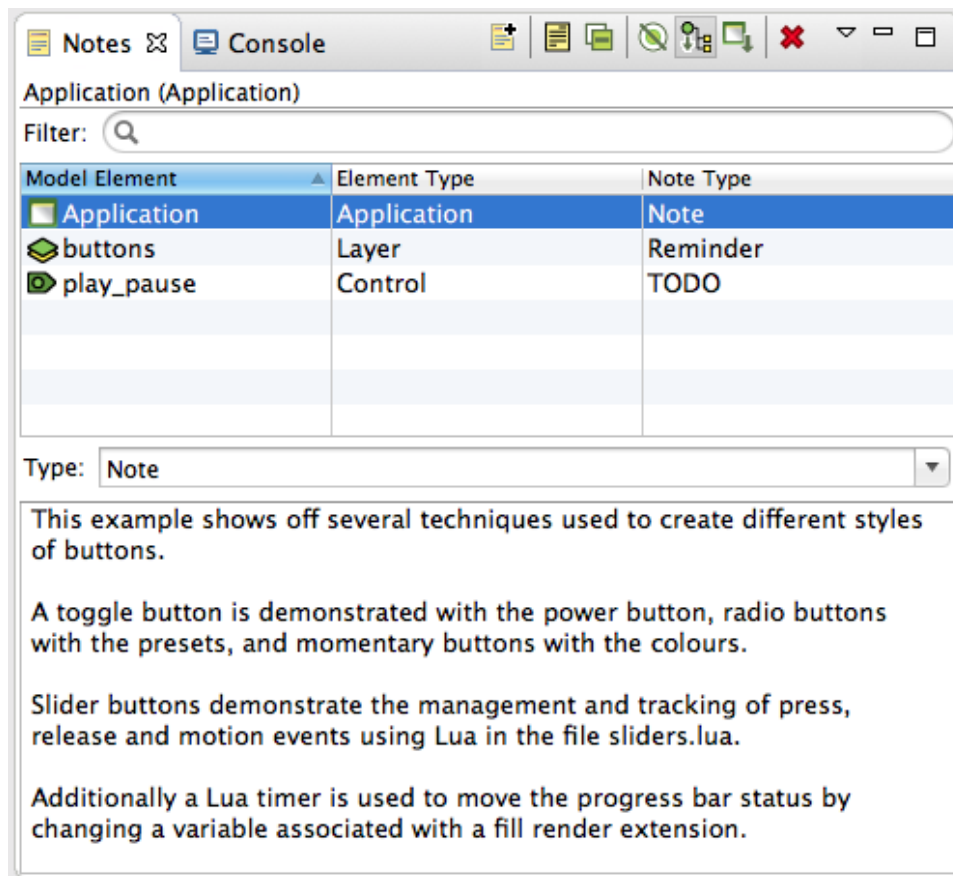
It is possible to associate a user defined event to be generated when a variable's value is changed. These events are designed to facilitate the synchronization of user interface elements that may not be directly associated with the variables whose data is changing. A typical scenario would be to monitor the position or location of a control and fire a notification when it changes in order to maintain a corresponding relationship in another control.

In order to specify the event to be generated, simply enter the event name into the **Event** column of the desired variable or select the variable from the list and right click and select **Bind Event** which will open the event definition list allowing you to pick from existing events or create a new one.

The variable change events are designed to be used to synchronize the user interface display with an updated variable value and are not meant to be used as counters for each changed value of a variable. For each variable change an event is added to the event queue only if there is not already an event with the same name in the queue waiting to be processed. Until that event is serviced, no additional events will be queued for that variable, or any variable generating the same event name.

Notes View

The Notes view displays notes attached to model objects within the current project.

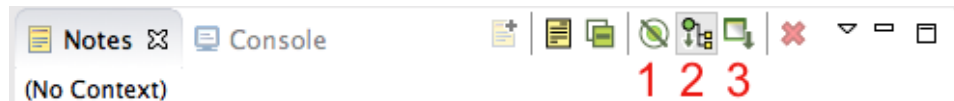


Notes contain text to help organize a project or keep track of useful information specific to a particular model object. Notes also have a type associated with them which can be selected using the combo box in the editing area. To create a custom type, simply type in the combo box instead of using the drop down.

When a note with a custom type is added, that type will be added to the combo box so that it can be quickly selected from the drop down when making other notes.

Notes attached to the current project are displayed in the table. It can display in three modes:

1. In context - only displays the note associated with the select model object.
2. Sub hierarchy - displays the note associated with the selected model object and any notes associated with model objects with are children on the selected object.
3. Full application - displays all the notes in the application.



You can search for notes by typing in the filter text box. The contents of the table can be sorted by clicking on the header of one of the columns (e.g. clicking “Model Element” will sort the notes alphabetically by the name of the model object the note is attached to).

Notes can be added to any model object through the right-click menu (New->Note...), or by selecting the model object and clicking either the New... icon in the Notes view or by clicking in the text box when it says “Click to add a new note...”. To delete a note select it in the table and then click the red X, or right-click on it in the table and select delete.

You can have the editor display the model object that a note is associated with by right-clicking a note in the table and selecting Go To->Screen.

All the notes in the current project can be quickly summarized in a list by clicking “Toggle Full View” (the yellow page icon).

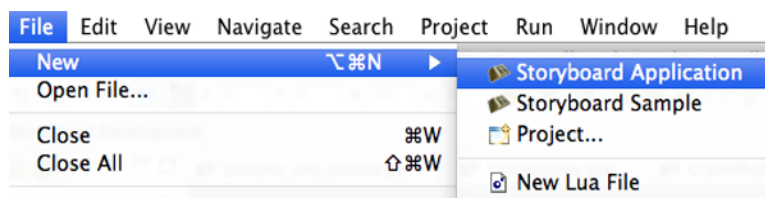


Creating a Storyboard Designer Project

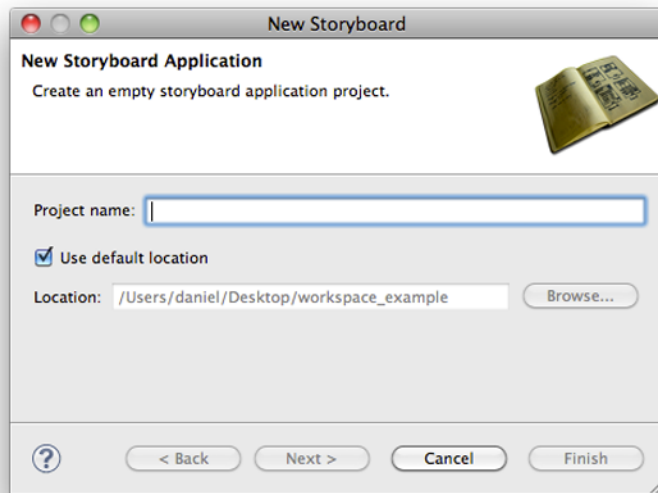
A new Storyboard project can be created in many different ways; as a new application, as a Photoshop™ import, or as an import from an existing Storyboard Embedded Engine (GAPP) deployment file.

New Storyboard Application

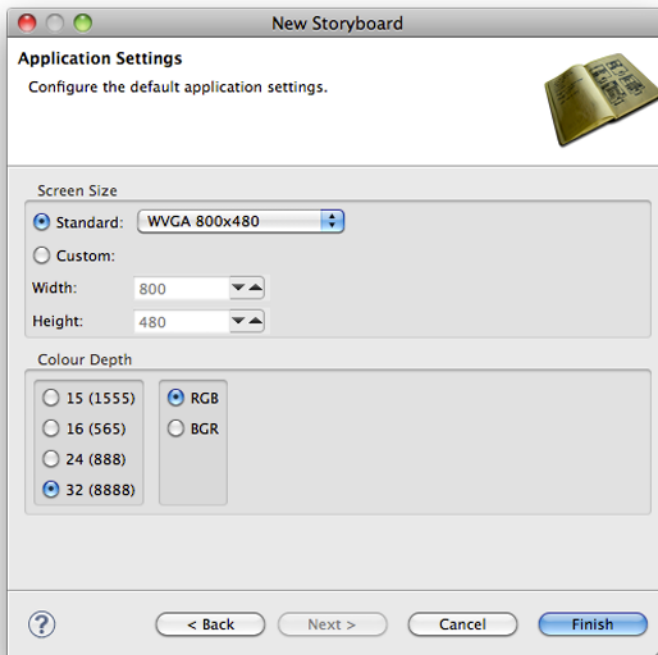
In order to create an empty Storyboard application, select **File > New > Storyboard Application** from the main menu (when in the Storyboard perspective, otherwise select Project...).



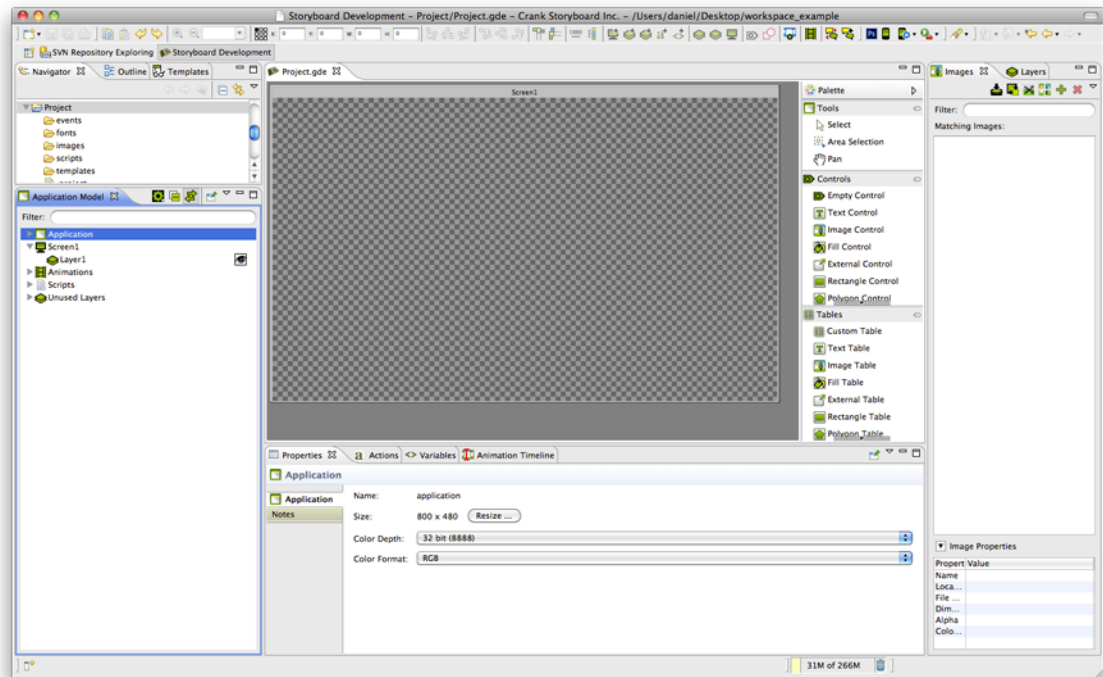
This will open up the New Storyboard Application wizard that prompts for the name to use for the Storyboard project. This name will also be used for the initial Storyboard application file.



The next page of the wizard provides the ability to customize the initial application screen settings. The screen settings, once applied, will remain locked for the design period of the application.



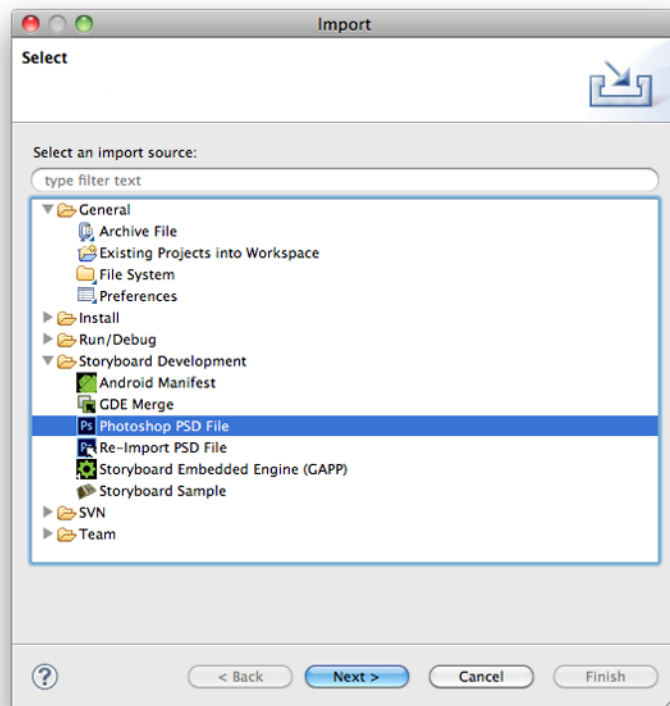
Selecting Finish will close the wizard, create the new project and automatically open the Storyboard editor on the project:



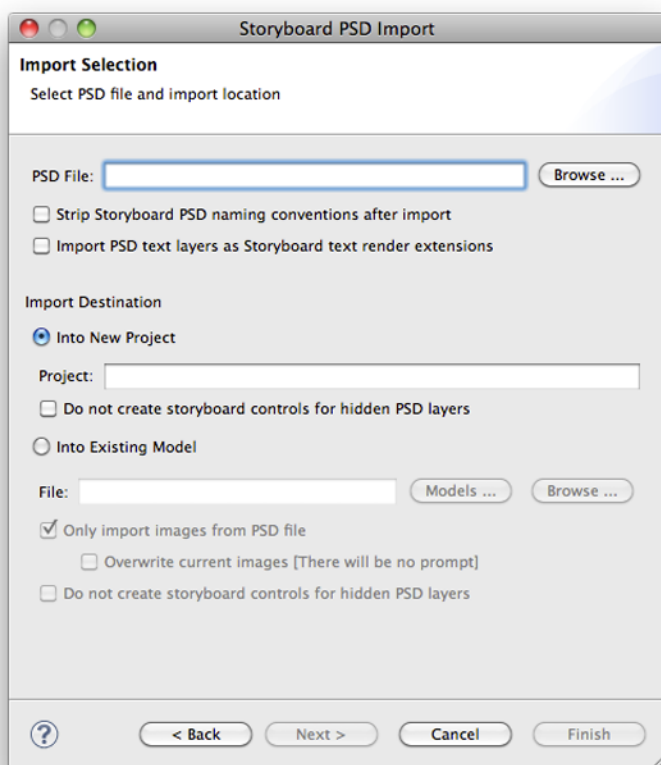
Photoshop PSD File Import

Often an excellent starting point for the application design is the graphics design that was used to do the initial preview of the application's functionality. The Photoshop™ file import provides the ability to use these design files to jumpstart the application development process.

The import wizard is initiated through the **File > Import** main menu item. This will bring up the initial import dialog.



Selecting Photoshop PSD File and clicking Next will move to the next page that allows you to select the PSD file that you want to import from the local filesystem and allow you to either create a new Storyboard project (Into New Project) or to incorporate the imported content into an existing project (Into Existing Location).



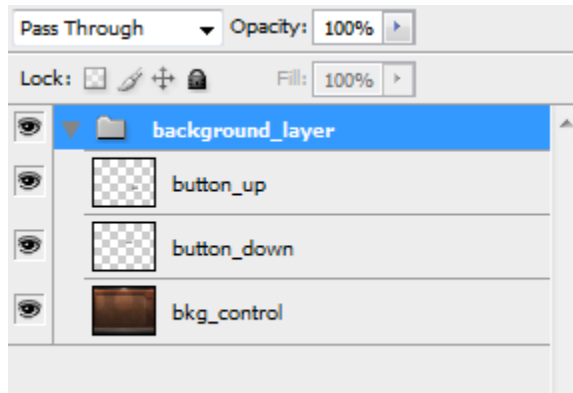
Select whether to import in to an existing project or in to a new project. You can control how the PSD import is executed by changing the PSD import options.

Once the Photoshop™ file is imported the application opens in the Storyboard editor. The layer information from the Photoshop™ file is maintained, and the layers created as part of the model are displayed in the Layers view.

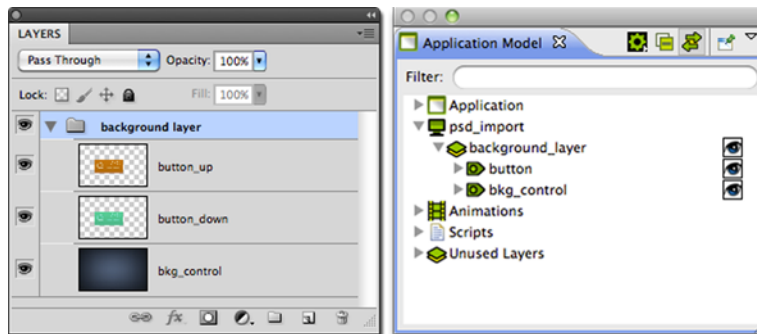
The screen with the name of the PSD file that is now a part of the application reflects the last visible state of the file when it was saved and contains those layers that were visible during that editing session.

You can also control how the importer will import elements from the PSD file to your project by following a naming convention in your PSD file. If you end the name of a PSD layer group with "_layer" this will cause all groups and layers beneath it to be added to a layer with that name. If you name a PSD layer group with a name ending with "_control" this will create a new control with all layers in the group as image render extensions. If you name a PSD layer group with a name ending with "_group" this will cause all layers and "_control" groups beneath it to be added to a group with that name. Also, if you name a PSD layer with a name that ends with "_up" and a then have a layer with the same name only ending in "_down" immediately following it, this will create a control that will act as a button. It will have an action for press and an action for release and outbound that will switch the image render extension in the control between the up and down images.

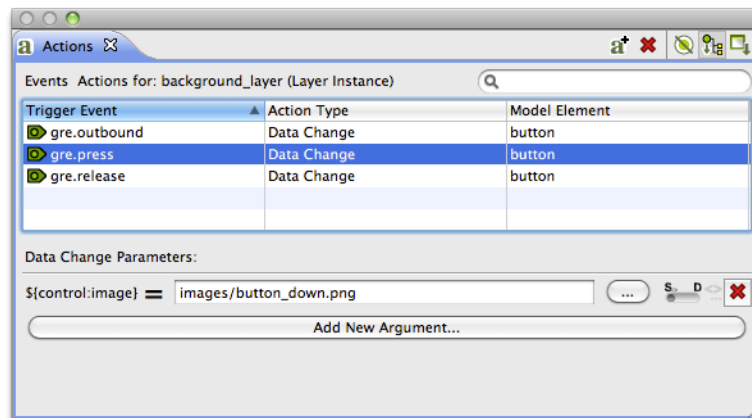
The following image demonstrates how to structure your Photoshop psd file to get a layer with a background image control and a button with actions associated with it to change the image when pressed.



Below is how the new project is laid out after the psd file has been imported into Storyboard.



Using the proper naming conventions, in Photoshop for the button images, have created actions and variables for the button control automatically.



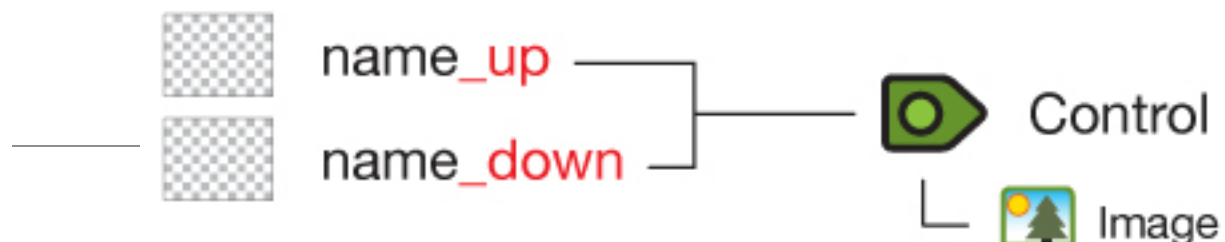
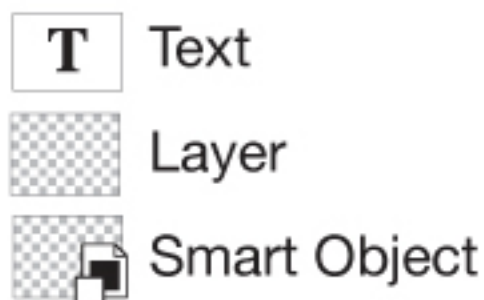
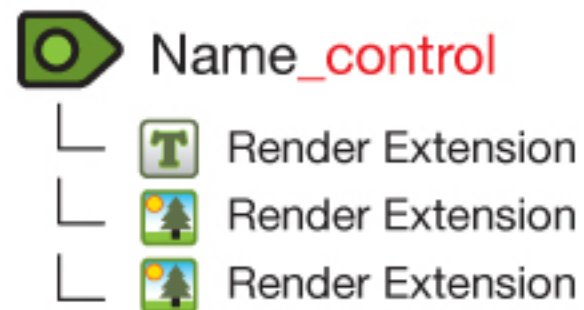
How Photoshop Content Will Import to Storyboard

By understanding how content from Photoshop will import into Storyboard gives users the ability to plan and organize a project from Photoshop. The Photoshop import feature makes the transition from the design environment to application development a single process. Users will be familiar with the structure of Storyboard application because the application model is built similarly to Photoshop's Layer Palette View. Projects in Storyboard can also receive multiple imports of Photoshop files so that a project can be added to as content becomes ready.

Photoshop



Storyboard



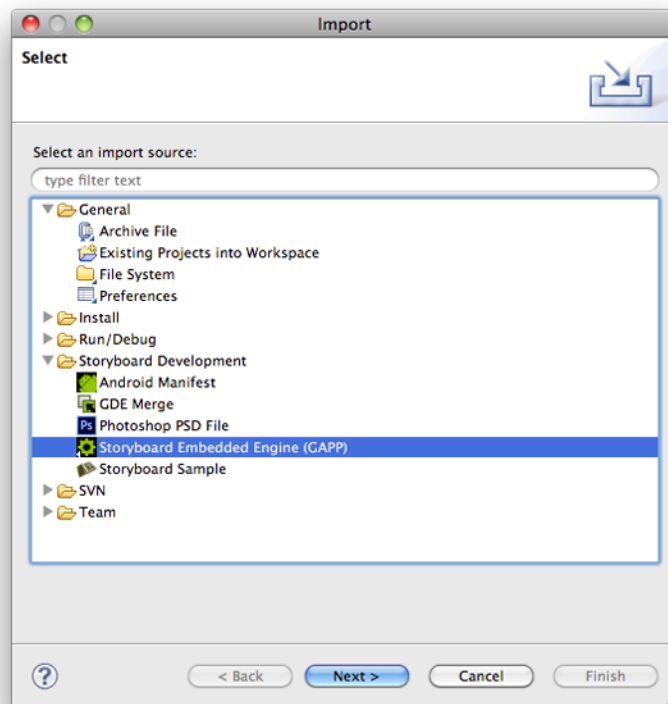
Properties and structure from Photoshop that are **not** supported in Storyboard:

- Photoshop files that are not 8-Bit and in RGB colour mode
- Layer Effects: -eg- bevels, drop shadows
- Layer Blending Modes: -eg- overlay, multiply
- Content that uses Layer Masks or Vector Masks
- Group folders that contain the same type of Group folders: -eg- a Storyboard Layer can not contain another Storyboard Layer
- Clipping Masks

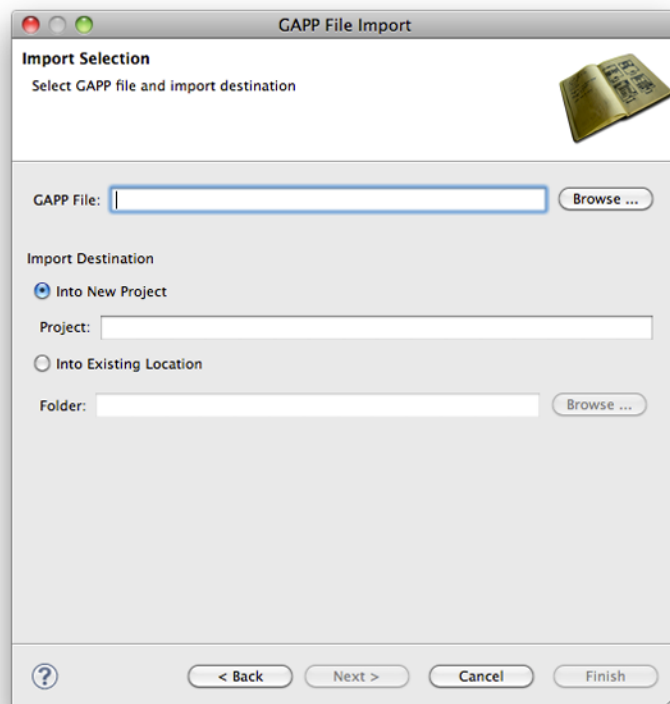
Prior to importing a photoshop file into Storyboard, content and properties that are not supported in Storyboard can be flattened, rasterized or converted to smart objects in order to maintain the same appearance between the photoshop file and application design.

Storyboard Embedded Engine Import

Storyboard Designer can also round-trip export/import the files created as part of the Storyboard Embedded Engine, also known as 'gapp files' since they typically have a .gapp file extension. These files are imported in the same way as a Photoshop™ file, using the import wizard from the **File > Import** main menu

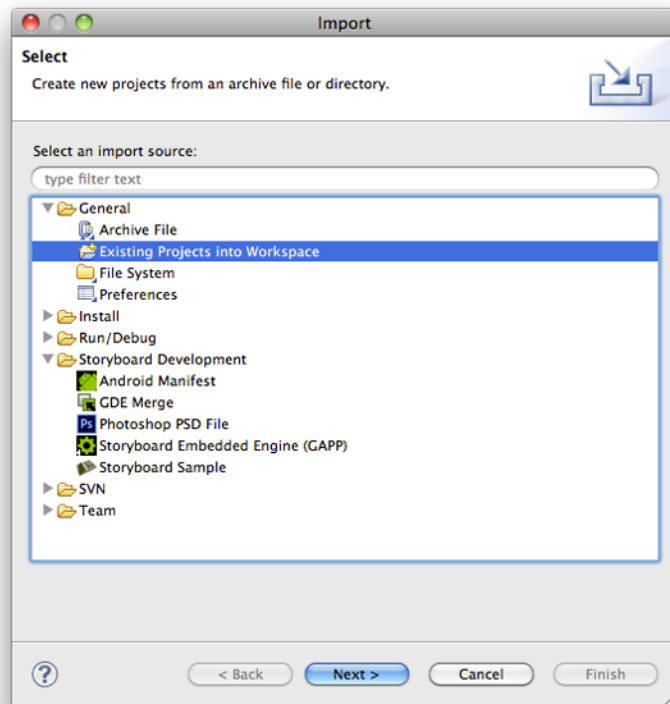


Selecting Storyboard Embedded Engine (GAPP) and Next takes you to the next page of the wizard allowing you to select the embedded engine file to be used for import.



Existing Project Import

Occasionally it may be convenient to share an existing project from one workspace to another. If the project is archived or its directory structure completely copied to a new location then it is possible to import the Storyboard Designer project as an existing project.



Selecting **File > Import > General > Existing Projects** into Workspace will launch the above import wizard. Selecting either the archive file or the project directory results in the Projects list being populated with all detected projects. Selecting one or more of the projects to import and then selecting Finish causes a new project to be created in the workspace.

Project names must be unique. If you are trying to import an existing project into the workspace, then it is required that the colliding project names be temporarily renamed so that there are no name collisions in the workspace.

Storyboard Designer Development

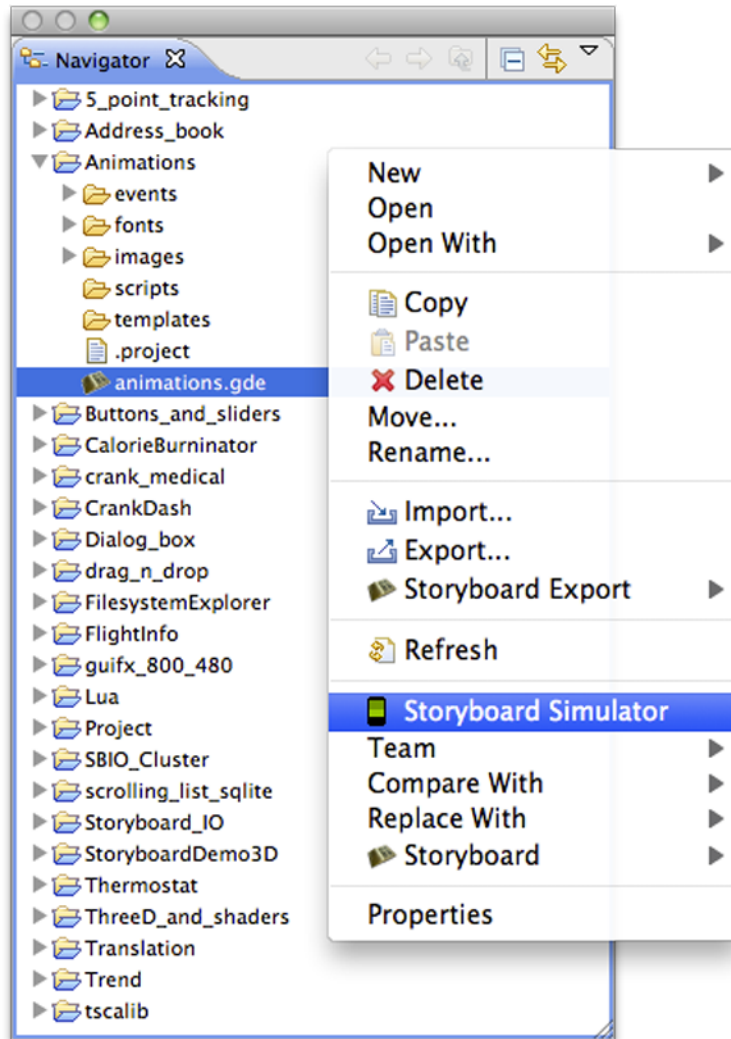
Simulating and Exporting an Application

After an application has been created, it is a good idea to run it through the simulator to validate the runtime behaviour before exporting the application to a Storyboard Embedded Engine deployment file.

The simulator is a host-based instance of the Storyboard Embedded Engine and should exhibit the same operational behaviour as the target, though there may be differences in the level of performance obtained because of the different CPU and graphic characteristics.

Simulating an Application

Simulating a Storyboard application is a straightforward process. From the Navigator view, right-click the Storyboard designer file (*.gde) and select the **Storyboard Simulator** option.



Alternatively, you can select the Storyboard Simulator option from the toolbar ...

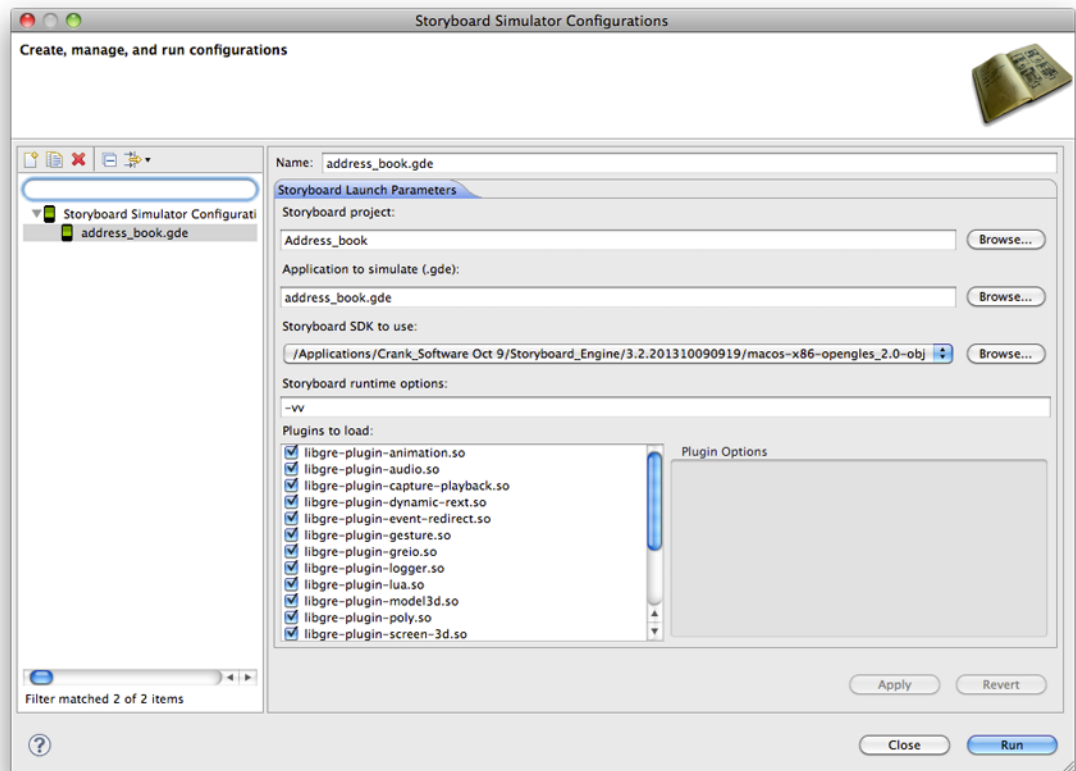


or the Storyboard Simulator Configurations from the toolbar ...

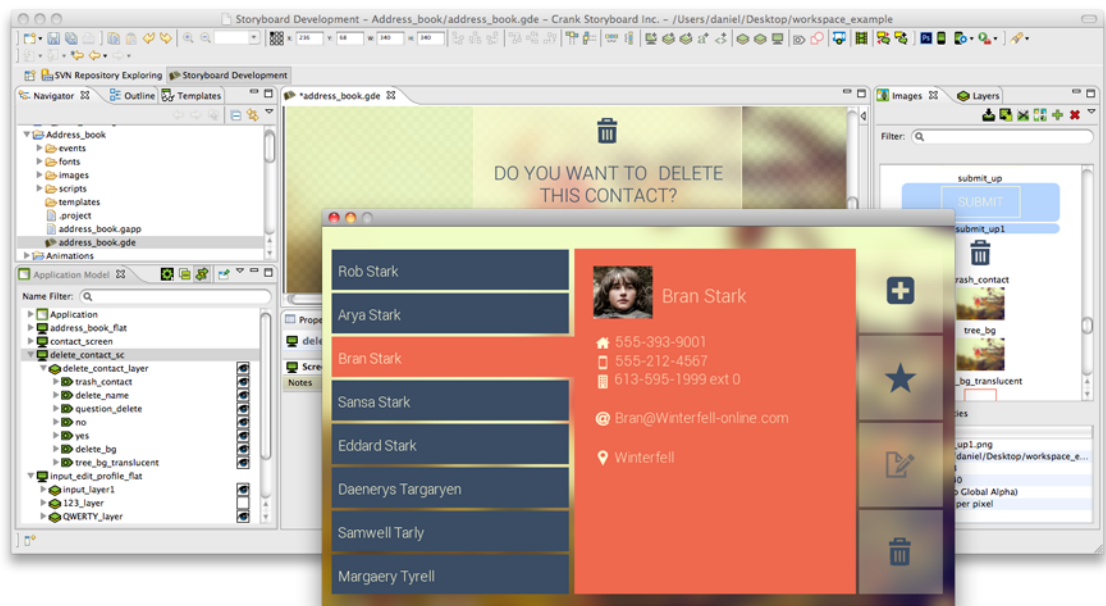


which will allow you to customize how the simulator is launched.

The options include configuring which plug-ins are to be loaded by the simulator and what level of verbosity should be used to run the simulator.

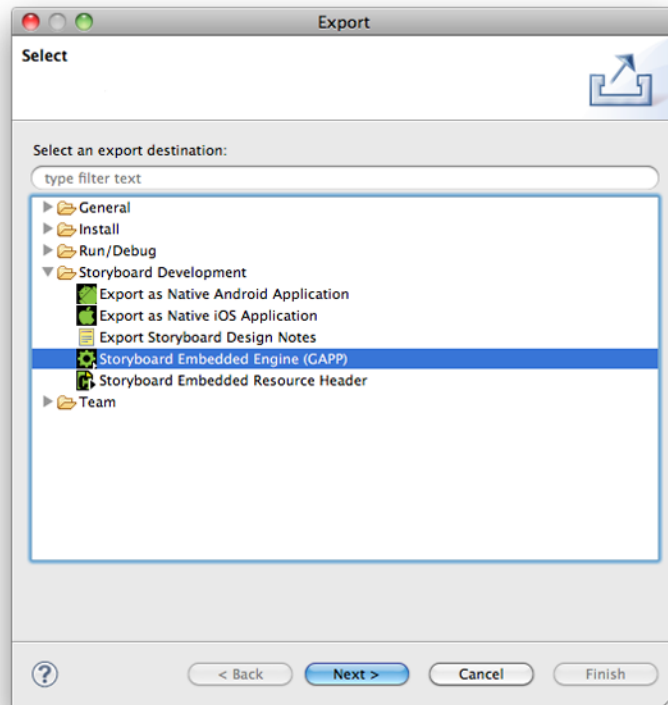


When the simulator is launched it will perform an automatic export to a Storyboard Embedded Engine to a temporary directory and run the application in its own window on the host system.

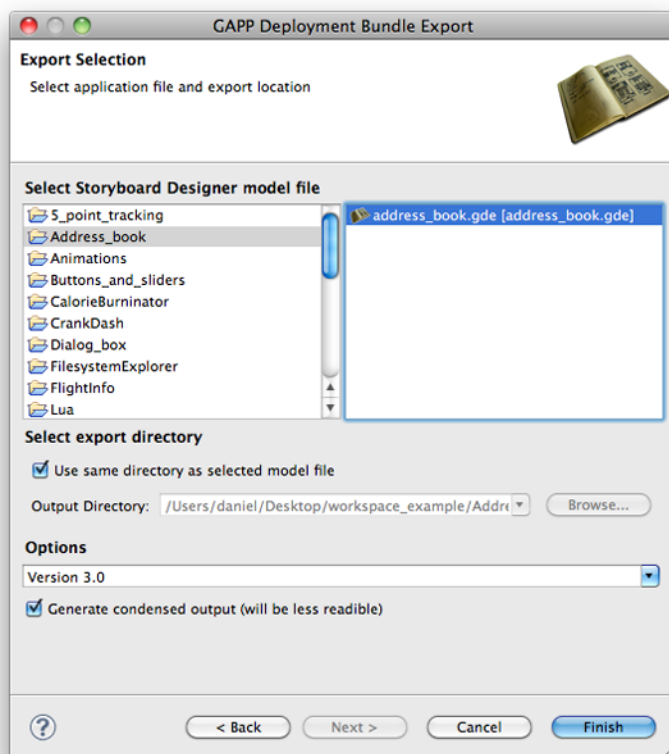


Exporting to a Storyboard Embedded Engine

Export a Storyboard Embedded Engine configuration from the main menu using **File > Export**. This launches the export wizard.



Select the Storyboard Embedded Engine (GAPP) option and click Next. Select the Storyboard application file (*.gde) that you want to export and the output location that you want to export to.



Select Finish to create the specified directory, create a Storyboard Embedded Engine (*.gapp) runtime file, and export the resources from the images, fonts, and scripts directories.

It also possible to perform headless exports of the Storyboard Design files to the Storyboard Embedded Engine files from a command line or scripting environment.

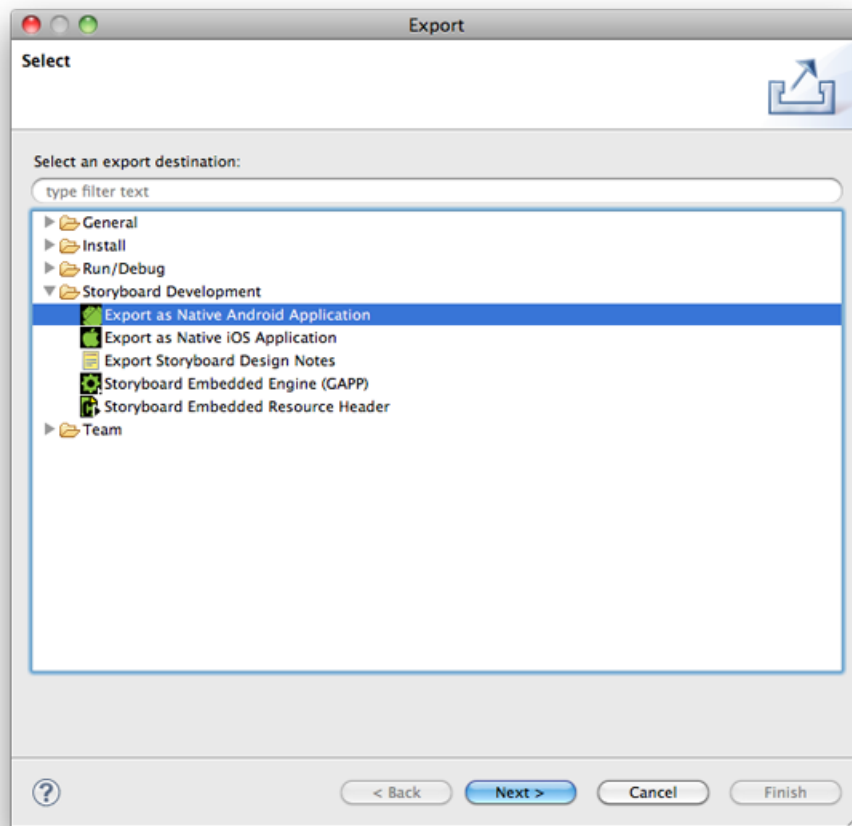
```
PATH_TO_INSTALL/Storyboard_Designer/storyboard/Storyboard      -
application                                                     com.crank.gdt.ui.gappexport
model=<PathToGDEFile[,PathToAdditionalGDEFile,...]>
output=<PathToGAPPFile>
```

Where the `model` is the full path to the Storyboard Designer model file. In the situation where multiple GDE model files are being joined together it is a comma separated list of model files where the first model file will be used for the start screen and the remaining models will be used for additional content. The `output` parameter specifies the filesystem path where the Storyboard Engine file will be created and the directory containing that file will be used to for the additional resource directories (scripts, images and fonts).

Exporting as a Native Android Application

Exporting as an Android application will create an Android package that is suitable for use on Android devices. The packages generated will only work on devices running *Android version 2.3.3 and above*.

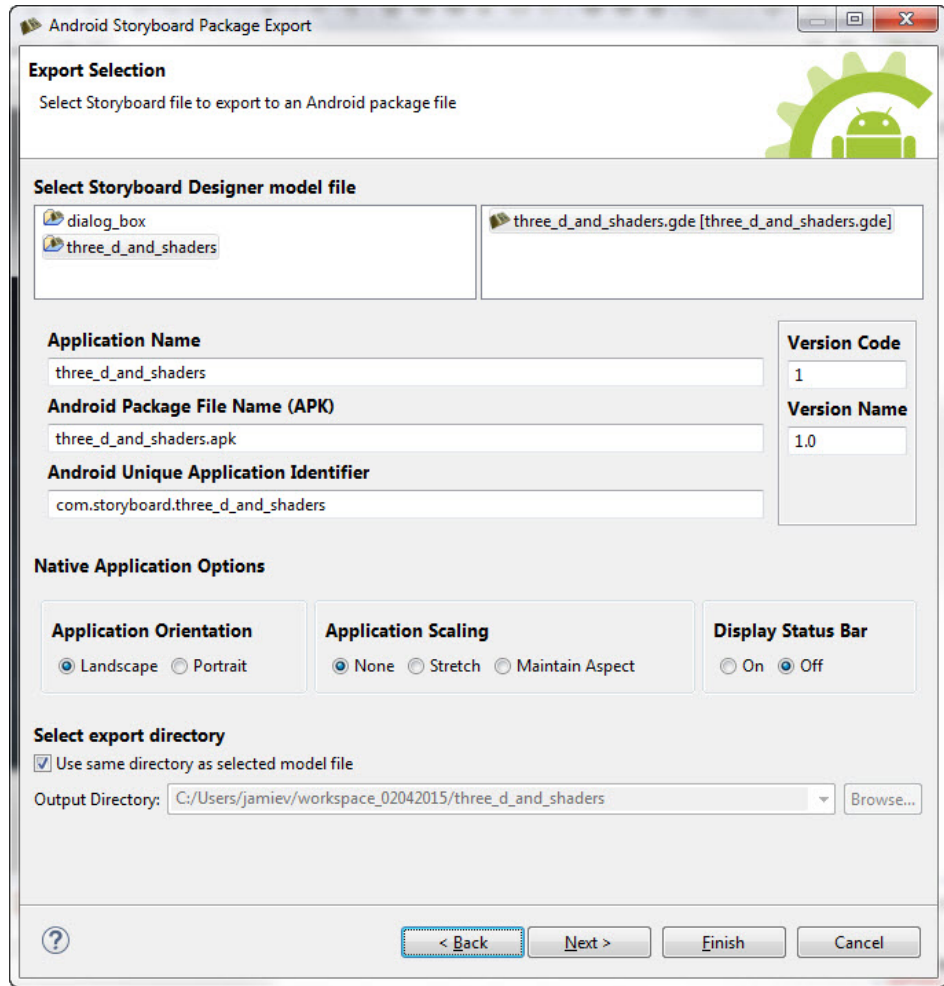
To create an Android package, select **File > Export** to start the export wizard:



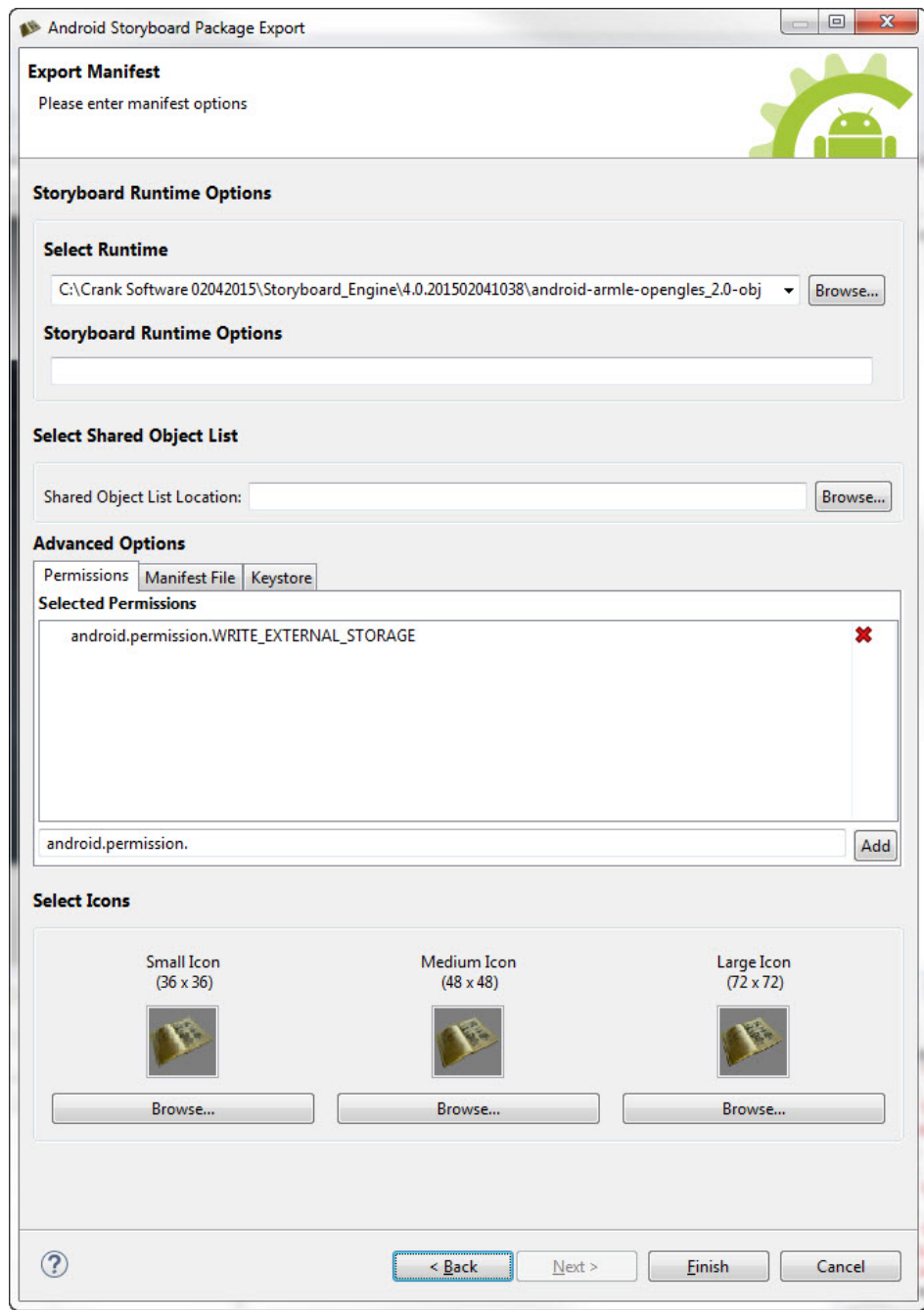
Select the Export as Native Android Application option and click Next. From the export file selection dialog, select the Storyboard application file (*.gde) that you want to export. Choose appropriate names for the application name, android package file name, and the package name. Select the directory you want to export to and options for application orientation and fullscreen. For Android devices version 4.4 and newer, the fullscreen option uses Android's sticky immersion fullscreen.

Note

Currently, without rooting an Android device, there is no way to disable the bottom bar for some Android 3.0+ devices.



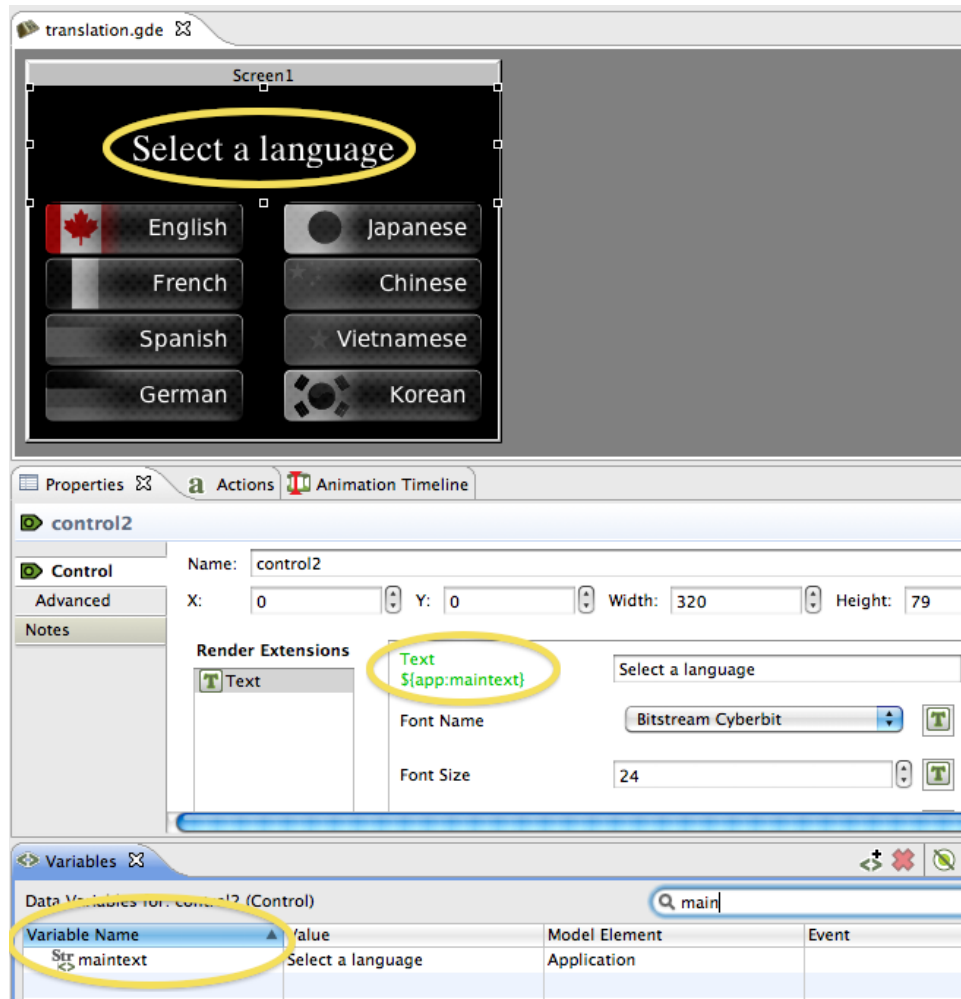
Click **Next** to bring up the a page with options for choosing a specific Storyboard Engine to use, setting Storyboard runtime options (see Storyboard Engine Options for list of available runtime options), and selecting the icons for the application. Click the tabs under Advanced Options to modify the default Permissions, Manifest file or Keystore settings (optional). If you don't need to alter the settings, just click **Finish** to use the defaults.



Select **Finish** to create the Android application package file (APK) in the directory specified. To transfer this application package to an Android device, simply copy the package onto a USB or SD card.

Translation and Internationalization

Storyboard makes it simple to translate and internationalize the text content of your application. Dynamic text content is treated the same as any other dynamic content that is rendered to the display. Within the text render extension, the translatable content should be associated with a variable. Any changes that occur to that variable, will trigger a screen re-draw to occur if that variable is being used in the current display.

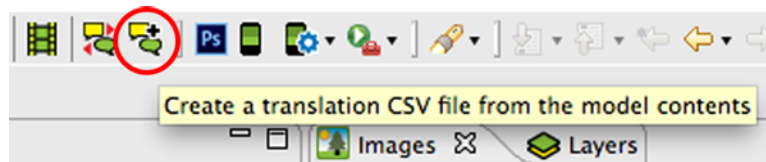


Changes in string content is automatically reflected, making the translation activity significantly less labour intensive. To apply the translated content to the application, simply update a number of data variables with the appropriate UTF-8 encoded text string.

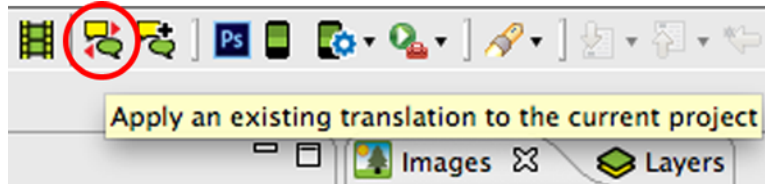
Two examples of translation are provided in the Storyboard samples. To further understand the internationalization process, be sure to check out *translation* and *thermostat* Storyboard samples available via **File > Import > Storyboard Sample**.

In some circumstances, most notably with non-latin character sets, it may also be a requirement to dynamically change the fonts being used to map to an alternative font that provides the appropriate glyph support for the characters being rendered. Additionally it may be that a change in translated text requires additional attributes to be adjusted, such as font point size or control dimensions, to accommodate the new translation. These can be adjusted as a straightforward data change to a dynamic variable. In all situations the UI will automatically refresh to show the new content.

Storyboard provides two editor functions that allow translations to be quickly prototyped in the context of the design environment.



The **Create Translation** toolbar item scans through the application, identifies all of the bound text variables that are used, and extracts them to a comma separated (csv) file that contains the Storyboard variable key and the translated text string. By default, this file is saved in the `translations` directory of the project. This file can serve as the basis for performing a dynamic load of translated content or can be used by designers to ensure that the UI is appropriate for different language configurations.



The **Apply Translation** toolbar item scans the `translations` directory of a project for suitable translation files and presents a dialog that allows the user to immediately change all of the variable definitions in a project to the values declared in the translation file. This feature allows developers a quick way to preview their content in different language configurations.

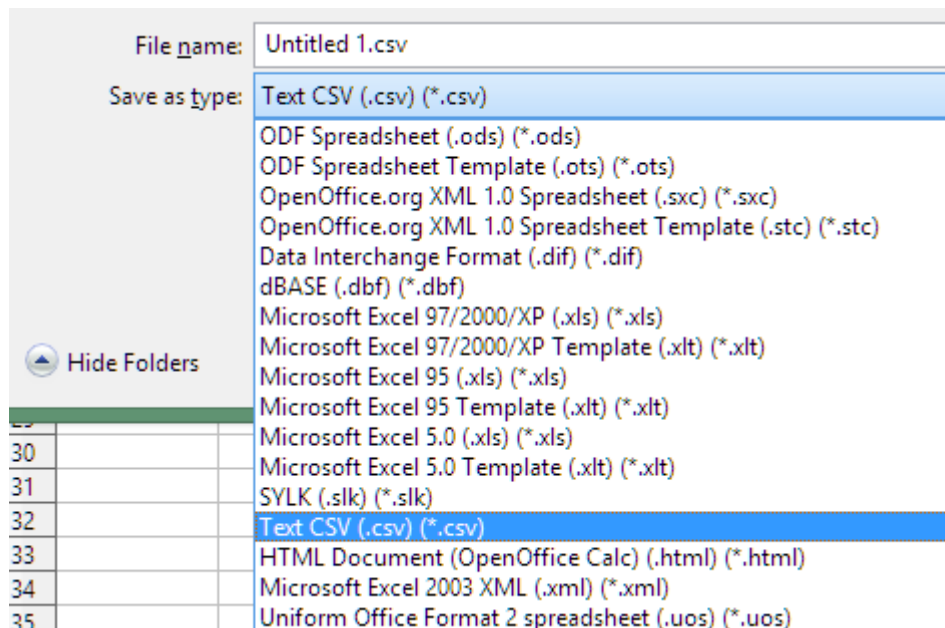
Creating and Editing Translation Content

We have explored two tools for editing and creating translation content. The recommended tool is Open Office Calc as it has the capability to character encode (csv) files in UTF-8. Some applications, such as Microsoft Excel are unable to save (csv) files using UTF-8 character encoding. We discuss two methods to create UTF-8 character encoded (csv) files. One solution is more straightforward and uses Open Office Calc and the other is a solution using Microsoft Excel and Notepad.

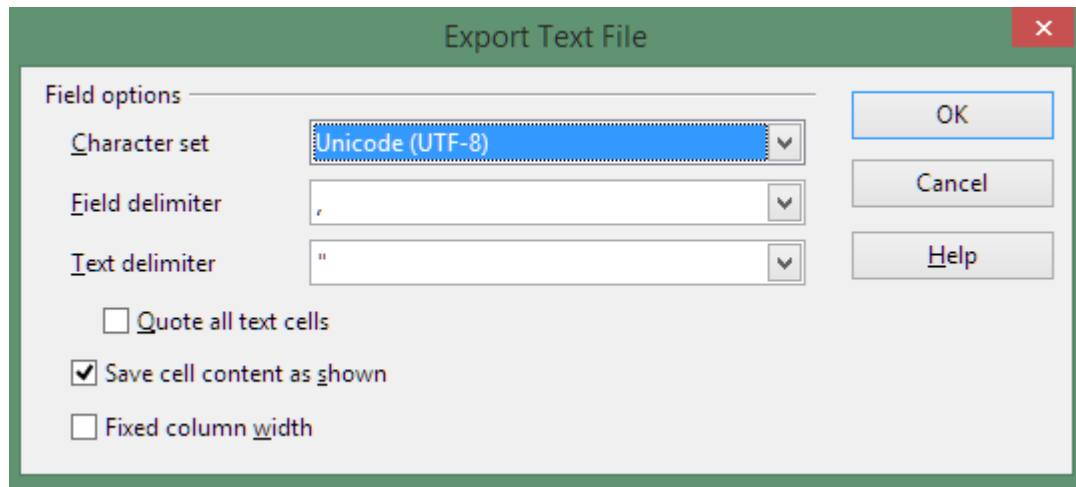
Open Office Calc

It is free and can be downloaded from www.openoffice.org [http://www.perforce.com/product/components/eclipse_plugin].

1. Using Calc, open a spreadsheet file via **File > Open...** (note: this can be a spreadsheet created using Calc or Excel) .
2. Save the file as a (csv) file via **File > Save as... > Text CSV(.csv)**



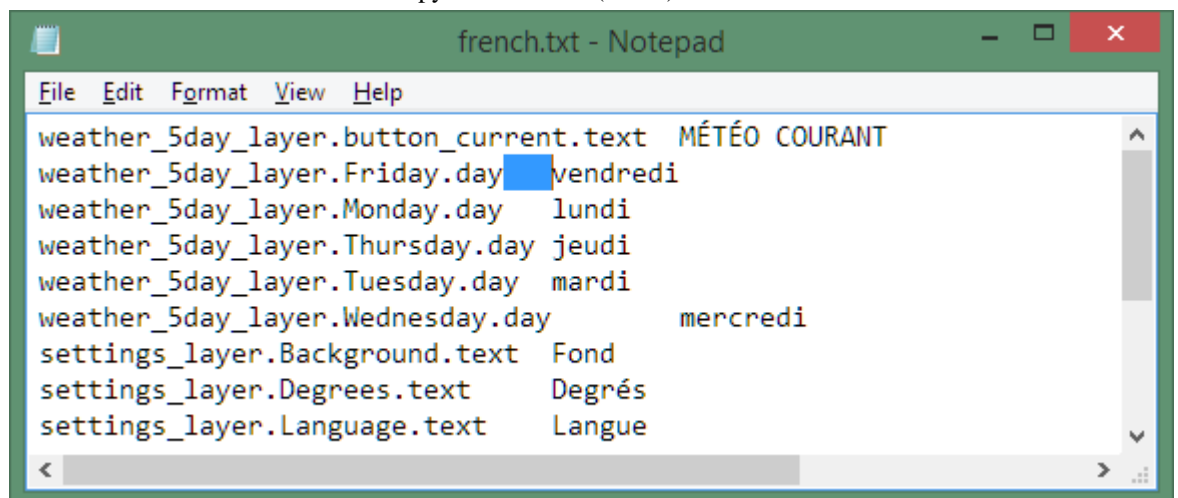
- When saving a (csv) file, Open Office Calc will ask which character encoding you wish to use for the file, be sure to choose Unicode(UTF-8).



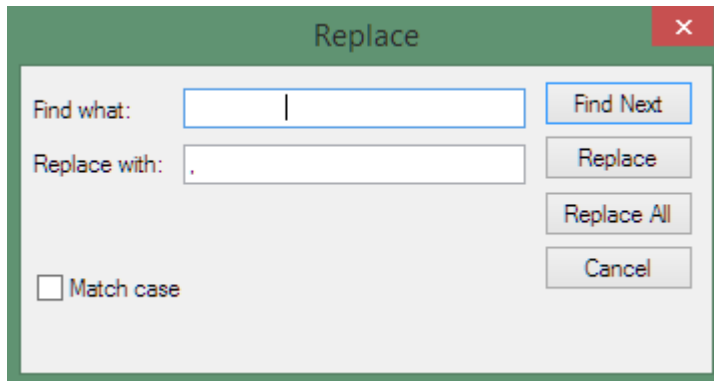
Microsoft Excel

The solution to create UTF-8 encoded (csv) files using Excel is to create a tab separated file in Excel by saving as a unicode text file then replace the tabs with commas in Notepad, save the file and change the file extension to (csv) with encoding set to UTF-8.

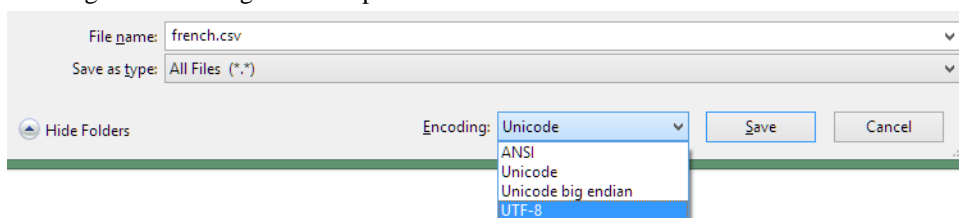
- Save the spreadsheet in unicode from within Excel **File > Save As... > Text CSV (.csv)**
- Open the .txt file from within MS Notepad
- Select one of the tab characters and copy the character (ctrl+c)



- Using the Replace tool **File > Save As... > Text CSV (.csv)** or (ctrl+h), paste the tab character in the "Find what" field (ctrl+v) and insert a comma in the "Replace with" field.



5. Save the file **File > Save As... > All Files (*.*)**, change the file extension from .txt to .csv and be sure to change the encoding in the drop down menu to UTF-8.



6. Now the resulting (csv) file is UTF-8 character coded and can be opened from Excel to ensure that the data is as correct. Do not save the file from Excel or the UTF-8 encoding will be lost.

More information about the Microsoft Excel solution can be found at https://help.salesforce.com/apex/HTViewSolution?id=000003837&language=en_US [http://www.perforce.com/product/components/eclipse_plugin].

OpenGL ES 2.0 Custom Shader, 3D Model and Compressed Texture Support

Storyboard provides the ability to leverage 3D capable OpenGL ES 2.0 hardware to render and display 3D objects and also create custom shaders to modify and adjust the display of controls.

Note

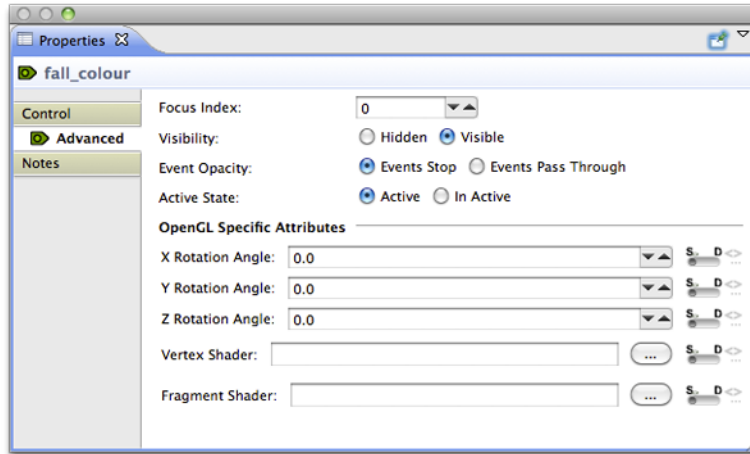
This functionality is only available for OpenGLES rendering systems.

Custom Shader

A complete description of OpenGL ES 2.0 shaders is beyond the scope of this document and is better treated in detail in the OpenGL ES Shader Language [http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf] specification. This document will focus on the shader programming aspects that are relevant to Storyboard developers.

Note

At this time, shader effects are not visible within the tool and are only visible when using an OpenGL based runtime or simulator.



Storyboard provides the ability to hook in a fragment shader and/or a vertex shader to be associated with the rendering of a control element. The shaders are provided as files and are associated with the control through the **Advanced** tab of the control's properties. Here you can select either a specific file for either shader or bind the shaders to variables that will resolve to files in the project at runtime.

The shaders will be applied to the resulting texture that is the outcome of having all of the current render extensions applied to it. That is to say that the control is effectively rendered as it would be in the normal sense, but the final result is provided as a texture to the shaders to manipulate before it is finally rendered to the display.

Shader programs have three types of variables: attributes, varying, and uniforms. Attributes are passed into the shader from the render manager and contain data such as vertex locations, and texture coordinates. Varying variables are calculated in the vertex shader and passed into the fragment shader after being interpolated based upon the location of the fragment. Uniforms are also passed in from the render manager, but are typically used for purposes other than storing the geometry being rendered, for instance, containing a global alpha value which can be used to blend an entire model.

When writing a shader program, it is important to follow the conventions for attributes and uniforms established by the render manager. Otherwise, it will not be able to pass in geometric data and nothing will be rendered. Below is a minimal vertex shader which matches the functionality of the built in vertex shader for images.

```
attribute vec4 myVertex;
attribute vec4 myUV;

varying vec2 vtex;

uniform mat4 projMatrix;
uniform mat4 mvMatrix;

void main(void)
{
    gl_Position = projMatrix * mvMatrix * myVertex;
    vtex = myUV.st;
}
```

The myVertex and myUV attributes contain vertex and texture coordinates respectively. The projMatrix and mvMatrix contain the projection and modelview matrices, which are used to transform the input vertex

position, which is then assigned to the `gl_Position` for the vertex. The varying `vtex` is used to hold the interpolated texture coordinate which is then passed to the fragment shader. The render manager looks up `myVertex`, `myUV`, `projMatrix` and `mvMatrix` by name when it loads the shader, so these names must be used in any custom vertex shader. The varying name must match between the vertex shader and the fragment shader.

Below is a fragment shader which matches the functionality of the built in fragment shader for images, and shows the minimal code required to work with the Storyboard render manager.

```
#ifdef GL_ES
precision mediump float;
#endif
uniform sampler2D sampler2d;
varying vec2 vtex;

void main (void)
{
    gl_FragColor = texture2D(sampler2d, vtex);
}
```

The initial precision declaration is required by OpenGL ES, but not supported by OpenGL, and is set with a preprocessor conditional. The `sampler2D` uniform controls which texture unit is used when sampling a texture. The render manager only supports a single texture. The varying variable is interpolated based upon the vertex values in the vertex shader, and is passed into the sampler to look up the color at the fragment location. This is assigned to `gl_FragColor` and becomes the fragment color.

It is also possible to pass data from Storyboard variables to shader uniform variables, subject to two constraints: the model element for the variable must be the control for which the custom shaders are being used and the variable type must be float. When the custom shader is loaded, a list of all of the uniforms present is created. The name of each uniform is then compared to the list of variables attached to the control, and if a matching name of the appropriate type is found, it is used to set the value of the uniform when the control is rendered.

As an example, consider animating a custom shader to do a simple fade-in based upon the value of a timer. First, create a variable of type float called "current_time" for the control with the custom shader. Then create an animation using the animation timeline which changes the value of the variable from 0.0 to 1.0 over a few seconds, and create an appropriate trigger for the animation, for instance a mouse press event. Then, edit your fragment shader as follows:

```
#ifdef GL_ES
precision mediump float;
#endif

uniform float current_time;
uniform sampler2D sampler2d;
varying vec2 vtex;

void main (void)
{
    gl_FragColor = texture2D(sampler, vtex) * current_time;
}
```

When the control is rendered, the value of the uniform `current_time` will be set from the value of the control variable `current_time`, which will cause the color read from the texture to be scaled from 0.0 to 1.0 over the duration of the animation, causing a fade in effect.

Compressed Textures

The OpenGL ES 2.0 render manager now supports compressed textures on supported hardware. The formats supported are PVRTC1, both 4BPP and 2BPP. This compression format is supported on most PowerVR graphics chipsets.

To determine if the chipset supports it, running Storyboard with a verbosity level of 6 (`-vvvvvv_` will print out, on startup, the GLES extensions supported by the chip. If PVRTC is supported, you will see `GL_IMG_texture_compression_pvrtc` in the extension string list.

Storyboard will manually decode these images if the runtime being used does not support them. Should a project that was running on a PowerVR chip and using compressed images be run on a SW runtime that does not support them, the images would still decode and render correctly, just without HW acceleration.

PVRTC provides a 8x improvement in memory size (A 1024x1024x4 BMP would take 4MB of memory, whereas a PVRTC image would take 512K)

Compression tools can be found at:

- PVRTexTool [<http://community.imgtec.com/developers/powervr/tools/pvrtextool/>]
- Using texturetool to Compress Textures [https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TextureTool/TextureTool.html]

Working With Templates

Storyboard Designer allows developers to create re-usable design components, templates, that can be shared among multiple projects.

A template is created by selecting a Storyboard Designer model element in either the Storyboard editor or the Application view and right clicking and selecting **Create New Template**. A dialog will prompt for a template name, description, and the name of the file to save the template.

Templates may be composed of a selection of one or more controls or one or more groups. If several controls considered together provide a logical functionality, then it is generally a better idea to encapsulate them within a group and create a template from that group rather than template the multiple controls together. Templating a group more easily allows the functionality to be encapsulated so that the template can be re-used several times within an application. When templating several controls, it is possible to template content originating on different layers. In this situation the position used for the control is going to be the position relative to it's parent layer and not the position of the control relative to the other controls in that particular screen context where the parent layer offset may be adjusted or different from screen to screen.

With any content being templated, the template generator will scan the selected model objects (controls and groups) and any child model objects and include over any variables, actions and render extensions that are directly referenced. This scanning will also attempt to identify external resources such as images, fonts

and script files that are used by the model object being templated. This will also include any referenced animations associated with animation actions that the templated content triggers. In the situation where a Lua script is referenced, the entire Lua file will be included so for this reason it is a good idea to keep the functionality used by the content to be templated isolated into its own file.

When the template is applied if any of the templated resources (images, fonts, scripts etc) don't exist in the target project then they will be transferred from the template to the target project. If the resources already exist, then it is assumed that these resources should be used in place and the existing resources are not modified.

Templates are stored in an external file independent of the project where they were created so that they can be shared and incorporated into other projects by adding them into the target project's templates directory. Once they have been added to that directory they will be parsed by the project and will be displayed in the Templates view and be made available for use. By default new templates are saved in that project's *templates* directory.

Working with Multiple Application Design Files

Sometimes in a large project it is desirable to split the application design work not only among multiple application designers, but also among multiple design files so as to minimize the amount of conflict that needs to be managed by a revision control system such as subversion, git or mercurial.

In order to facilitate working with multiple files, Storyboard maintains each design as a stand-alone free runnable application, even when it may be later incorporated into a larger unified application. This separation allows a more rapid development cycle as application developers are simulating and tuning content in a more focused environment rather than having to consider all other system functionality.

To create a unified application, the Storyboard simulators and exporters have been modified to allow them to accept multiple application files which they will merge together to produce a single unified output for the Storyboard Engine.

Getting Started

Starting a multi-file application development is the same as starting a regular project since before you can have two projects, you need to have one project. Typically the initial project created will be the master application and will host the majority of the shared resource content. This is only a convention and not a technical requirement.

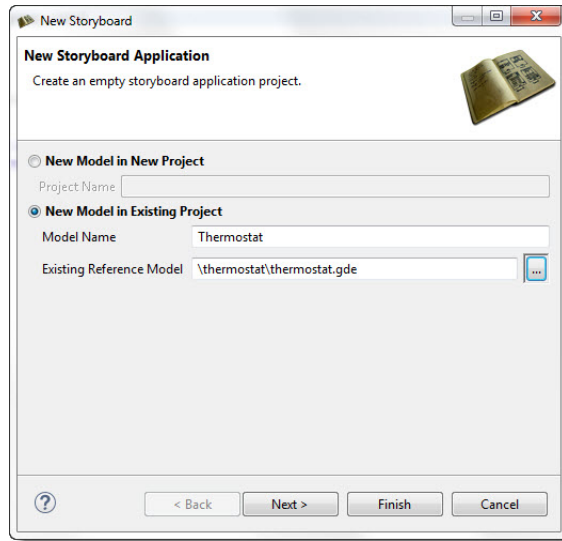
Once another application is required, for example to represent a distinctly themed area of the unified application, then you will want to create another Designer application model file within the base project.

Note

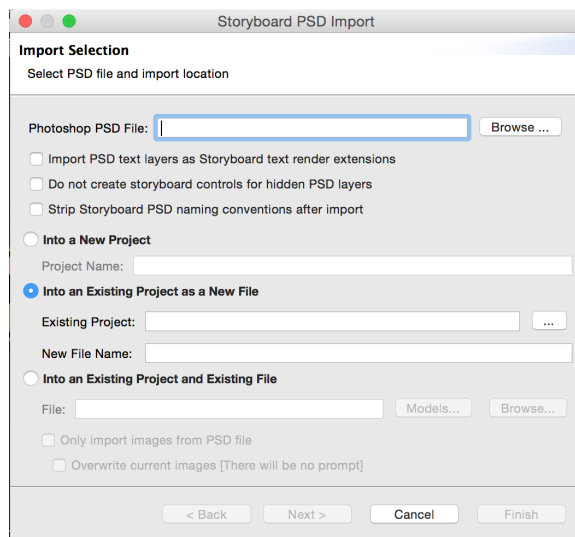
Currently multi-file applications are all hosted in the same project so that they can ensure proper sharing of image, font, and script resources.

You can create another application file in several ways:

Create a blank application using: File > New > Storyboard Application ... and select "New Model in Existing Project"



Import PSD content into a application: File > Import > PSD ... and select "Into an Existing Project as a new File"



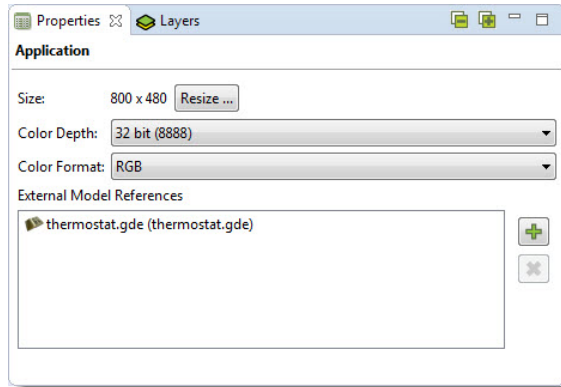
You can also simply copy and paste an existing application and rename the *.gde file to bootstrap a new design.

Doing this will result in multiple application model (*.gde) files that are all contained within the same project. They all reference the same image and script resources.

Integrating Application Content

Screen transitions provide the means to tie together multiple applications to form a single, larger, unified application.

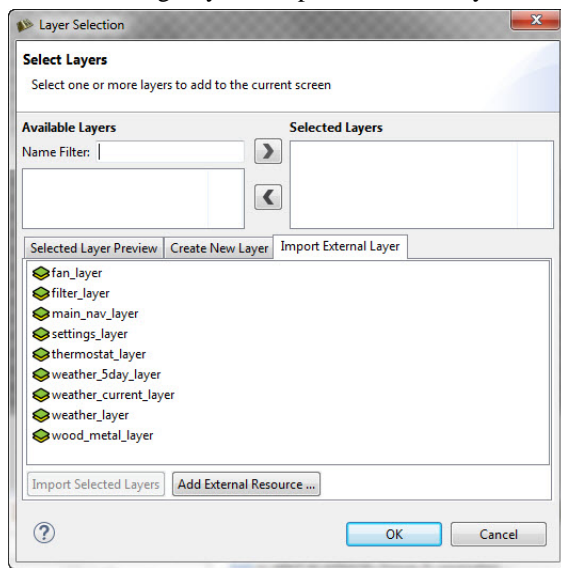
In order to indicate that one application (the source application) will be making a reference to screens in another application (external reference), the source application should list the external application in its Properties. Go to Application > Properties > External Model References



Once an application is added as an external reference, the screens from that application will show up in any of the Screen transition selection lists as well as any animation definitions..

Layers from any listed external applications can be added to an application by explicitly importing them using:

Add > Existing Layer > Import External Layer

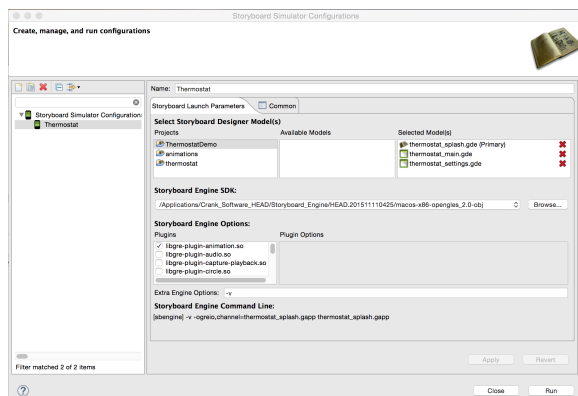


Once a layer is added as an external layer, then a copy of the original layer from the external application is snapshotted and incorporated directly into the source application. At this point, the external layer can be used with any source application screens just like any other layers.

Simulating and Exporting Multiple Model Files

Once applications have been developed, shared content leveraged, and transitions between application screens established the next step is to simulate or export the unified application.

The Storyboard simulation configuration dialog, accessed via Run > Storyboard Simulator Configurations allows multiple model files to be specified. All models in the 'Selected Models' block will be included in the application when launched.



The first source application in the list (tagged as Primary) will be used to determine the unified application's launch screen.

Similarly exporting a runtime application to be used with Storyboard Engine, Android, or iOS targets also allows multiple Storyboard application files to be selected. In these export scenarios, you will be prompted to select the source application that should be used for the unified application's launch screen.

If there are no conflicts among the selected applications they will be merged together and converted into a single unified application and used for the user-selected operation of simulation or export.

If there are conflicts among the resources then the differences will need to be resolved before continuing with an application merge.

Resolving Conflicts and Synchronizing Changes

When multiple applications are merged together to form a unified application, the following occurs:

- All layers from all applications are assembled together into a unified list of available layers. If two or more layer names are the same then those layers have their content compared. If the content is identical then the merge continues. If the layer content differs, then an error is flagged and the user will be prompted to resolve the differences and the application merge stops.
- All application/global level variables from all applications are assembled together into a unified list of global variables. If two or more variable names are the same then those variables have their values compared. If the values are identical then the merge continues. If the variable values differ, then an error is flagged and the user will be prompted to resolve the differences and the application merge stops.
- All animation definitions from all applications are assembled together into a unified list of available animations. If two or more animation names are the same then those animations have their definitions compared. If the animation definitions are identical, the merge continues. If the animation definitions differ, then an error is flagged and the user will be prompted to resolve the differences and the application merge stops.
- All screens from all applications are assembled together into a unified list of available screens. If two or more screens have the same name, then an error is immediately flagged and the user is prompted to resolve the differences.

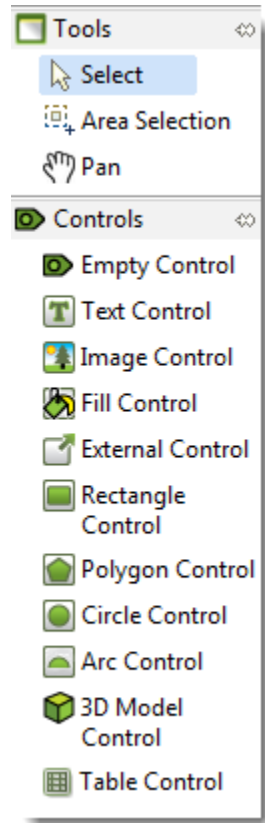
The external referencing of model elements relies on names remaining consistent during the application development. In some instances, if names change it may be possible for content to become unsynchronized and it may need to be resynchronized on an application by application basis.

The application properties page provides a synchronization action that scans the project for externally referenced content and then compares that content to the source reference. If there is a difference, then the

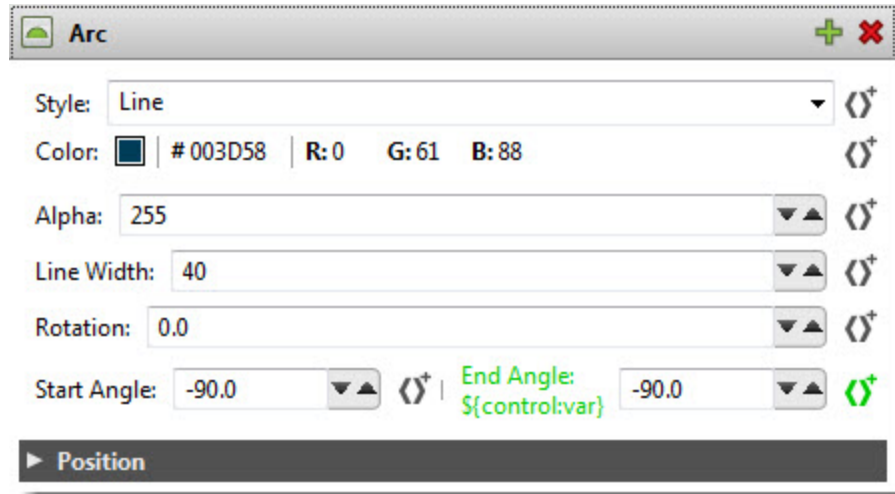
difference is flagged as a conflict for resolution and the user is prompted for different ways to solve the conflict based on the nature of the issue

Circles and Arcs

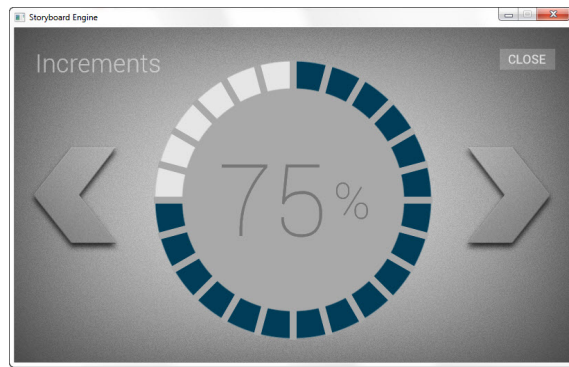
Using the circle and arc controls are an easy way to quickly add spherical objects to your Storyboard application without having to create them from scratch.



It is easy to manipulate the circle or arc controls by adding variables to their various properties. You can then dynamically change those values through Lua script or external sources. (thirdparty applications or device drivers)

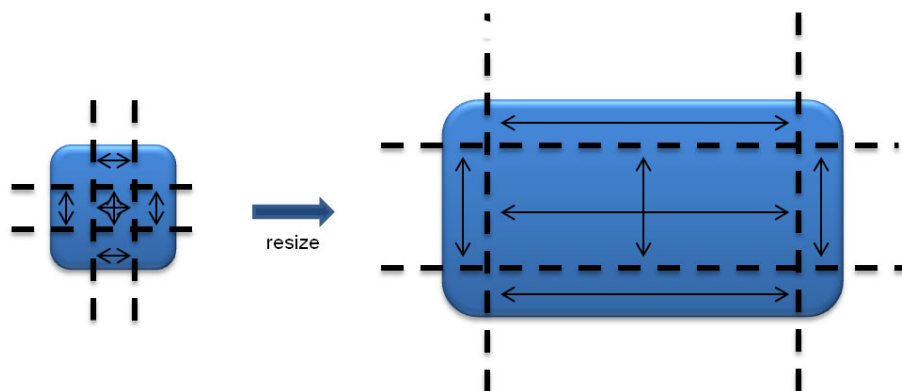


Our circles_sample pictured below is located in the Samples directory of your installation and demonstrates ways of using our arc control.



9-Patch

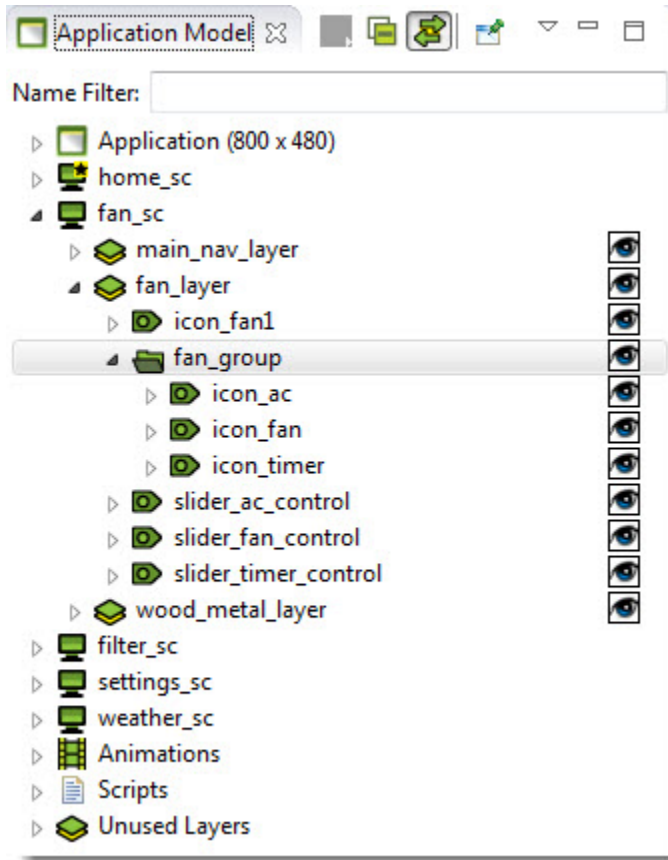
9-Patch is technique used to scale an image in such a way that the 4 corners remain unscaled. The four edges are scaled in one axis and the middle is scaled in both axis. 9-Patch support has been added to Storyboard Designer to make scaling images on embedded applications easier. Instead of having multiple button images of various sizes, customers can now have one image that scales and maintains image quality.



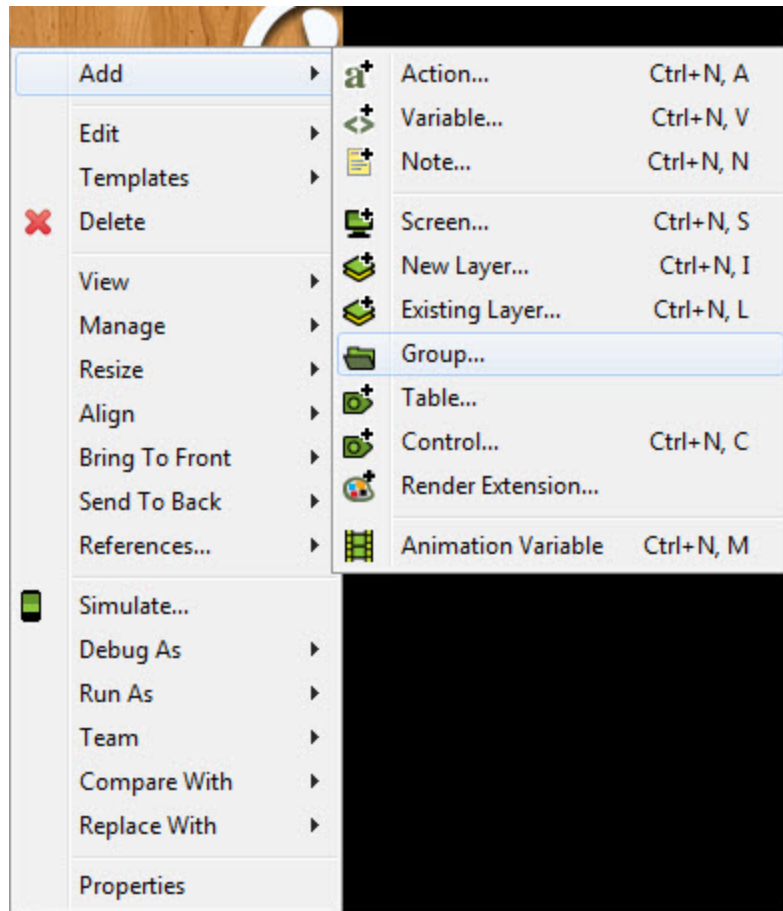
9-Patch images can be designed and edited directly in the Storyboard Design environment. You can quickly analyze and convert existing large or scaled image content to 9-Patch format to achieve immediate memory and runtime performance improvements.

Groups

Groups have been added to alleviate the situation of placing many render extensions in the same control. Customers now have the ability to group individual controls together. Groups can contain actions and can be manipulated using their x, y, and hidden internal variables. More information about Group data variables can be found in the "Group, Control and Layer Data Variables" section.



Adding groups to a Storyboard application is very straightforward. You can group existing controls together by first selecting the ones you want in the Main Editor, then right-clicking and selecting Add -> Group. This allows you to name a group and adds all the highlighted controls into that group.

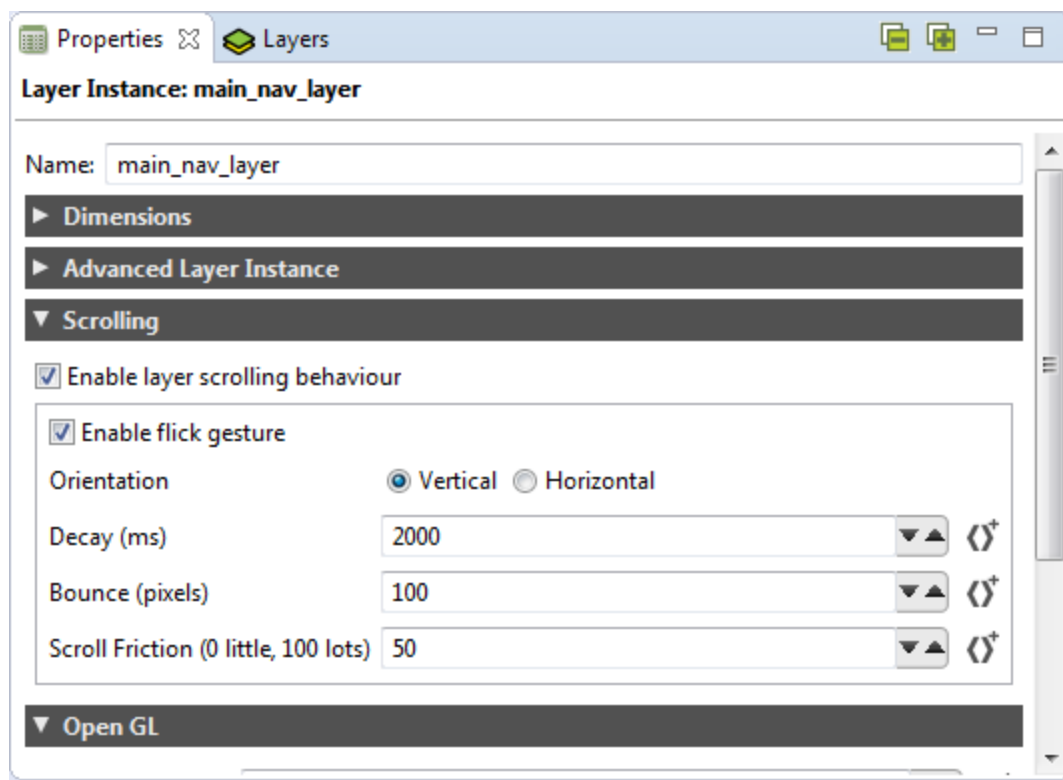


You can also add a group to your Storyboard application and then decide later on what controls you want in that group. Simply right-click in the Main Editor, and select Add -> Group. After the group has been created you can drag the needed controls into the group using the Application Model view.

Groups are highlighted upon selection in the Application Model or if they are ALT-selected in the editor. The selection displays the outline of the group contents and allows for resizing. Groups are also shown in the wireframe and outline mode.

Scrolling Layers

Scrolling layers provide the ability to scroll multiple controls on a layer in an easy fashion similar to a scrolling table. Select the “Enable layer scrolling behaviour” check box in the Layer’s properties view to enable scrolling on a layer.



A scrolling layer behaves as a viewport to the controls on the layer. This means the size of the layer should be the size of the intended viewport. By scrolling the layer, Controls that are placed outside of the viewport will be brought into view.

When scrolling, the layer itself is not moving, the controls within the layer are. To measure this, the variables `grd_xoffset` and `grd_yoffset` may be used to determine how far the scrolling layer has scrolled away from its origin.

Orientation: Vertical or Horizontal scrolling

Decay: Maximum time in milliseconds the scrolling layer will scroll after a flick.

Bounce: Number of pixels for the scrolling layer to bounce when reaching the edge of the contents within the layer.

Scroll Friction: A number from 0 to 100 to determine the friction level of the scrolling layer. With higher friction it becomes more resistant to moving when swiped and more inclined to slowing down when released.

Target Configuration

iOS Devices

When exporting to an iOS Device all related files and available plugins are packaged into a single application for the device. Since this is the case there is no need to set up environment variables or specify runtime options on the target, since this will all be done in storyboard designer when you export the application.

To configure an iOS device to run your apps you need to set up a few things from Apple first:

1. Xcode
2. iOS developer account
3. A code signing certificate
4. The device ID of the apple products you will be running on
5. The identifiers of the applications you will be making
6. A provisioning profile

Xcode

Xcode no longer includes the command line tools and need to be downloaded separately from the app store.

iOS Developer Account

To obtain a developer account you will need to go to <https://developer.apple.com>, click on iOS Dev Center, then click register. After that follow their step by step instructions and you will be a registered iOS Developer.

Code Signing Certificate

Log into your iOS Development account and click on the iOS Provisioning Portal link at the right hand side of the page. In your Provisioning Portal, click "Certificates" in the left tab bar. If you do not have a certificate, there will be an option to submit one. To do this you will need to make a Signing request. You can do this by launching Keychain Access, located at /Applications/Utilities. Then go to Keychain Access > Certificate Assistant > Request a Certificate From a Certificate Authority. Enter your email address and your name, then check "save to disk". Once that has been generated, go back into the certificates page of your provisioning profile and submit it for approval. After the certificate gets approved, download the file and open it, and it will be added to you keychain access.

Device IDs

Log into your iOS Development account and click on the iOS Provisioning Portal link at the right hand side of the page. In your Provisioning Portal, click "Devices" in the left tab bar. Click the add devices button at the top right of the page. Here you will need the device's name and Device ID. To get these connect the device to your computer and find it in iTunes. With the device selected click on the device's serial number, and it will switch to the Device's Identifier. Then click Edit > Copy Identifier (UDID). Now return to your Provisioning Portal, paste the identifier and enter the name of the device.

Application IDs

Log into your iOS Development account and click on the iOS Provisioning Portal link at the right hand side of the page. In your Provisioning Portal, click "App IDs" in the left tab bar. For this section we recommend you set up a generic App ID and have it accept all of you applications. However if you wish to enable other iOS like Push Notifications or In-App Purchase, you will need to make an ID for that individual app. To create a generic App ID, click the Add New App ID Button. Now enter a description of the app this Id will match with, ex "Application Development ID". Now enter the Bundle identifier. If this is a generic App ID simply type "*". If this is for a specific app, enter the app's identifier. ex "com.cranksoftware.storyboardApp". Now click submit and you can go back to the previous page, find this App ID and configure all of the options this specific application needs.

Provisioning Profile

Log into your iOS Development account and click on the iOS Provisioning Portal link at the right hand side of the page. In your Provisioning Portal, click "Provisioning" in the left tab bar. Click the New Profile

button. Create a name for this profile. Select the certificates that will be used by this profile. Select the App ID that will be used by this profile. Select the Devices that will be used by this profile. Once this is completed, download the `YourProvisionProfileName.mobileprovision` file and save it to your computer. When you are exporting your Storyboard application, you will need to tell storyboard where this file is.

User Defined Actions

New actions can be added to the Storyboard Engine through standard programming extension points. In order to make those new actions available within the Storyboard Designer development environment it is necessary to describe the name and type of the action arguments in a template so that they can be properly presented within the Storyboard Designer user interface. This can be done on a project by project basis using an action template file.

An action template is an XML file with the following formatted content:

```
<actiontemplates>
<template name="NAME">
  <arguments>
    <element name="ARG_NAME" type="ARG_TYPE" />
    ... as many elements as there are arguments ...
  </arguments>
</template>
... as many templates as there are actions ...
</actiontemplates>
```

The user defined fields are as follows:

NAME This is the name of the action as it appears in the Storyboard Engine runtime (gapp) file.

ARG_NAME This is the name of an argument option as it appears in the Storyboard Engine runtime (gapp) file.

ARG_TYPE This is the type of the argument and can be one of the following:

string	A text string value
integer	A numeric value with an optional range specified by 'min' and 'max' attributes
float	A floating point numeric value with an options range specified by a 'min' and 'max' attributes
boolean	A boolean true/false value

In order to be automatically included in a Storyboard Designer project, the action template file should be placed in the `templates` directory of the project where it is to be used. The name of the template file can be anything valid for the file system, but it should contain the file extension `.sbat` in order to identify it as an action template file.

User Defined Render Extensions

New render extensions can be added to the Storyboard Engine through standard programming extension points. In order to make those new render extensions available within the Storyboard Designer

development environment it is necessary to describe the name and type of the render extension arguments in a template so that they can be properly presented within the Storyboard Designer user interface. This can be done on a project by project basis using an render extension template file.

A render extension template is an XML file with the following formatted content:

```
<rendertemplates>
<template name="NAME">
  <arguments>
    <element name="ARG_NAME" type="ARG_TYPE" />
    ... as many elements as there are arguments ...
  </arguments>
</template>
... as many templates as there are actions ...
</rendertemplates>
```

The user defined fields are as follows:

NAME This is the name of the render extension as it appears in the Storyboard Engine runtime (gapp) file.

ARG_NAME This is the name of an argument option as it appears in the Storyboard Engine runtime (gapp) file.

ARG_TYPE This is the type of the argument and can be one of the following:

string	A text string value
integer	A numeric value with an optional range specified by 'min' and 'max' attributes
float	A floating point numeric value with an options range specified by a 'min' and 'max' attributes
boolean	A boolean true/false value

In order to be automatically included in a Storyboard Designer project, the render extension template file should be placed in the `templates` directory of the project where it is to be used. The name of the template file can be anything valid for the file system, but it should contain the file extension `.sbrt` in order to identify it as an render template file.

Photoshop Re-Import Feature

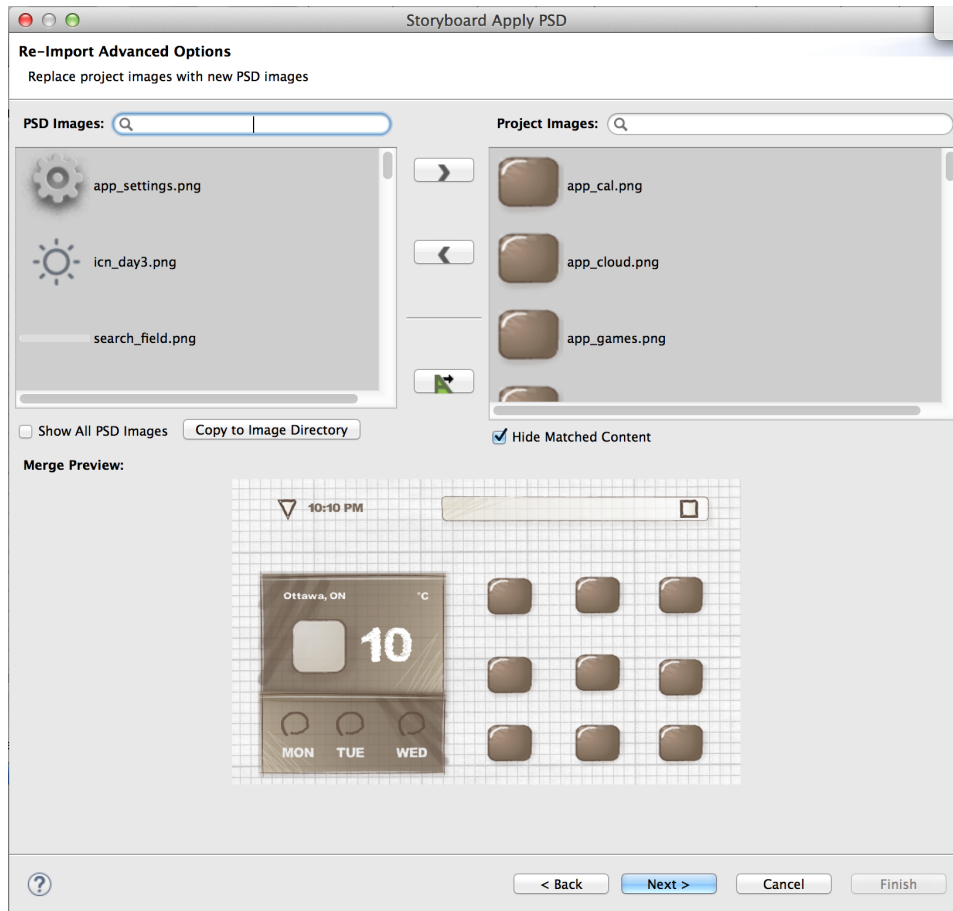
The Photoshop™ Re-Import feature enables the user to re-import a new or revised PSD file to replace existing images in a project's image directory. The re-import wizard is initiated from the main menu » File » Import

In the import dialog that appears, select the source PSD file to re-import and the destination Storyboard application where the new PSD image content will be placed. Then choose Next.

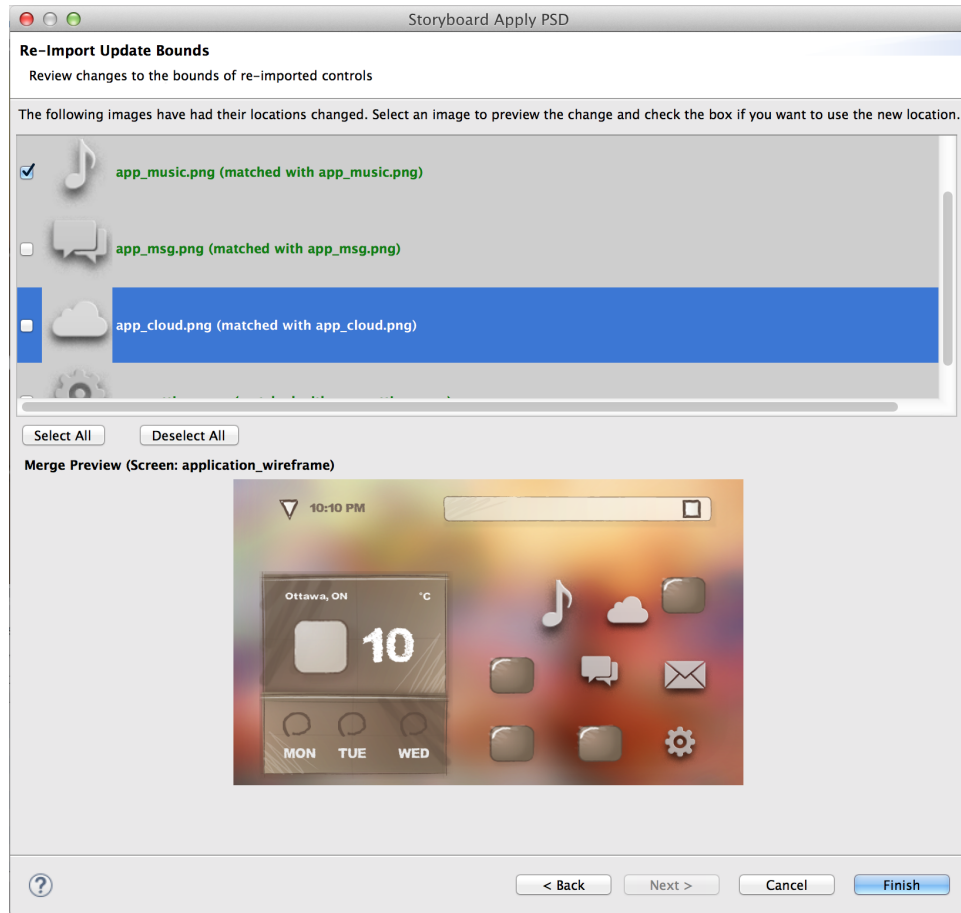
On the left of the dialog window is a list of image content from the PSD file. On the right is the content of the project's image directory. Selecting any PSD image will show candidate image match content from the PSD file. By clicking the "move right" arrow the selected PSD image is adopted as the new project image and replaces the model selection.

Below the separator in between the two viewers is the 'Match All Images' button. This will go through and automatically map any unique images of the same name. The name and extension must be identical for images to be mapped. If there is more than one identical match, the image won't be mapped.

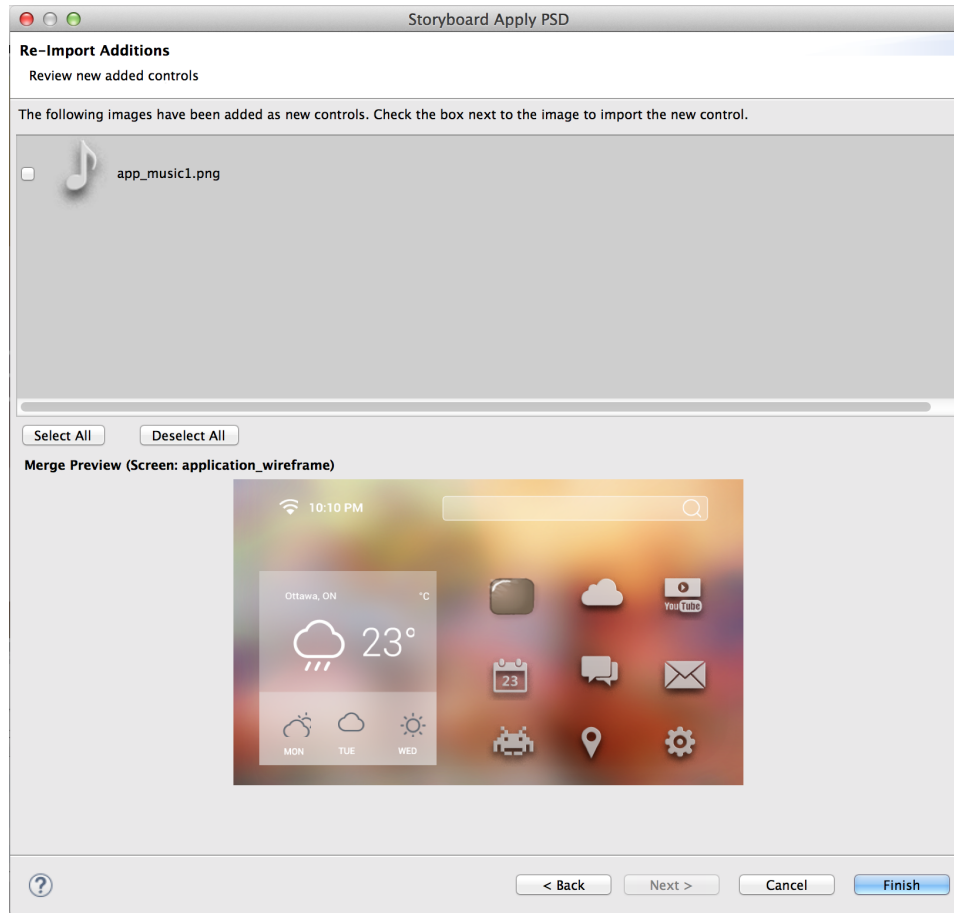
Matching and unmatching can be undone and redone by using the common keyboard shortcuts 'CTRL-Z' and 'CTRL-R'. If a PSD image does not have a project image with the same name it can still be added to the project image directory by clicking Copy to Image Directory button. Content that is being replaced will be shown in a preview below.



Choosing 'Next' brings you to a page that allows the user to update the bounds of any of the selected matches from the previous page. Any matches from the previous page that have changed location in the re-imported PSD file will appear in the list at the top. Selecting an item from this list shows the location change in the preview at the bottom, similar to the previous page. If you want to use the new location, check the box beside the match before hitting 'Finish'.



After updating the bounds, the next page allows the user to review any new images that have been added in the re-import. An image is considered new if it has no identical match in the current model, and it hasn't been mapped to anything in the first page. Selecting an item will preview it in the thumbnail below, similar to the previous page, and checking it will import the new control into the model.



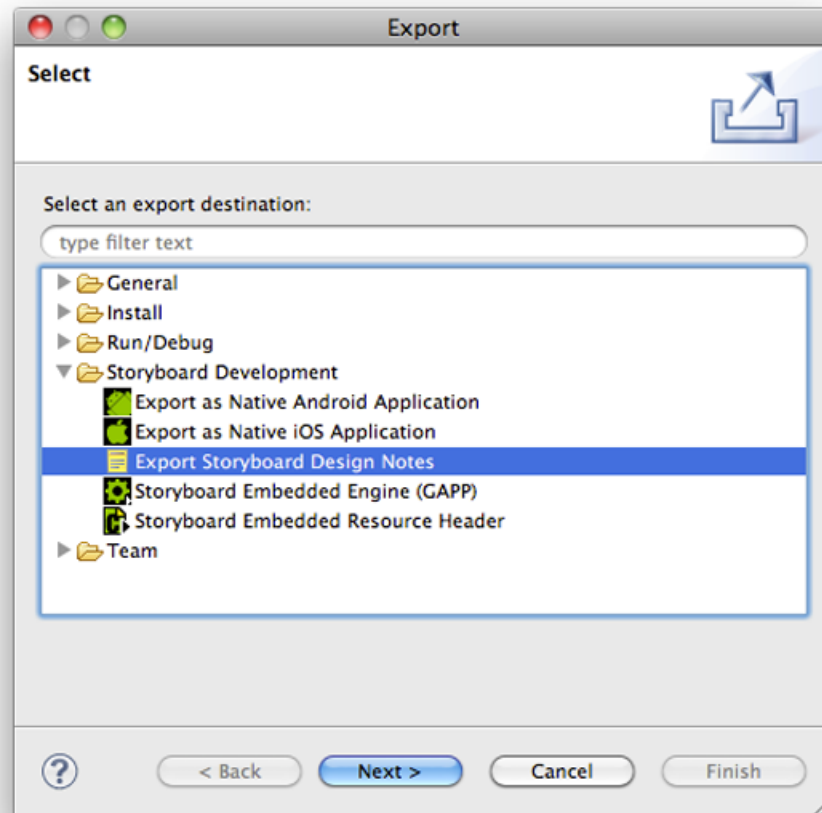
After choosing to replace, copy or add at least one image, clicking the Finish button will show a prompt that gives the option to overwrite image files that are being updated or keep a copy of the older images that are being replaced. The Photoshop PSD Re-Import feature is meant to update existing content using consistent naming. Using the Import Photoshop PSD File is better for adding new content to an existing project.

Storyboard Designer Utilities

Storyboard Designer offers many features to assist with the efficient development of your embedded user interface. Some of these features improve performance by reducing potential runtime inefficiencies, while other features speed the development of the application by providing greater insight into an existing design or allowing multiple designers to work in parallel on the application's user interface.

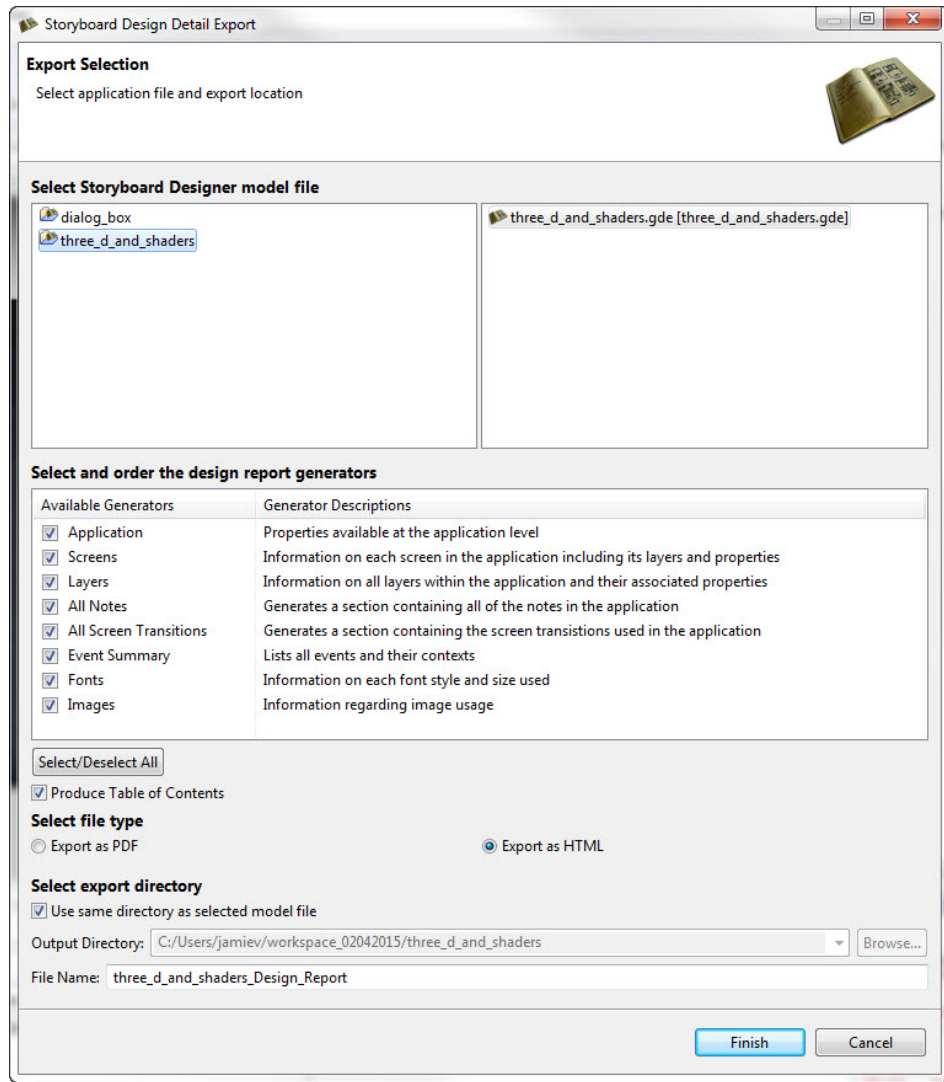
Design Notes

The Design Notes export wizard is used to generate an HTML or PDF report of the storyboard project.



The Design Note export can be accessed from **File > Export > Storyboard Design Notes** or by right-clicking a Storyboard GDE file and selecting Export Storyboard Design Notes. This will open a dialog allowing the user to select the file system location for the design report and the format, HTML or PDF, that the design report should be exported to.

The Design Notes report can be customized to contain a variety of information about the design model. Some of the available content options include screen transition information, content thumbnails, resource usage, and event and variable bindings.



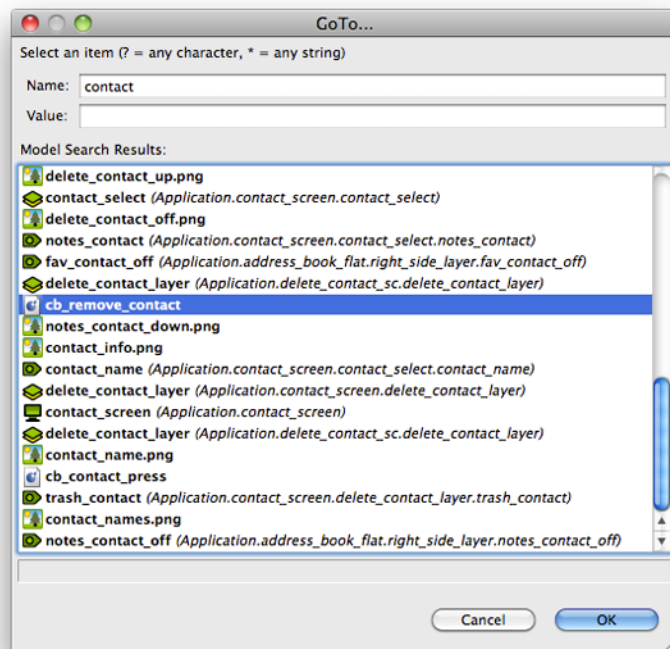
It is also possible to generate headless Design Note exports from the Storyboard Design files that can be used from a command line or scripting environment.

```
PATH_TO_INSTALL/Storyboard_Designer/storyboard/Storyboard -application
com.crank.gdt.designreport.designreport model=PATH_TO_GDE_MODEL format=[pdf|html]
output=PATH_TO_OUTPUT_FILE
```

The `model` is the full path to the Storyboard Designer model file and output parameter specifies the file system path where the report and associated resources will be created. The `format` parameter can be set to either `html` or `pdf` to indicate HTML or PDF outputs respectively.

GoTo Dialog

The GoTo dialog provides a quick way to locate and navigate to items in the application design. The GoTo dialog is activated when the CTRL+1 (Windows/Linux) or COMMAND+1 (Mac) keys are pressed while working in the main editor.



The Name and Value entries allow you to narrow the search criteria for the object you wish to find. The text entered here will filter the results in the list to only display search results matching what you have typed:

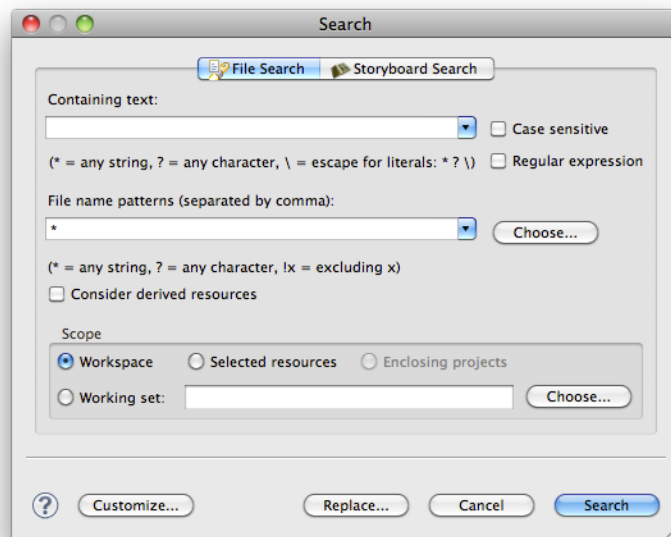
Name This selection limits the search to the primary name of the objects being searched. The name field also searches the name attributes of render extension and action attributes.

Value This selection limits the search to the value field of actions or render extensions being searched.

Storyboard Search Dialog

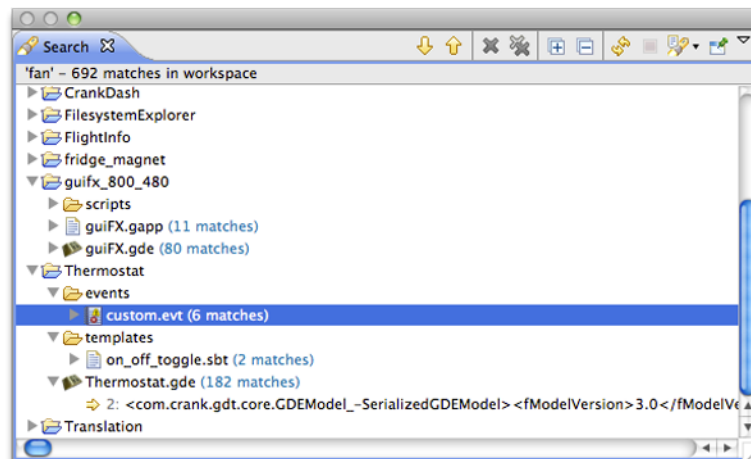
Storyboard Search provides an extension of the Go To functionality. Rather than providing the ability to see and navigate to a single selection, the results of a search are displayed in a tree format.

To search, invoke the Search dialog via the Search menu item or the CTRL+H key command. This will open up the search criteria dialog.



Similar to the GoTo dialog, the Name and Value search criteria allow model object names to be searched or in the case of actions and render extension arguments and variables also their values.

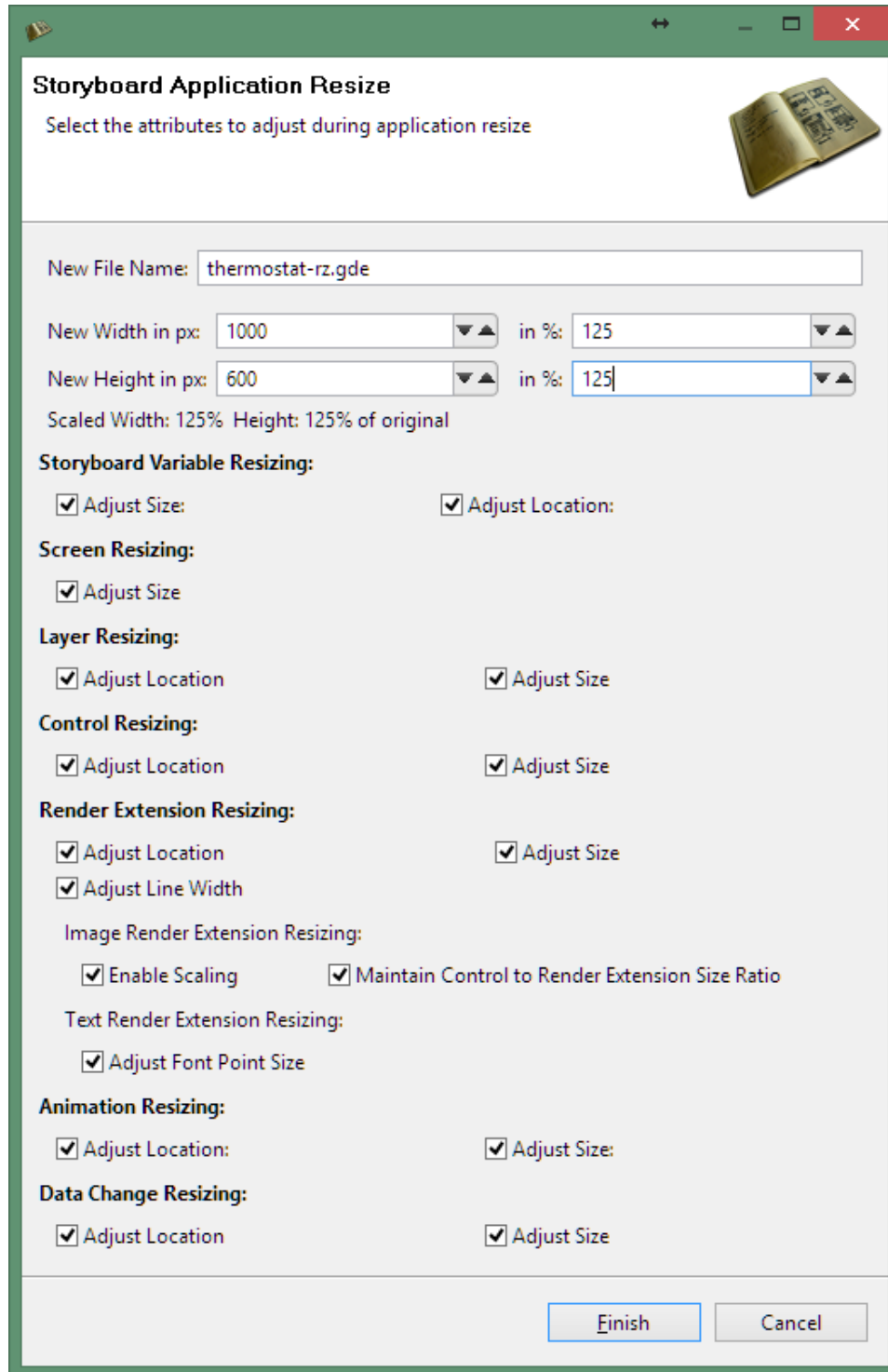
The search results will be displayed hierarchically in the Search view:



Where applicable, double clicking the results will cause the appropriate model element to be selected and brought forward in the main Storyboard editor.

Resize Storyboard Application

Use the Resize Storyboard Application dialog to create a new Storyboard Designer model file, with different screen width and height settings, based on an existing model file.



The application resize action dialog allows a Storyboard Designer model file to resize its screen dimensions. Since all of the layers, controls, and render extensions in a model are placed at specific locations within the model, the resize operation provides several parameters to allow the layers, controls and render extensions to be either re-positioned or re-scaled as appropriate.

The output of the resize action is a new model file located in the same file system location as the source model file. Creating this new scaled model file next to the original model file allows the designer to validate and inspect the scaled result before deciding to replace the original file.

Storyboard Variable Resizing

Adjust Location: The resize action will scale the location of all the user defined variables in the application that have been bound to a render extension location or center of rotation points.

Adjust Size: The resize action will scale the size of all the user defined variables in the application that have been bound to a render extension size.

Screen Resizing

Adjust Size: The resize action will scale the size of all the screens in the application.

Layer Resizing

Adjust Location: The resize action will scale the location for all of the layer instances in the application.

Adjust Size: The resize action will scale the size of all the layers in the application.

Control Resizing

Adjust Location: The resize action will scale the location for all of the controls in the application.

Adjust Size: The resize action will scale the size of all the controls in the application.

Render Extension Resizing

Adjust Line Width: The resize action will change the width of lines that appear in any render extension with the style set to “Line”. The line width will be scaled by the value of the smallest scale factor, be it that of height or that of width.

Image Render Extension Resizing

Enable Scaling: The resize action will enable the scale flag on all image render extensions. **Maintain Control to Render Extension Size Ratio:** “Enable Scaling” by itself will stretch images to fit the entire control and may not have the desired effect. This option will preserve the ratio from Control to Render Extension so the images will not look disproportionate or out of place within the control.

Text Render Extension Resizing

Adjust Font Point Size: The text font size will be scaled by the value of the smallest scale factor, be it that of height or that of width.

Animation Resizing

Adjust Location: The resize action will scale the start and end values for `grd_x`, `grd_y`. **Adjust Size:** The resize action will scale the start and end values for `grd_width` and `grd_height`.

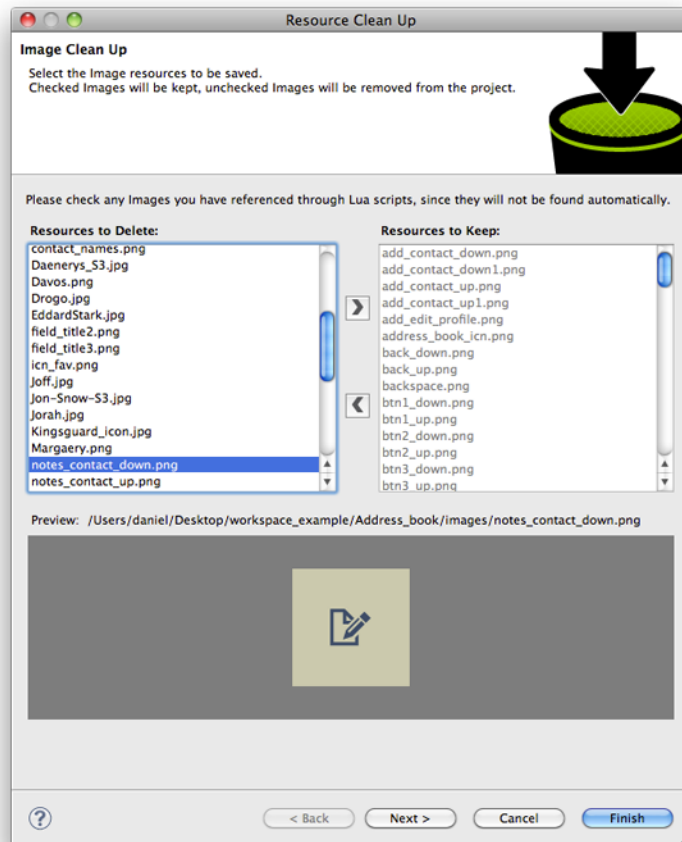
Data Change Resizing

Adjust Location: The resize action will scale values for `grd_x`, `grd_y`. **Adjust Size:** The resize action will scale values for `grd_width` and `grd_height`. For the best results we recommend performing a resize action with each of the options enabled and revisiting the resize dialog if one or some of the options did not produce the desired results. The resizer is a tool to reduce the amount of time spent in the event that an

application must be resized, with that said, there is usually some fine grain touches left to be completed by the application developer.

Resource Clean Up Wizard

Run the Resource Clean Up wizard from within the Images view or by selecting the model file in the Navigator view and selecting **Resource Clean Up...** from the menu.



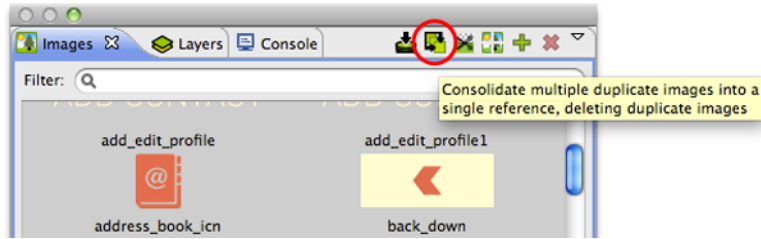
The Resource Clean Up wizard is used to remove resources that are present in the workspace but are no longer referenced by the project. In the wizard, the list on the left side contains the resources (fonts, images) and is used to select the resources that should be maintained/kept in the project. All resources that are not selected will be permanently removed from the project and file system. The preview on the right side contains a preview of the selected resource and the file system location of that resource.

This tool can only detect those resources that are directly referenced by the project. It is important that resources that are referenced from external programs or scripts be manually checked so as to prevent their removal from the project.

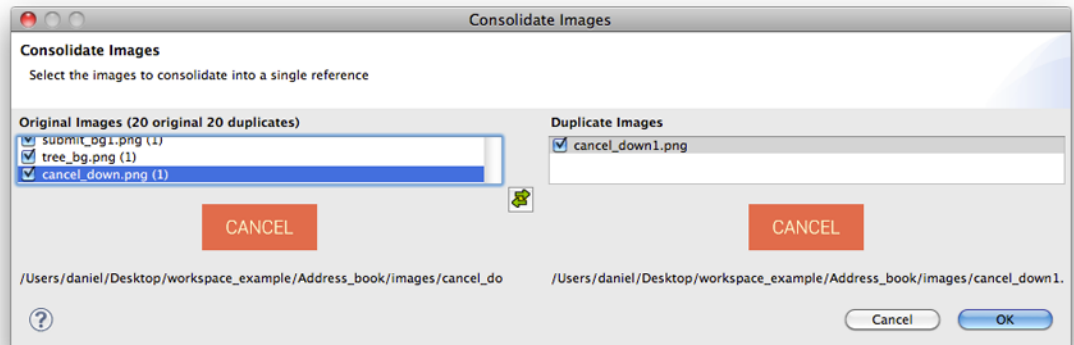
Once all of the resources to remove have been identified, selecting OK will perform the permanent removal of those files from the file system.

Consolidate Images Wizard

The Consolidate Images Wizard can be launched from the toolbar in the Images view.



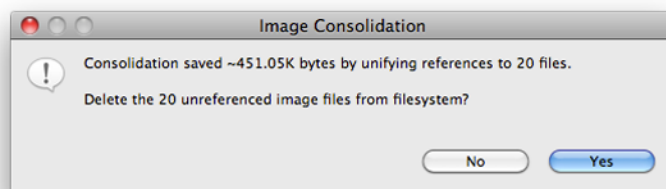
Launching the Consolidate Images wizard will start an analysis of all of the images in the workspace. These images are compared byte by byte to determine if their content is identical and can safely be consolidated together into a single reference.



Once the analysis is complete, a dialog will present the results showing the duplicate images that have been detected and provides a visual comparison of the source and reference images to ensure that they are indeed different.

By default, all duplicates will be consolidated into a single reference. To remove a reference from being consolidated, deselect the item.

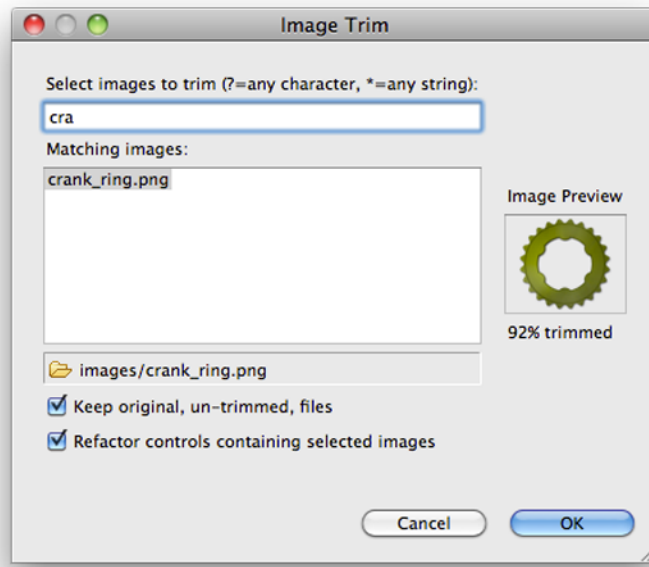
Selecting OK will search the model and consolidate variable and argument references to unify their references.



After all of the references have been combined there are likely to be a number of images that are no longer used. These can be immediately deleted from the workspace at this point.

Trim Images Wizard

The Trim Images Wizard can be launched from the toolbar in the Images view.



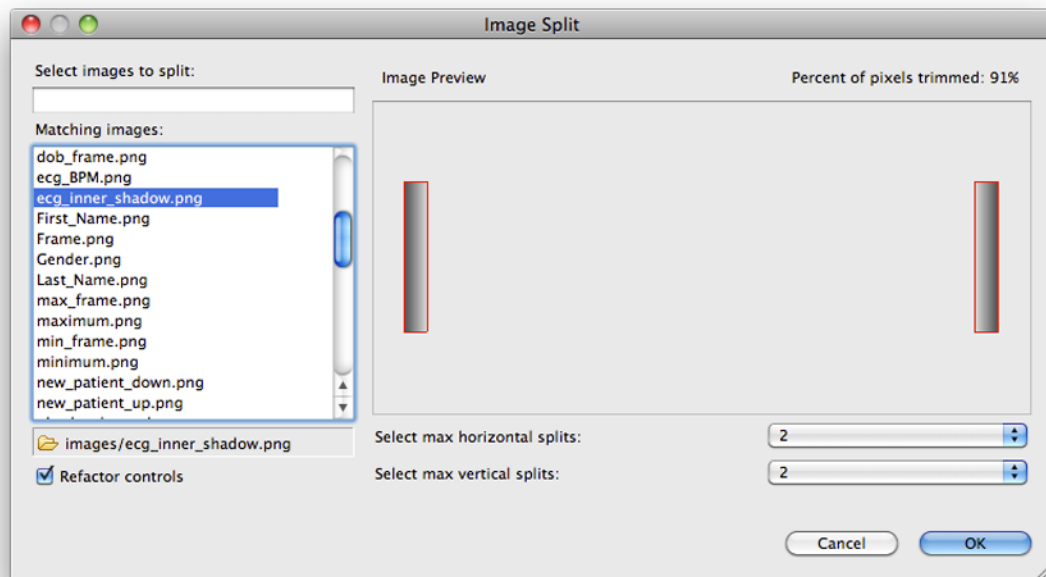
Launching the Image Trim wizard starts an analysis of either all (no selection made) or just the selected images from the Image view. It scans the image looking for transparent pixels on the borders of the image, in order to shrink the image to its smallest possible size.

The Wizard displays the selected images (or all images, if no selection) that have any pixels to be trimmed. The image preview shows the candidate image, and provides information on how much of the image will get trimmed. By default, the wizard will make copies of the original, untrimmed files and keep them in your file system. If you would like to delete the original untrimmed files, simply unselect the appropriate checkbox.

The wizard will also refactor all controls containing this image, while maintaining the proper control size and the position of the image within each control.

Split Images Wizard

The Split Images Wizard can be launched from the toolbar in the Images view.



Launching the Image Split wizard starts an analysis of either all (no selection made) or just the selected images from the Image view. It scans the image once, looking for transparent pixels and calculates how the image should be split to get rid of the maximum amount of transparent pixels.

The preview displays a red outline of exactly how the image will be split, and will update according to how many horizontal and vertical splits you specify. The percent of pixels eliminated is displayed in the top right corner.

When you select OK in the dialog the image split operation will proceed on those images you have selected. After splitting the images, all references in your app to the previous image will be refactored such that the controls containing those images will be replaced with the new split images positioned appropriately.

If you would like to leave the controls as they are and not perform the refactoring, simply deselect the "Refactor controls" check box at the bottom of the wizard.

Merge Control Images

This Merge Control Images can be launched from the Application Model view or by right clicking on the desired controls: right click on control(s) *Manage -> Merge Control Images*.

This can be used when you have one or more controls with static image content that will not change during runtime. It is sometimes more efficient to flatten those render extensions into a single static image. The controls must be in the same layer for the wizard to work correctly.

Collaboration and Team Development

Traditional development techniques rely on a common source code repository that is revision controlled using tools such as SVN, Perforce, Clearcase, GIT or Mercurial. Storyboard Designer projects are designed to be directly integrated into this type of environment so that the UI can be shared and improved by many developers working in parallel.

Revision Control System Integration

In order to provide an integrated support for various revision control systems, Designer uses the *Eclipse Team Provider* plugins. Plugins are available for most revision control systems from marketplace.eclipse.org [<http://marketplace.eclipse.org>]. Here are links for several of the more common/popular plugins:

- Subversion svn Plugin: <http://marketplace.eclipse.org/content/subversive-svn-team-provider>
- Mercurial hg Plugin: <http://marketplace.eclipse.org/content/mercurialeclipse-was-hgeclipse>
- GIT egit Plugin: <http://marketplace.eclipse.org/content/egit-git-team-provider>
- Performce p4 Plugin: http://www.perforce.com/product/components/eclipse_plugin

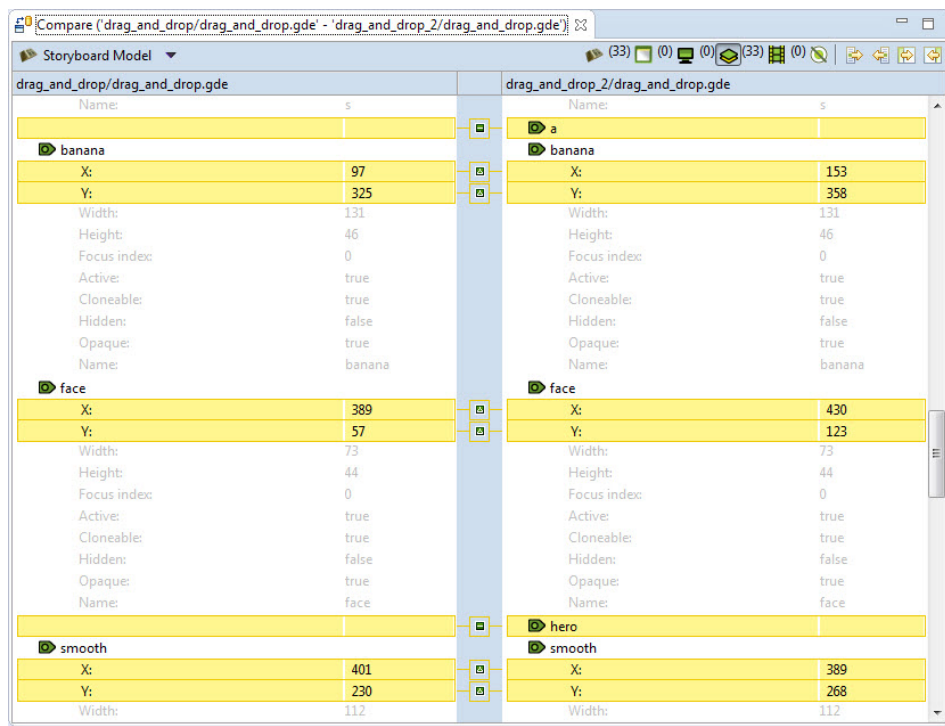
Comparing and Merging Model Files

The Storyboard Designer model file is a single model file. Conflicting changes to this model file can be visually inspected and merged from within Designer using the model comparison tools.

To compare two Designer files within the same project or file system workspace. Select both of the model files (i.e. `file1.gde` and `file2.gde`) in the Navigator view. Right click and select **Compare With > Each Other** from the menu.

To compare a Designer file that is in under revision control to a previous version, right-click on the file and select the menu entry **Compare With**. Different version control systems provide different specific terms, but . In the sub-menu you can select **Latest from Repository** or **Revision** if you want to compare with a specific version.

In either case, local comparisons or comparisons with versions from a revision control system, the comparison will open an editor that will highlight the differences in the model elements in the two files and allow each of the changes to be viewed in context and merged or discarded as may be required by the final design.



Triggering a comparison provides a hierarchical breakdown of the models' objects, with two sides representing the two files. Any differences between the two models will be highlighted in yellow. The two types of differences are property changes and additions(deletions). Property changes show the value on both sides of the viewer. They are marked by a delta icon in the centre sash in the two way case, or an arrow representing the direction of the change in the three way case. For additions(deletions), the side representing the file that has the object will show it, and the other side will show empty space. Additions and deletions are marked by a '+' icon or a '-' icon, and an arrow representing the direction of the change in the three way case. The comparator will also show any unchanged objects/properties, for reference and context. They are displayed in gray text, with no background color. In a three way comparison with an ancestor, it is possible that a conflicting change exists, where both sides have modified the same object/property from the original ancestor. These will be highlighted in red.



The toolbar contains actions and options to merge changes and switch the view. From left to right, here is a description of each one:

- Toggle Graphical Compare: Enables the graphical compare, which will appear on the bottom half of the screen and allow the user to visualize the changes on a model object.
- All Changes Filter: Displays all changes across all model objects.
- Application Filter: Displays changes on the application level. This includes application properties, and any application level variables or actions that have been added or deleted.
- Screen Filter: Displays changes on the screen level. This includes screen properties, layer instance properties and layer instance additions/deletions.
- Layer Filter: Displays changes on the layer level. This includes layer/control/render extension properties and control/render extension additions/deletions.

- Animation Filter: Displays any changes related to animations.
- Toggle Unchanged Properties: Shows or hides the unchanged objects/properties.
- Copy Left-Right/Copy Right-Left: Merges changes that have been selected in the viewer. If a model object is selected, any changes to its children will be merged.
- Copy All Left-Right/Copy All Right-Left: Merges all changes. In the two way case, this is non-destructive and will not delete any deletions, but will add the additions. In the three way case, this respects the direction of the changes.

After making changes, saving the comparator tab will appropriately save the changes to the model file(s). Exiting the comparator without saving will revert any applied changes. The global undo/redo functions are also available to revert and re-apply changes.

Comparing and Merging Projects

Entire projects and directory structures can be compared as easily as comparing single model files. This will allow developers to understand which resources such as images, fonts and script files have changed in addition to the changes to the model logic. The same as a model file comparison, this can be performed by right clicking on two projects or directories and selecting **Compare With > Each Other** from the menu.

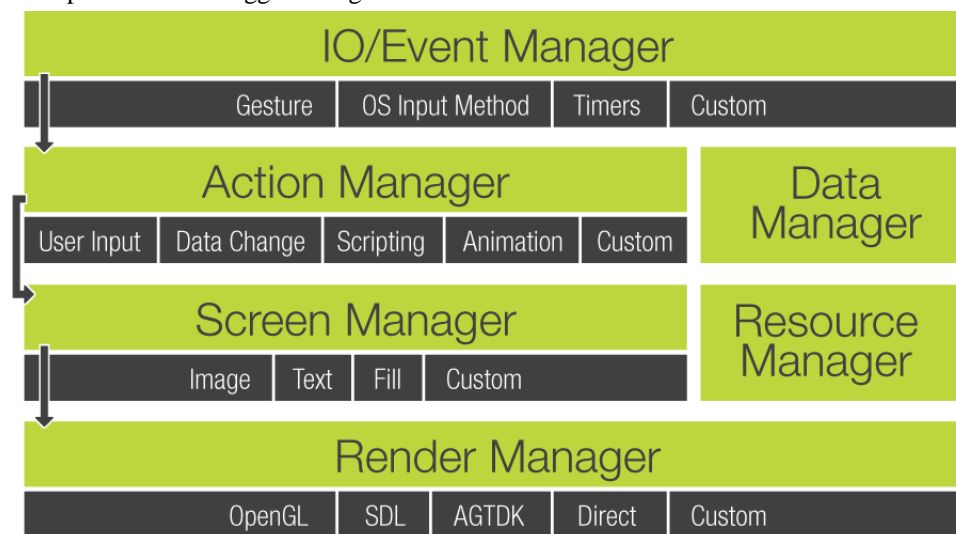
When this comparison is done a hierarchy is presented indicating files that have changed, been added or removed from the source or the destination directory. Selecting any of these files will open a comparison editor that is appropriate for that file type.

Chapter 3. Storyboard Engine

Introduction

The Storyboard Embedded Engine (Engine) is a runtime framework that allows a description of a graphical application to be interpreted and executed. The graphical application description, known as a deployment bundle, contains all of the instructions required to render screens to a display and to process events that would cause state transitions to occur in the application, potentially leading to additional screens being displayed.

The separation of the visual display logic from the system behaviour is achieved through the use of events. Events are asynchronous notifications containing a data payload that can be delivered to the Engine from multiple sources to trigger changes.



Target Configuration

When building a target configuration all of the application files and target binaries must be copied to the target system. The following is a description of the files needed for the target system and their runtime requirements. Collectively these files are referred to as a Storyboard *deployment bundle*.

Application Files

The deployment bundle must be accessible to the embedded target. The deployment bundle is exported from a Storyboard Designer model to a location on the host file system. For more information creating a deployment bundle, refer to the section Exporting to Storyboard Embedded Engine in this document.

A deployment bundle generally includes:

- A Storyboard application file (required). These are usually files with the extension `.gapp`.
- A `scripts` directory (optional). These are usually Lua script files that provide glue logic for the application
- An `images` directory (optional). These are image assets that are required by the application.

- A `fonts` directory (optional). These are fonts that are required by the application

Users may include additional directories as required by the application, but these are the standard contents of a deployment bundle

The directory structure below the root directory of the deployment bundle should not have its layout changed after being exported from Designer. All paths used within a Storyboard application for resources such as images, fonts and scripts are relative to the base directory containing the Storyboard application.

Setting up Storyboard Engine

An engine execution environment is provided for each supported operating system, architecture and rendering system. The target system should be configured with the Engine (`sbengine`) and plugins required for the target application. All plugins are loaded via the `SB_PLUGINS` environment variable and all libraries are loaded via the `LD_LIBRARY_PATH` environment variable.

For example:

```
export SB_PLUGINS=/home/crank/linux-imx6yocto-armle-opengles_2.0-obj/
plugins
```

```
export LD_LIBRARY_PATH=/home/crank/linux-imx6yocto-armle-opengles_2.0-
obj/lib
```

On Windows systems, there is no `LD_LIBRARY_PATH` so the `PATH` environment variable should be used instead. Similarly on MacOS, `DYLD_LIBRARY_PATH` should be used instead of `LD_LIBRARY_PATH`

Running Storyboard Engine

The Storyboard Engine executable (`sbengine`) is located in the `bin` folder of the Storyboard Engine directory structure. Now that the Storyboard Engine and Storyboard application (development bundle) are located on the embedded target and the Environmental Variables have been set, the Storyboard Engine can run a Storyboard application as follows:

```
sbengine thermostat.gapp
```

Command line Options

The Storyboard Engine is a self contained executable which loads plugins for added functionality. The Engine can be run as follows:

```
sbengine [-i] [-v] [-o] [storyboard application]
```

Table 3.1. Options

OPTION	DESCRIPTION
-i	displays which version of <code>sbengine</code> and related libs are being used
-v	verbosity, more v's means more verbose output.
-o	plugin or manager options

Each plugin or manager defines its name and possible options.

As the verbosity level to storyboard is increased, you will see more information about the execution of the runtime engine.

Engine Manager Options

Table 3.2. Action Manager Options

OPTION	DESCRIPTION
None	None

Table 3.3. Data Manager Options

OPTION	DESCRIPTION
None	None

Table 3.4. IO Manager Options

OPTION	DESCRIPTION
-oio_mgr,queue_size=4096	will limit the event queue size to a maximum of 4K. If the queue exceeds this size, events will be dropped and diagnostic messages will be logged regarding the dropped events. The default behaviour is to have an unlimited event queue size.

Table 3.5. Model Manager Options

OPTION	DESCRIPTION
-omodel_mgr,plugin_path=/temp	Sets the plugin path to the specified directory (/temp in this example). This setting overrides the SB_PLUGINS environment variable setting.
-omodel_mgr,fps=25	Limits the frame rate for all animations to a maximum fps specified (25 in this example).
-omodel_mgr,mem_stats=1	On platforms where process/task memory usage or heap allocator memory usage values are available, report them as performance log metrics. The value should be set to 1 to enable the statistics, future values are reserved.

Table 3.6. Render Manager Options: Windows, win32, OpenGL ES 2.0, x86

OPTION	DESCRIPTION
-orender_mgr,x=[xpos]	This will position the application at the defined x-position
-orender_mgr,y=[ypos]	This will position the application at the defined y-position

Table 3.7. Render Manager Options: Linux, sdl, x86

OPTION	DESCRIPTION
-orender_mgr,quality=[0 1 2]	Quality of image rotation. The default, 0 is fastest but lowest quality rendering. A setting of 2 is highest quality but slowest performing.

Table 3.8. Render Manager Options: Linux, fbdev, x86, armle

OPTION	DESCRIPTION
-orender_mgr,dblbuffer	Enable double buffering (fullscreen redraws)
-orender_mgr,fb=[device path]	The path to the framebuffer device to use (default=/dev/fb0)
-orender_mgr,fullscreen	Run in fullscreen mode
-orender_mgr,quality=[0 1 2]	Quality of image rotation. The default, 0 is fastest but lowest quality rendering. A setting of 2 is highest quality but slowest performing.

Table 3.9. Render Manager Options: Linux, directfb, x86, armle

OPTION	DESCRIPTION
-orender_mgr,dumpconfig	Dump the layer and graphics configuration information at startup
-orender_mgr,double	Enable double buffering (fullscreen redraws)
-orender_mgr,layer=[index]	Set the directfb layer index that content will render to (default=0)
-orender_mgr,quality=[0 1 2]	Quality of image rotation. The default, 0 is fastest but lowest quality rendering. A setting of 2 is highest quality but slowest performing.

Table 3.10. Render Manager Options: Linux, Windows CE, Windows Compact 7, Mac OSX, Neutrino 6.5, OpenGL ES 2.0, armle (Beagleboard)

OPTION	DESCRIPTION
-orender_mgr,quality=[0 1 2]	Set the rendering quality including shade model and texture filter, and image rotation quality. A value of 0=nearest/flat, 1=linear/flat, 2=nearest/smooth (default=linear/flat). For image rotation, lower numbers mean faster algorithms, but better quality
-orender_mgr,fullscreen	Run in fullscreen mode
-orender_mgr,multisample=[value]	This sets value indicates the degree of multisampling which affects the visual smoothness of edges. For example, 4 would be 4x multisampling while 0 would be no multisampling. The default value is 4.
-orender_mgr,vbo	use vertex buffer objects
-orender_mgr,scale=[aspect]	scale the application to the physical display size. If aspect is passed the application will retain the proper aspect ratio when scaled.
-orender_mgr,backbuffer	Render the scene using a damage rectangle. On some OpenGL ES implementations this will give better performance but will use more memory as it has to allocate a separate display buffer.

OPTION	DESCRIPTION
-orender_mgr,npot	Disable power-of-two texture allocations. By default the OpenGL ES API is queried to check for NPOT texture support. This option can be used to override this behaviour and force support. NPOT textures will use less memory for image data.
-orender_mgr,fontsize=[size]	Specify the size of the font texture sheet. Fonts are generated into sheets and the default size is 512x512. The number of glyphs put into the sheet is a function of the point size and the texture size. This option can be used to tune the number of available glyphs and the memory usage.
-orender_mgr>window_w=[w]	Scale the application content and window to the specified width. This option is only valid on desktop systems which use a window manager (MACOS and Windows). This option must be used along with 'window_h'
-orender_mgr>window_h=[h]	Scale the application content and window to the specified height. This option is only valid on desktop systems which use a window manager (MACOS and Windows). This option must be used along with 'window_w'
-orender_mgr,linejoin=[0 1]	Set line join style for path drawing, drawing joins can have a performance impact on frame rate. 0=none, 1=round (default=1)
-orender_mgr,clipmode=[stencil scissor]	Set the clipping mode to use, may have performance impacts. Each implementation defaults to the best performance.
-orender_mgr,error_event	An error event is generated for OpenGL render errors. Image and font errors will identify the image and font related to the error.
-orender_mgr,display=[index]	Connect to the given display index, this option is only available for the QNX Screen OpenGL ES 2.0 render manager or the Linux i.MX6 OpenGL ES 2.0 render manager where the value is the selected framebuffer index.
-orender_mgr,fb=[x]	This option pertains specifically to iMX6 hardware platforms. Starting at 0, x defines the framebuffer number to render to.
-orender_mgr,x=[xpos]	When using the QNX Screen engine this will position the application at the defined x-position
-orender_mgr,y=[ypos]	When using the QNX Screen engine this will position the application at the defined y-position.
-orender_mgr,rotated=[90 180 270]	Rotate the application by the defined angle.
-orender_mgr,zorder=[z]	When using the QNX Screen engine this will position the application window at the defined Z index.

Table 3.11. Render Manager Options: QNX Neutrino 6.5, Linux, Fujitsu Jade, armle

OPTION	DESCRIPTION
-orender_mgr,mainlayer=[number]	The main layer to use for rendering, defaults to layer 0
-orender_mgr,display=[number]	The display to connect to, defaults to display 0
-orender_mgr,conf_file=[path]	A path to a LCD configuration file which includes the display settings

Table 3.12. Render Manager Options: WinCE 6.0, Windows Compact 7, win32, armle

OPTION	DESCRIPTION
-orender_mgr,quality=[0 1 2]	quality of image rotation. 0 is fastest but lowest quality. 2 highest and slowest
-orender_mgr,fullscreen	Removes the window border and fills the display outside of the application area
-orender_mgr,dumpcaps	Print the device capabilities and acceleration flags
-orender_mgr,nohwcursor	disables the HW cursor

Table 3.13. Resource Manager Options

OPTION	DESCRIPTION
-oresource_mgr,image=4	will set the image cache limit to 4K for example (the texture memory on GPU based systems is related to the image cache limit as well)
-oresource_mgr,font=4	will set the font cache limit to 4K for example
-oresource_mgr,error=0	send or skip resource manager error events, default is 1 (send)

Table 3.14. Screen Manager Options

OPTION	DESCRIPTION
-oscreen_mgr,swcursor	enables the rendering of a software cursor
-oscreen_mgr,redraw_complete	generate a completed event for every screen update
-oscreen_mgr,dl=1	disable the use of a particular hardware layer
-oscreen_mgr,fps	display the frames per second of the display update, requires level 4 verbosity (-vvvv)
-oscreen_mgr,overlay	Allow the application to be run as an overlay and show the content below. This only functions on particular render managers and if the application has a transparent background.

Plugins

The Storyboard product ships with a standard set of plugins which add functionality to the system. Plugins are loaded based on the SB_PLUGINS environment variable. This variable can be a directory where all plugins are loaded from or a “;” separated list of plugins.

Some plugins have options that can be passed via the command line to the plugin. To pass on option to a plugin use the -o option to sbengine in this format -o[PLUGIN_NAME],[PLUGIN OPTION]

Table 3.15. 3D model rendering: libgre-plugin-model3d.so

OPTION	DESCRIPTION
-omodel3d,novbo	Disable the use of vertex buffer objects, by default Vertex buffer objects are used for rendering.

Table 3.16. Capture/Playback: libgre-plugin-capture-playback.so

OPTION	DESCRIPTION
-ocapture_playback,mode=[capture playback]	Specifies the behaviour of the plugin for either playback or capture. If capture is specified then input events (gre.press, gre.release, gre.motion, gre.keypress etc) will be logged and stored in an output file. If playback is specified then the contents of an input file are read and the input events are injected back into the application.
-ocapture_playback,file=filename	If "capture" is specified as the mode it indicates the contents of the file specified will be overwritten with the new event stream. If "playback" is specified then the contents of the file specified will be used as an event stream source.
-ocapture_playback,verbosity	Indicates that the plugin should log events that it is either capturing or playing back to the standard output
-ocapture_playback,loop=count	Playback option that indicates how many times the playback should iterate through its content. The default is to playback the content once, otherwise if a count is specified the plugin will replay the full content count times.
-ocapture_playback,quit_playback	Playback option that indicates if a gre.quit event should be automatically generated after playback is complete. The default is to not generate a quit event.

Table 3.17. Gesture: libgre-plugin-gesture.so

OPTION	DESCRIPTION
-ogesture,file=filename	filename is a text file containing custom gesture definitions.

OPTION	DESCRIPTION
-ogesture,mode=[single multi]	single allows for single touch gestures. multi allows for both single and multi-touch gestures. By default both single and multi touch gestures are disabled
-ogesture,threshold=[level]	Specifying a level allows the user to configure the sensitivity level which determines if we translate a motion into a gesture event. By default the threshold is set to 100.
	See below for more options

Table 3.18. Linux Input Support: libgre-plugin-dev-input.so

OPTION	DESCRIPTION
-odev-input,mouse=/dev/input/event0	The name of the mouse device, for example /dev/input/event0
-odev-input,kbd=/dev/input/event1	The name of the keyboard device, for example /dev/input/event1
	One of either the mouse or kbd arguments must be passed to enable this plugin. There are no default bindings so the full path to the desired input device must be specified.

Table 3.19. Lua Scripting: libgre-plugin-lua.so

OPTION	DESCRIPTION
-olua,hold=[0 1]	This option controls how Lua posts data manager change notifications. By default all changes are held until the end of script execution (1) but if 0 is specified, change notifications and events are triggered as soon as changes are made using <code>gre.set_data</code> calls.
-olua,gc=[0 1]	This option is used to minimize the runtime memory footprint of the Lua script engine by invoking the Lua garbage collector after every Lua action. By default this option is set to 0 indicating that garbage collection will occur at the natural points specified by Lua's <code>collectgarbage</code> option. If the value is set to 1, then garbage collection is run after every Storyboard Lua action invocation, reducing the active runtime memory footprint with a slight cost to execution performance.

Table 3.20. Linux Multi-touch Protocol: libgre-plugin-mtdev.so

OPTION	DESCRIPTION
-omtdev,device=[path to touch device]	Plugin for Linux Multitouch Protocol to be used with kernels supporting multitouch events. This plugin is only available for the imx6 runtime currently.

OPTION	DESCRIPTION
-omtdev,rotation=[0 90 180 270]	Clockwise rotation of the touch input coordinates. When X1,Y1 is in the top-left corner, rotation is 0.
-omtdev,bounds=[X1:Y1:X2:Y2]	Where X1,Y1 is the top-left corner and X2,Y2 is the bottom-right corner of the touch screen device. On some screens, X1 may be less than X2 and Y1 may be less than Y2.
-omtdev,max_x=[max touch value in the x direction]	DEPRECATED in Storyboard 4.2
-omtdev,max_y=[max touch value in the y direction]	DEPRECATED in Storyboard 4.2
-omtdev,threshold=[integer]	This is the number of pixels a touch point has to move in order to generate a motion event, the default value is 1
-omtdev,points=[integer]	This is the number of multitouch fingers that is supported. Events will only be generated for this number of fingers in contact with the screen, the default is 5

Table 3.21. Linux Touchscreen Support: libre-plugin-tslib.so

OPTION	DESCRIPTION
-otslib,pressure=1	Set the pressure value with corresponds to a press, the default is any value greater than 0 is a press.
-otslib,motion=5	The number of consecutive motion events to compress, can be useful on a device which delivers a high rate of motion events, default is to not compress.
-otslib,calibrate	Put tslib into raw mode which is used for calibration.
	If you do not have the following <code>tslib</code> variables setup the plugin will not load or function properly. TSLIB_CONSOLEDEVICE TSLIB_TSDEVICE TSLIB_CALIBFILE TSLIB_CONFFILE

Table 3.22. Logger: libre-plugin-logger.so

OPTION	DESCRIPTION
-ologger,output=[filename]	This option specifies a path in the file system to direct the Storyboard standard output to. The directory path to the file must already exist. + At the beginning of the file name will append to the log file, otherwise the file will be overridden on each invocation of sbengine. %D in the filename will be replaced by a date stamp with YYYY-MM-DD format.

OPTION	DESCRIPTION
	<p>%T in the filename will be replaced by a 24h time stamp with HHMMSS format.</p> <p>Both %D and %T may be used on the same filename.</p> <p>A valid command would be: <code>-o logger, output=/logs/log-%D-%T.txt</code>, provided the /logs/ directory existed prior to runtime.</p>
<code>-o logger,buffer=[bytes]</code>	This option will buffer all logging output to an allocated in-memory buffer and only flush the output when the buffer content is full. The number of bytes allocated for the buffer are provided by the option to the argument. If the buffer size is 0 or invalid, then 4K will be allocated for the buffer.
<code>-o logger,event=[event_name]</code>	This option will enable the generation of a custom Storyboard event any time that an ERROR message is detected by the Storyboard Engine logging system. When the error is detected, an event "event_name" will be sent to the application and it will contain a payload of "1s0 msg" where the "msg" is the diagnostic string that would have been logged.
<code>-o logger,slogger=[opcode]</code> (QNX ONLY)	<p>This option specifies that sbengine should use the QNX system logging infrastructure. if no opcode is given, sbengine will provide an appropriate opcode.</p> <p>The opcode is a combination of a major and minor code. Create the opcode using the <code>_SLOG_SETCODE(major, minor)</code> macro that's defined in <code>sys/slog.h</code>.</p> <p>This option is only available for systems running QNX.</p>
<code>-o logger,io=[level]</code>	This option enables the logging of IO events in variable levels of verbosity. If no option is specified then the level will default to 1 and the event name and its size in bytes will be displayed. If 2 is specified then the format is displayed. If 3 is specified then the data payload will be dumped to the standard output in both hexadecimal and character formats.
<code>-o logger,data</code>	This option enables the logging of data change events as they occur. The data key that has been changed is displayed to standard output.
<code>-o logger,perf</code>	This option enables the logging of performance data to the standard output (or file if <code>perf_file</code> is used). If a value of 0 is specified to the perf option then performance logging is enabled, but the capture of data is not immediately started and can be toggled using the <code>gra.perf_state</code> action. If the value is set to 1 or is not specified, then

OPTION	DESCRIPTION
	performance data will be immediately captured. For example <code>sbengine -ologger,perf=1</code> will enable performance logging with the immediate capture of performance metrics.
<code>-ologger,perf_file=[filename]</code>	This option specifies a path in the file system to direct the performance data output to. The directory path to the file must already exist and the contents of the file will be overwritten on each invocation of <code>sbengine</code> .
<code>-ologger,filter=[keyword]</code>	This option enabled filtering of events based on the keyword provided. Multiple keywords can be specified up to a maximum of 5. Each keyword can also be negated by the '^' symbol. Therefor we could ignore all motion events by passing in the following command: <code>sbengine -ologger,filter=^motion</code> Filtering applies to the 'io' and 'data' options.
	If performance logging is enabled then the output is a set of comma separated values (CSV) with the following fields: PERF, application time, type, operation, name, duration
application time	This is the time that the performance event was finished relative to the start time of the application in milliseconds.
type	This is the type of performance operation that was recorded as a broad classifier
operation	This is a sub-classification of the type used for additional tracing granularity
name	This is an identifier that can be used, along with the type and operation fields, to identify the context of the performance operation being performed
duration	This is the duration of the operation in milliseconds

Table 3.23. QNX input support: libgre-plugin-gfi-input.so

OPTION	DESCRIPTION
<code>-ogfi-input,mouse=/dev/devi/mouse0</code>	The name of the mouse device, for example <code>/dev/devi/mouse0</code>
<code>-ogfi-input,rotated=[90 270]</code>	If specified, this indicates that the input co-ordinates should be rotated by 90 or 270 degrees
	By default the input system used the gfi interface based on the devi drivers. The devi driver must be run with the <code>-P</code> option. If you pass the <code>mouse</code> option then the mouse/touchscreen is used in raw mode.

Table 3.24. Storyboard IO: libgre-plugin-greio.so

OPTION	DESCRIPTION
-ogreio,channel=name	The value specifies the name that the applicaitons Storyboard IO channel will use. This name can then be used by <code>gre_io_open</code> or <code>iogen</code> clients to send events to the application.
-ogreio,queue	This flag indicates if the events on the Storyboard IO channel should be asynchronously queued into the application's message queue or if a new event should be added only after the last event has been processed. The default is to only have one event being processed by the application at a time.

Gesture

Capture mouse or touchscreen events and generate gesture events

Name: `gesture`

Plugin: `libgre-plugin-gesture.so`

Options: `file=filename`

`mode=[single|multi]`

`threshold=[level]`

File is the filename of a text file containing custom gesture definitions.

Mode specifies if we are anticipating single or multi-touch events.

Threshold is the sensitivity level which determines if we translate a motion into a gesture event.

What the gesture plugin does is to translates input `gre.move` events from a mouse or a touchscreen into a gesture event. When a button is depressed, or your finger is dragged across the touchscreen, the gesture plugin tracks it. When the button is release or you take your finger off of the touchscreen, the plugin emits the gesture.

The gesture plugin also handles input from multiple touch points when running on a multi-touch enabled device.

The conventional gestures below are only generated from the mouse or a single touch point. Using Multiple Touch points will generate other events described later in this document.

Gestures are made up of a series of numbers. The numbers represent the direction that the cursor was traveling as a grid arranged from 1 to 8:

- 1 Up
- 2 Up and to the Right
- 3 Right
- 4 Down and to the Right
- 5 Down

6 Down and to the Left

7 Left

8 Up and to the Left

By default the gesture plugin registers the Up, Right, Down, and Left gestures as 1, 3, 5, 7. The numbers 2, 4, 6, 8 aren't enabled by default, but you can define them in a custom gesture definition file. The gesture definition file is a text file that defines an event, followed by the gesture number.

For example, to define a z gesture, you could put the following in the a `gesture-definition.txt` file:

```
gre.gesture.ze,363
```

This definition stats when the gesture plugin detects a right motion, followed by a down and to the left motion, followed by another right motion, it will emit a `gre.gesture.ze` event.

You can point the gesture plugin at the custom gesture definition file by running storyboard with the option "gesture,file=filename" , where filename would be the name of the file like, `gesture-definition.txt`, from the example above. Another option is to export the environment variable `GESTURE_DEF_FILE` which will set to the path to the custom definition file.

It should be noted that gestures are capped at a maximum of 30 motions and anything above this limit will cause a warning and be ignored.

Multi-Touch Gestures

Unlike the single touch gestures, which state which gesture you have just entered, the multi-touch gestures are events that fire whenever you have more than one finger on the touchscreen. The plugin tracks up to five contact points, if 6 or more are present they will simply be ignored by the plugin. The events the plugin listens to are `gre.press`, `gre.release`, and `gre.motion` to track the touchscreen info while only one finger is present and `gre.mtpress`, `gre.mtrelease`, and `gre.mtmotion`, to track the touchscreen info while multiple touches are present. Note when using a multi-touch enabled device single the press, release and motion events will be sent only while there is only one touch point present. As soon as there are multiple touch points present, all events will be mt events.

After listening to the events, if more than one touch point is present and one or more touch points move, the plugin will do an update where it compares the old touch locations to the updated touch locations and generates the related multi-touch gesture events. These events are all of the form `gre.mtXaction`, where X is the number of touch points present (between 2 and 5), and action is the name of the event, which will be one of, move, pinch or rotate.

All multi-touch gesture events have the same format of a gre pointer event, with a few extra data fields.

```
gre.mtXmove
```

This event has an `x_move` and `y_move` data field, which will be the difference in x and y of the midpoint of all present touch touches between the current and last event sent from the touchscreen.

```
gre.mtXpinch
```

This event has a value data field, which will be the scale factor of the average spacing from all current touch points compared to the spacing of all the old touch points. The scale factor is calculated by `newspacing/oldspacing`, so a value of 1.1 indicates a growth of 10% and a value of 0.9 indicates a shrink of 10%

```
gre.mtXrotate
```

This event has a value data field, which will be the difference in rotation between the average angle of all current touch points compared to the average angle of all the previous touch points. The value will be in degrees.

Custom Shader Support

Storyboard supports custom OpenGL ES shaders written in GLSL. Shader programs can be attached to controls by creating a vertex and fragment shader program. These programs are then compiled at runtime and used by the Storyboard Engine. When creating a shader the uniforms can be manipulated in Storyboard Designer through variables. The naming of the shader uniform determines how it's variable is resolved. All shader variables must be float type variables. The uniform naming can be prefixed in order to tell Storyboard which context to resolve the variable:

```
grd_a: This variable is resolved at the application level
grd_l: This variable is resolved at the Layer level (layer where the
control is)
grd_g: This variable is resolved at the Group level (group where the
control is)
grd_c: This variable is resolved at the control level (control where
the shader is connected). This is the default if no prefix is used
```

Fragment shader Example:

All variables can be created at the application level. The variables would be:

```
r: float
g: float
b: float
a: float
```

The program would be:

```
uniform float grd_a_r;
uniform float grd_a_g;
uniform float grd_a_b;
uniform float grd_a_a;

void main (void)
{
    gl_FragColor = vec4(grd_a_r, grd_a_g, grd_a_b, grd_a_a);
}
```

Vertex shader Example:

```
attribute vec4 myVertex;
attribute vec4 myUV;

varying vec2 myTexCoord;

uniform mat4 projMatrix;
uniform mat4 mvMatrix;
```

```
void main(void)
{
    gl_Position = projMatrix * mvMatrix * myVertex;
    myTexCoord = myUV.st;
}
```

Font Environment Variable

sbengine has environment variables for font options:

```
SB_FONT_HINT="normal"
```

This corresponds to the default hinting algorithm, optimized for standard gray-level rendering.

```
SB_FONT_HINT="light"
```

A lighter hinting algorithm for non-monochrome modes. Many generated glyphs are more fuzzy but better resemble its original shape. A bit like rendering on Mac OS X.

```
SB_FONT_CACHE_SIZE=[size]
```

This variable is used to set the size of the FreeType font face cache. By default the cache is disabled and a value of 0 will disable the cache. any other number is the number of font faces to cache, using this cache can decrease memory mappings of font files.

System Specific Requirements

Certain operating system or rendering platforms require additional configuration parameters or extra environment variables. The following is a list of platform notes for setting up Storyboard Engine.

The most recent target configuration information is available in the online support forums for Storyboard Engine at www.cranksoftware.com/forums.

Linux FBDEV x86, armle

Requirements:

This build renders directly to the Linux framebuffer device (/dev/fb0). No other Graphical User Interface should be running when Storyboard is started as it assumes control of the framebuffer device. This build also uses the FreeType library for font loading and rendering.

For the ARM version a plugin is available which supports a touchscreen device through the use of tslib (libgre-plugin-tslib.so). This plugin will use the standard tslib environment variables in order to find and configure the touch device as follows:

```
export TSLIB_CONSOLEDEVICE=none
export TSLIB_TSDEVICE=/dev/input/ts0
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/etc/ts.conf
```

It is assumed that the touch device has been configured previously. In order to configure the touch device please run the `ts_calibrate` which is part of the tslib distribution or build for Linux systems.

If your application uses the Storyboard IO library then the Linux kernel must have SysV message queue support.

Libraries:

- `libfreetype.so`
- `libts.so` (only for tslib plugin)

Note

Storyboard requires a `libts-0.0.so.0` to be in the lib path to use the tslib-plugin. If the board has `libts-1.0.so.0` simply create a symlink for `libts-0.0.so.0` and point it at the `libts-1.0.so.0`.

```
ln -s /lib/libts-1.0.so.0 /lib/libts-0.0.so.0
```

Microsoft WinCE, Compact7 win32, armle

Requirements:

Alpha blending must be compiled in to the target WinCE image

To utilize the `-v` verbosity options, a console must be compiled in to the target WinCE image

Use command line option to pass `SB_PLUGINS` directory since WinCE does not support environment variables. Eg: `sbengine -omodel_mgr,plugin_path="/Temp"`

`liblua.dll` must reside in the same directory as `sbengine.exe`, due to the lack of a `PATH` environment variable

Libraries:

- `libgwes.dll` Must be built into target WinCE image

Yocto Jethro Linux kernel (3.14) OpenGL, FBDEV, armle

Using the Yocto Jethro linux kernel (3.14) with the boundary devices branch for the nitrogen6x you might encounter flickering graphics.

Requirements:

```
echo 10 >/sys/devices/soc0/backlight_lvs0.17/backlight/backlight_lvs0.17/brightness
```

Chapter 4. Storyboard Media

Introduction

The Storyboard runtime engine provides a media plugin interface in order to support the playback of several different types of audio and video formats. The media plugin accomplishes this by communicating with an external media player back-end application. Media back-end applications are provided for particular platforms based on their underlying media support. Video playback can be accomplished by either using the Storyboard External render extension in order to overlay the video content within Storyboard or by controlling an external display layer as hardware permits.

Video and Audio playback is accomplished by using a set of defined actions. When creating a media application you must include these media actions in your Storyboard project. The action definition file can be found in your Storyboard Installation under:

Samples/action_definitions/media.sbat

Copy this file to the “templates” directory of your project in order to expose the actions.

A sample video application can be found in the Storyboard online repository in order to demonstrate the actions provided.

Media Actions

The following actions can be used to control the media playback. Note that all actions take a “channel_name” argument. This is used to target a specific playback channel. For example if a video is started with “gra.media.new.video” with “channel_name=video1” then any subsequent action which wants to act on this video, such as play/pause, must set the channel “video1”.

gra.media.new.audio

Tells the plugin to play a new audio file.

The action arguments are:

channel_name	The channel name the new video is to be played on
media_name	The name of the media to play, full path to an audio file
volume	The initial volume value to play the media at. The value should be between 0 and 100.
update_interval	The number of milliseconds to wait in between update messages
emit_time_events	A value that is set to 1 to emit time update events, 0 otherwise
extra_data	Any extra data that should be passed to the back-end, can be NULL

gra.media.new.video

Tells the plugin to play a new video file.

The action arguments are:

channel_name	The channel name the new video is to be played on
media_name	The name of the media to play, full path to a video file
volume	The initial volume that the media should be played at
object_name	The name of the external object to display content on. This is necessary when using an external render extension to display the content, please refer to the external render extension documentation
external_name	The name of the render extension to display content on. This is necessary when using an external render extension to display the content, please refer to the external render extension documentation
update_interval	The number of milliseconds to wait in between update messages
emit_time_events	A value that is set to 1 to emit time update events, 0 otherwise
output_width	The width of the video
output_height	The height of the video
output_depth	The output depth of the video in bytes per pixel. 16bit = 2, 24bit = 3, 32bit = 4
extra_data	Any extra data that should be passed to the back-end, can be NULL. See each backend for a description of this data

gra.media.volume

Triggers a change in the playback volume.

The action arguments are:

channel_name	The channel name to change the volume on
volume	The value to change the volume to, a number between 0 and 100
emit_volume_event	A value that is set to 1 if an event should be emitted or 0 otherwise

gra.media.seek

Triggers a change to the current playback position of the media that is playing.

The action arguments are:

channel_name	The channel name to change the seek position on
seek_num	The new seek position for the media file
emit_state_event	A value that is set to 1 if an event should be emitted or 0 otherwise

gra.media.stop

Changes the media playback state to stopped.

The action arguments are:

channel_name	The channel name to change the state on
emit_state_event	A value that is set to 1 if an event should be emitted or 0 otherwise

gra.media.playpause

Changes the media playback state from paused to playing or from playing to paused

The action arguments are:

channel_name	The channel name to change the state on
emit_state_event	A value that is set to 1 if an event should be emitted or 0 otherwise

Media Events

A value that is set to 1 if an event should be emitted or 0 otherwise

gre.media.exit

The media back-end application has exited.

Data:

No data payload

gre.media.timeupdate

Emitted when the time has been updated.

Data: "4u1 time_elapsed 4u1 total_time 1s0 channel_name"

```
unsigned time_elapsed
unsigned total_time
char channel_name[MAX_CHANNEL_NAME_LEN + 1]
```

Where:

time_elapsed	The time that has elapsed during play back
total_time	The total time for play back
channel_name	The name of the channel that this time event occurred on

gre.media.statechange

Emitted when the player has changed state, between a paused and playing state.

Data: "1s33 channel_name 1s0 state"

```
char channel_name[MAX_CHANNEL_NAME_LEN + 1]
```

```
char state[1]
```

Where:

channel_name The name of the channel that is changing state

state The new state: “paused” | “playing”

gre.media.complete

Triggered when the named media has played to the end and stopped playing

Data: "1s33 channel 1s0 name"

```
char channel_name[MAX_CHANNEL_NAME_LEN + 1]
char media_name[1]
```

Where:

channel_name The name of the channel that has completed playback

media_name The name of the media stream that completed playback

gre.media.error

Triggered when there was an error playing the media source.

Data: "1s33 channel_name 1s0 error_msg"

```
char channel_name[MAX_CHANNEL_NAME_LEN + 1]
char error_msg[1]
```

Where:

channel_name The name of the channel that received an error

error_msg[1] The error message

Media backends

The media backend application does the work of playing and controlling the media based on requests from the Storyboard application over a Storyboard IO channel. The default Storyboard IO channel name is `com.crank.media_backend`. This value can be overwritten by setting the `SBMEDIA_CHANNEL_NAME` environment variable to a new value. Currently the only supported media backend is based on `gstreamer`.

gstreamer-backend

This media backend uses the `gstreamer` framework to play and control audio and video files. In order to use this backend the platform must have `gstreamer` and the required plugins installed. It is a good idea to try

and play content with the “gst-launch” application to ensure a proper installation before running gstreamer-backend. This backend also uses Storyboard IO for communication with the Storyboard application so please ensure Storyboard IO is functional and the application has the “greio” plugin loaded.

Options:

-e : Render the video content with an external buffer
-p pipeline: Use the defined gstreamer pipeline to play the media
-v: increase verbosity, debug output

Action data The “new.audio” and “new.video” actions take an extra_data argument. This argument is a string which can contain the following options which must be separated by a “;”.

Gstreamer pipeline

You can specify the gstreamer pipeline used to play the particular media by either passing it on the command line to gstreamer-backend with the “-p” option or by passing it to the actions. The pipeline can be passed in as:

“pipeline:[your pipeline]”

This pipeline can be similar to the one used with the “gst-launch” application with a few minor modifications. In order to allow the changing of the media file the first part of the pipeline must contain a named filesrc element as follows:

“pipeline:filesrc location=video.mov name=media-src”

Doing this will allow the code to find the named element and replace the location with a new video file.

External render extensions

If the content is to be rendered with an external render extension then you can pass the “-e” option or add an option to the extra_data argument as follows:

“use_external”

Chapter 5. Event Definitions

Standard Event Definitions

Storyboard supports a list of standard events. These events are all prefixed with `gre.` and can be used by your application.

System Events

`gre.init`

The system has been initialized and is ready. This is the first event set in the system.

Data:

No data payload

`gre.quit`

The system is being shutdown.

Data:

No data payload

`gre.redraw`

An area of the screen has been damaged (visible data has changed). A redraw event may not cause actual screen drawing if the control which has changed is hidden or offscreen.

Data:

```
int32_t    x
int32_t    y
int32_t    width
int32_t    height
```

If the values are all 0 then the entire screen has been damaged

Pointer Events

The following events are generated in response to a device such as a mouse or a touchscreen. These events are targeted at specific controls based upon the controls location and sensitivity.

`gre.press`

A mouse/touchscreen has been pressed.

Data

```

uint32_t    button
uint32_t    timestamp
int16_t     subtype
int16_t     x
int16_t     y
int16_t     z
int16_t     id
int16_t     spare

```

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.release

A mouse/touchscreen has been released.

Data

```

uint32_t    button
uint32_t    timestamp
int16_t     subtype
int16_t     x
int16_t     y
int16_t     z
int16_t     id
int16_t     spare

```

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information

id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.touch

If a mouse/touchscreen presses and then releases on the same control then a touch event will be generated. This is useful for activating button style elements. If the release is found to intersect a different control then a touch event is not generated.

Note

This event is synthetically generated by the framework based on incoming `gre.press` and `gre.release` events. Event redirectors should generally not include this event in their list of redirection events.

Data

uint32_t	button
uint32_t	timestamp
int16_t	subtype
int16_t	x
int16_t	y
int16_t	z
int16_t	id
int16_t	spare

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.mtpress

A touchscreen has been pressed. This event is emitted where are two or more contact points.

Data

uint32_t	button
uint32_t	timestamp
int16_t	subtype

```
int16_t    x
int16_t    y
int16_t    z
int16_t    id
int16_t    spare
```

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.mtrelease

A touchscreen has been released. This event is emitted when there are two or more contact points.

Data

```
uint32_t    button
uint32_t    timestamp
int16_t     subtype
int16_t     x
int16_t     y
int16_t     z
int16_t     id
int16_t     spare
```

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.inbound

A mouse/touchscreen has entered a control (if dragging a pointer or finger). This event is generated once the coordinates enter a control boundary. If mouse motion events are disabled in the render manager then this event will not be generated.

Note

Control groups can not receive inbound events.

Data

uint32_t	button
uint32_t	timestamp
int16_t	subtype
int16_t	x
int16_t	y
int16_t	z
int16_t	id
int16_t	spare

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

gre.outbound

A mouse/touchscreen has left a control (if dragging a pointer or finger). This event is generated once the coordinates leave a control boundary. If mouse motion events are disabled in the render manager then this event will not be generated.

Note

Control groups can not receive outbound events.

Data

uint32_t	button
uint32_t	timestamp
int16_t	subtype


```
int16_t    x
int16_t    y
int16_t    z
int16_t    id
int16_t    spare
```

Where:

button	GR_EVENT_BTN_LEFT - 0x0001: if this is a touchscreen then the button is always left GR_EVENT_BTN_MIDDLE - 0x0002 GR_EVENT_BTN_RIGHT - 0x0004
timestamp	This is an event timestamp in milliseconds since application start
subtype	GR_EVENT_RELEASE_IN GR_EVENT_RELEASE_OUT
z	This parameter is dependent on the availability of z- co-ordinate information
id	This parameter is used to track multi-touch presses as they come in
spare	This is padding and should be 0

Keyboard Events

The following events are generated if a keyboard is present and supported by the render manager.

gre.keydown

A key is in the pressed state

Data

```
uint32_t    code
uint32_t    modifiers
```

Where:

code	This is the UTF-8 key value
modifiers	A set of modifiers applied to the key GR_EVENT_KEYMOD_ALT GR_EVENT_KEYMOD_CTRL GR_EVENT_KEYMOD_SHIFT

gre.keyup

A key which was previously pressed has been released

Data

```
uint32_t    code
uint32_t    modifiers
```

Where:

```
code        This is the UTF-8 key value

modifiers    A set of modifiers applied to the key

              GR_EVENT_KEYMOD_ALT
              GR_EVENT_KEYMOD_CTRL
              GR_EVENT_KEYMOD_SHIFT
```

Screen Manager Events

The following events are generated by the Screen Manager during screen transitions. These events are generated in the following order:

```
gre.screenshow.pre delivered to target (end) screen
gre.screenhide.pre delivered to source (start) screen
gre.screenshow.post delivered to target (end) screen
gre.screenhide.post delivered to source (start) screen
```

gre.screenshow.pre

A screen is being shown. This event is triggered before the screen is shown and signifies that a transition may be starting

Data:

```
char *name    The name of the screen which is being shown
```

gre.screenshow.post

A screen has been shown. This event is triggered after the screen is shown and signifies that a transition has ended.

Data:

```
char *name    The name of the screen which has been shown
```

gre.screenhide.pre

A screen is being hidden. This event is triggered before the screen is hidden and signifies that a transition may be starting.

Data:

```
char *name    The name of the screen which is being hidden
```

gre.screenhide.post

A screen has been hidden. This event is triggered after the screen is hidden and signifies that a transition has ended.

Data:

char *name The name of the screen which has been hidden

Focus Events

The following events are generated on a change of control focus. If there is no focusable control on the current screen then these events will not be generated. These events are targeted at the currently or last focused control. When focus shifts from one control to another the lost focus event is sent first followed by the got focus event.

gre.gotfocus

A control has received focus, delivered to the control that received the focus.

No data payload.

gre.lostfocus

A control has lost focus, delivered to the control that has lost the focus.

No data payload.

Table Events

The following events are generated by a Table control. If no table control is present then these events will not be generated.

gre.table.viewport

A table has been resized via the table resize action. This event notifies the system of the new table size and visible area.

Data:

```
uint32_t    top_row
uint32_t    left_col
uint32_t    bot_row;
uint32_t    right_col;
char        *table
```

Where:

top_row	The top row that is visible
left_col	The left column that is visible
bot_row	The bottom row that is visible
right_col	The right column that is visible
table	The name of the table whose viewport changed to cause this event

gre.cell.gotfocus

A table cell has received focus and is the currently active cell. This is delivered to the control template with the *cell* focus information.

Data:

```
uint32_t    row;  
uint32_t    col;  
char *table
```

Where:

row The row that received focus

col The column that received focus

table The name of the table where the cell focus changed

gre.cell.lostfocus

A table cell has lost focus and is no longer the active cell. This is delivered to the control template with the *cell* focus information

Data:

```
uint32_t    row;  
uint32_t    col;  
char *table
```

Where:

row The row that received focus

col The column that received focus

table The name of the table where the cell focus changed

Table Scroll Events

The following events are only generated when the "Enable list scrolling behaviour" option is checked in the Table properties.

gre.table.drag_start

This event is generated when a user begins dragging a scrolling table.

Data:

No data payload

gre.table.drag_stop

This event is generated when a user stops dragging a scrolling table.

Data:

No data payload

gre.table.scroll_start

This event is generated when the scroll animation begins.

Data:

No data payload

gre.table.scroll_stop

This event is generated when the scroll animation completes.

Data:

No data payload

gre.table.scroll_cancel

This event is generated when the scroll animation is interrupted.

Data:

No data payload

Layer Scroll Events

The following events are only generated when the "Enable layer scrolling behaviour" option is checked in the Layer Scrolling properties.

gre.drag.start

This event is generated when a user begins dragging a scrolling layer.

Data:

The name of the object being scrolled

gre.drag.stop

This event is generated when a user stops dragging a scrolling layer.

Data:

The name of the object being scrolled

gre.scroll.start

This event is generated when the scroll animation begins.

Data:

The name of the object being scrolled

gre.scroll.stop

This event is generated when the scroll animation completes.

Data:

The name of the object being scrolled

gre.scroll.cancel

This event is generated when the scroll animation is interrupted.

Data:

The name of the object being scrolled

Mobile Events (Android and iOS)

The following events are only generated when running on Android and iOS.

gre.mobile.on_pause

The application has become inactive. The application will not be rendering to the screen after this event is received.

Data:

No data payload

gre.mobile.on_resume

The application has become active. The application will be rendering to the screen after this event is received.

Data:

No data payload

gre.mobile.on_background

The application has lost focus.

Data:

No data payload

Android Events

The following event is only generated when running on Android.

android.onBack

The back button on the Android application has been pressed.

Data:

No data payload

Windows Embedded Compact 2013 (WEC2013) Events

Limited gestures support has been added to the winevent plugin for the Windows Embedded Compact 2013 platform. This support has been added via the Storyboard Engine winevent plugin (libgre-plugin-winevent.dll) and this plugin must be included in Storyboard Engine distribution.

The gesture support is designed to make visible the internal Windows gesture events and payloads that are generated from the underlying system as described in this document: <https://msdn.microsoft.com/en-us/library/ee503599.aspx>

As of the Storyboard 4.2 release, only the GID_PAN and GID_SCROLL sub-category of WM_GESTURE gesture events are translated into corresponding Storyboard events. These events will only be generated on WEC2013 hardware platforms where the BSP has been configured with gesture event support and the touchscreen driver configured to enable such event generation.

In order to add application support for receiving these win.gesture events, the events must be added to the Storyboard Designer application. The events can be added in the same manner as any other user defined events are added at the point where they are used to trigger an action within the "New Action" dialog.

win.gesture.pinch

This event is generated in response to the Windows MW_GESTURE:GID_PAN event. The event data is taken directly from the Windows event.

Data: (4s1 x 4s1 y 4s1 spread)

```
int32_t x
int32_t y
int32_t spread
```

win.gesture.[up|down|left|right|unknown]

This event is generated in response to the Windows MW_GESTURE:GID_SCROLL event. The event data is taken directly from the Windows event.

Data: (4s1 velocity 4s1 angle)

```
int32_t velocity
int32_t angle
```

Plugin Specific Event Definitions

The following events are generated by optional Storyboard plugins.

gre.gesture.up

Data:

```
char *gesture_num
int32_t time
```

gre.gesture.down

Data:

```
char *gesture_num
int32_t time
```

gre.gesture.left

Data:

```
char *gesture_num
int32_t time
```

gre.gesture.right

Data:

```
char *gesture_num
int32_t time
```

gre.screendump.complete

A screen dump action has completed.

timer.[name] Timer Events

Timer events are generated as a result of a timer action. See documentation on the `gra.timer` action for further information about configuring timers.

The timer event name will be formatted as `timer.[name]` where `name` is the value set as the name of the timer when the action was defined.

gre.animate.complete.[name] Animation Events

Animation events are generated as a result of an animation action. See documentation on the `gra.animation` action for further information about configuring animations.

When an animation is completed, an animation event will be fired. The event name will be formatted as `gre.animate.complete.[name]` where `name` is the value set as the name of the animation when the action was defined.

`gre.rendermgr.error`

This event will be generated when an error has occurred with a OpenGL ES 2.0 runtime using the "error_event" option.

Format:

```
4s1 code 1s0 msg
```

`code`:The code is the error code that is returned from the GL framework on the API call `glGetError()`

`msg`:The msg is a human readable diagnostic message about the context of the error and any associated resources involved. For example: problems loading font or image resource textures will identify the image and font related to the error, other API calls will be identified by context of execution (ie GL function name, shader compilation).

Chapter 6. Action Definitions

Built-in Action Definitions

Storyboard supports a number of standard actions which are built-in to the framework. These actions are all prefixed with `gra.` and can be incorporated into your application design without any plugin dependency.

gra.screen

Cause a screen transition to occur by replacing the current screen with the new one.

The action arguments are:

`screen` The name of the screen to transition to.

gra.screen.fade

Causes a screen transition to occur by fading the old screen into the new one.

The action arguments are:

`screen` The name of the screen to transition to.

`rate` Defines how the alpha value will change over the transition:

`linear`
`easein`
`easeout`
`easeinout`
`bounce`

`fps` The frames per second to use for the transition

`duration` The duration of the transition in milliseconds

gra.screen.hold

Hold all screen updates. While held a screen will not redraw.

gra.screen.release

Release a held screen. If a screen was damaged during the period of time that the screen was being held, then a redraw action will be triggered.

gra.sendevent

Send an event to the application's input event queue. This action is equivalent to injecting an event via Storyboard IO or using the Lua `gre.send_event()` API

The action arguments are:

`event` The name of the event to send

gra.datachange

Change or create a variable value in the data manager.

The action arguments are key/value pairs such that the key is the fully qualified model path for the variable and the value is the new value to assign to that variable. For more information on creating the model paths, see the Data Variables section of this document.

gra.screen.focus.set

Set the focus to a specific control.

The action arguments are:

index The focus index to set the focus to.

control The name of a control to set the focus to.

Only one of either the index or the control need to be provided for this action. If both arguments are set, then the index value will be used.

In order for this action to complete successfully, the control specified must be set as focusable. For more information about making controls focusable and the focus operation in Storyboard, refer to the Focus section in the Execution Pipeline part of this document.

gra.screen.focus.next

Move the current focus to the next focusable control.

The action arguments are:

min The minimum focus index to move to or -1 to have no minimum value

max The maximum focus index to move to or -1 to have no maximum value

In order for this action to complete successfully, there must be a control whose focus index lies between the min and max values. For more information about making controls focusable and the focus operation in Storyboard, refer to the Focus section in the Execution Pipeline part of this document.

gra.screen.focus.prev

Move the current focus to the previous focusable control.

The action arguments are:

min The minimum focus index to move to or -1 to have no minimum value

max The maximum focus index to move to or -1 to have no maximum value

In order for this action to complete successfully, there must be a control whose focus index lies between the min and max values. For more information about making controls focusable and the focus operation in Storyboard, refer to the Focus section in the Execution Pipeline part of this document.

gra.screen.focus.direction

Move the current focus to the next control in a direction.

The action arguments are:

min	The minimum focus index to move to or -1 to have no minimum value
max	The maximum focus index to move to or -1 to have no maximum value
direction	The direction to search for the next focusable control <ul style="list-style-type: none"> up down left right

In order for this action to complete successfully, there must be a control whose focus index lies between the min and max values. For more information about making controls focusable and the focus operation in Storyboard, refer to the Focus section in the Execution Pipeline part of this document.

gra.table.scroll

Scroll the content of one or more tables.

The action arguments are:

control	The name of the table control to scroll. May be a comma separated list if multiple tables are specified.
row	The absolute 1 based row to start the scroll from. The default, 0, indicates that scrolling should start from the current row
col	The absolute 1 based column to start the scroll from. The default, 0, indicates that scrolling should start from the current column
delta_row	The number of rows to move. A positive value moves the table <i>down</i> a negative value moves the table <i>up</i>
delta_col	The number of columns to move. A positive value moves the table <i>right</i> a negative value moves the table <i>left</i>
fps	The frames per second rate at which to scroll the table.
duration	The duration in milliseconds to run the scroll over.

gra.table.resize

Set the number of rows and columns for a table. This action does not resize the control, simply the number of cells contained within the virtual table.

The action arguments are:

control	The name of the table control to resize.
rows	The number of rows for the table. Specifying 0 will leave the current number of rows unchanged.
columns	The number of columns for the table. Specifying 0 will leave the current number of columns unchanged.

When the table is resized, a `gre.table.resize` event will be emitted.

gra.table.navigate

Navigates the cells of a table, sets the active cell which in turn generates the cell focus events. If the new active cell is not visible the table will be scrolled in order to show this cell.

The action arguments are:

control	The name of the table control to scroll. May be a comma separated list if multiple tables are specified.
fps	The frames per second rate at which to scroll the table, 0 performs an immediate scroll.
duration	The duration in milliseconds to run the scroll over, 0 scrolls it immediately.
direction	The type of navigation to perform
set	Sets the active row and column to what is specified in the row/col parameters. The option only ensures that the cell is visible and does not guarantee the cell will be at the top of the visible list.
next	Move to the next cell, scroll by column then by row
prev	Move to the previous cell, scroll by column then by row
up	Move to the cell above the current one
down	Move to the cell below the current one
left	Move to the cell to the left of the current one
right	Move to the cell to the right of the current one
home	Move to the first cell in the table at row,column 1,1
end	Move to the last cell in the table
row	The row to navigate to. This is only used if <code>direction</code> is assigned <code>set</code>
col	The column to navigate to. This is only used if <code>direction</code> is assigned <code>set</code>

gra.log

Use the GRE logging mechanism to output a message.

The action argument is the string message that should be output.

gra.resource.dump_def

Remove a resource which is managed by the resource manager.

The action arguments are:

pool	The pool name containing the resource to dump
ref	The name of the resource to dump

The currently defined resource pools are `image` containing all of the images and `font` containing all of the fonts associated with the Storyboard application.

gra.playback

This is the action definition for Designer. For more information on adding action definitions to Storyboard Designer please refer to Section 2.6.6 User Defined Actions.

```
<actiontemplates>
  <template name="gra.playback">
    <arguments>
      <element name="filename" type="string" />
      <element name="loop" type="boolean" />
      <element name="quit" type="boolean" />
      <element name="verbose" type="boolean" />
    </arguments>
  </template>
</actiontemplates>
```

The action arguments are:

Filename	The name of the file to capture the events
Loop	True or false
Quit	Send a quit message when finished
Verbose	Set verbosity. More v's means more verbose output

Plugin Action Definitions

The following actions are only available when optional Storyboard plugins has been loaded.

gra.lua

Cause a Lua script function to execute.

Plugin	<code>libgre-plugin-lua.so</code>
Options:	<code>script</code> The name of the Lua function to invoke

Additional arguments can be passed to the function by providing additional key/value pairs to the action. The key/value pairs are provided to the Lua function as values in the argument table.

For example to call the Lua function `myfunction` with an extra argument, `firstargument`, that corresponds to the value of the application variable `myvar` you would simply add a new entry to the parameter list.

The corresponding call to the Lua function would fill the entry into the argument table such that:

```
function myfunction(mapargs)
  print("The value is: " .. tostring(mapargs.firstargument))
end
```

would print out the value of `${app:myvar}`.

gra.animate

Start an animation. Animations are started based on their name. Each animation can have an optional identifier (id) which is used to ensure that animations run in an exclusive manner. If an existing animation is running that uses the same identifier, then that animation is stopped before this animation is started. The data argument is as follows:

Plugin	<code>libgre-plugin-animation.so</code>
Options:	<div> <div>name</div> <div>The animation name to start</div> </div> <div> <div>id</div> <div>An optional instance id to be associated with the animation. Animation identifiers can be used with different animations to ensure that only one animation of the set is running at a time.</div> </div>

When the animation stops it will emit a notification event in the form of `gre.animate.complete`. `[name]`. This event will be delivered within the context of the `gra.animate` action and will be delivered to the object which invoked the action.

gra.animate.stop

Stop an animation. If you stop an animation only by name then all running animations with that name will stop and emit a complete event. If you stop an animation by id then only that specific animation will stop and emit a complete event. The data argument is as follows:

Plugin	<code>libgre-plugin-animation.so</code>
Options:	<div> <div>name</div> <div>The animation name to stop</div> </div> <div> <div>id</div> <div>An optional instance id associated with the name</div> </div>

When the animation stops it will emit a notification event in the form of `gre.animate.complete`. `[name]`

gra.audio

Start or stop the asynchronous playback of a WAV audio file. The data argument is as follows:

Plugin:	<code>libgre-plugin-audio.so</code>
Options:	<div> <div>filename</div> <div>A filename to play, or empty to stop the current playback.</div> </div>

gra.greio

Send a new event over a Storyboard IO channel.

Plugin:	<code>libgre-plugin-greio.so</code>
---------	-------------------------------------

Options:	name	Storyboard IO channel name to send the event to (required)
	event	The name of the event to generate (required)
	target	The target of the event to generate (optional)
	format	The format of the event data (optional)
	data	The data payload for the event

gra.perf_state

Control the capture of performance data

Plugin: `libgre-plugin-logger.so`

Options: state Turn off (0) or on (1) performance data capture

In order for this action to be used, the `libgre-plugin-logger.so` must have been started with performance logging enabled, but not necessarily to have it start capturing the performance data. For example `sbengine -ologger,perf=0` will enable performance logging but not start capturing events at startup while `sbengine -ologging,perf=1` will enable performance logging and immediately start capturing events.

gra.redirect

Redirect all events to another Storyboard IO channel.

Plugin: `libgre-plugin-redirect.so`

Options: channel Storyboard IO channel name to send the events to

gra.screen.path

Causes a screen transition to occur by fading the old screen into the new one.

Plugin: `libgre-plugin-screen-path.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear
		easein
		easeout
		easeinout
		bounce
	fps	Number of frames per second
	duration	Length of the transition in milliseconds
	direction	The direction to transition from
		left
		right

	top bottom
layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
	all delta
moving	The screen(s) to animate with the desired path transition.
	both new only old only

gra.screen.scale

Causes a screen transition to occur by scaling the old screen into the new one.

Plugin: `libgre-plugin-screen-scale.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear easein easeout easeinout bounce
	fps	Number of frames per second
	duration	Length of the transition in milliseconds
	layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
		all delta

gra.screen.gls witch

Causes a screen transition to occur by using 3D to switch the old screen into the new one.

Plugin: `libgre-plugin-screen-3d.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear easein easeout easeinout bounce

fps	Number of frames per second
duration	Length of the transition in milliseconds
layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
	all
	delta

gra.screen.glrotate

Causes a screen transition to occur by using 3D to rotate the old screen in the x-axis into the new one.

Plugin: `libgre-plugin-screen-3d.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear
		easein
		easeout
		easeinout
		bounce
	fps	Number of frames per second
	duration	Length of the transition in milliseconds
	direction	The direction to transition from
		left
		right
		top
		bottom
	layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
		all
		delta

gra.screen.gflip

Causes a screen transition to occur by using 3D to switch the old screen into the new one.

Plugin: `libgre-plugin-screen-3d.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear
		easein

	easeout easeinout bounce
fps	Number of frames per second
duration	Length of the transition in milliseconds
layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
	all delta

gra.screen.gldoors

Causes a screen transition to occur by using 3D to switch the old screen into the new one using a door opening animation.

Plugin: libgre-plugin-screen-3d.so

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear easein easeout easeinout bounce
	fps	Number of frames per second
	duration	Length of the transition in milliseconds
	direction	The direction to transition from
		left right top bottom
	layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
		all delta

gra.screen.gltip

Causes a screen transition to occur by using 3D to switch the old screen into the new one by tipping the display forward.

Plugin: libgre-plugin-screen-3d.so

Options: screen The screen to transition to

rate	Defines how the alpha values will change
	linear easein easeout easeinout bounce
fps	Number of frames per second
duration	Length of the transition in milliseconds
direction	The direction to transition from
	left right top bottom
layers	The layers to transition, all of the layers or just the layers that are different between source and destination.
	all delta

gra.screen.glcube

Causes a screen transition to occur by using 3D to switch the old screen into the new one using a cube animation.

Plugin: `libgre-plugin-screen-3d.so`

Options:	screen	The screen to transition to
	rate	Defines how the alpha values will change
		linear easein easeout easeinout bounce
	fps	Number of frames per second
	duration	Length of the transition in milliseconds
	direction	The direction to transition from
		left right top bottom
	layers	The layers to transition, all of the layers or just the layers that are different between source and destination.

all
delta

gra.screendump

Dump the contents of the screen to an image file.

Plugin: `libgre-plugin-screen-dump.so`

Options: `filename` The filename of the image file to create. The directory path to the filename must exist and the filename will be overwritten if it is. The filename must end with either a `.bmp` extension to generate BMP formatted images or `.tga` to generate TGA formatted images

gra.timer

Start, stop and control a timer.

Plugin: `libgre-plugin-timer.so`

Options: `name` The name to use to identify this timer (required)

`rtime` The time delay in milliseconds *relative* to the action invocation. Specify a value of 0 to stop an existing timer.

`repeat` The number of milliseconds to delay after the timer first fires, used to provide a stable repeat timer. Specify 0 for a one shot timer.

`count` The number of times that the timer should repeat before automatically stopping, assuming that the timer is not a one shot timer. Specify -1 to allow an unlimited number of repeat firings

`rtime` must be specified and a value of 0 for `rtime` and `repeat` indicates that the timer should stop firing.

For example, to start a timer that fires in 1s from the event and then every 500ms afterwards:

```
<action ... type="timer" data="name=MyTimer,rtime=1000,repeat=500" />
```

Then to stop the timer:

```
<action ... type="timer" data="name=MyTimer,rtime=0,repeat=0" />
```

An event will be generated each time that the timer fires and will be named `timer.<name>` so for the examples above, the event would be generated would be `timer.MyTimer`.

Chapter 7. Render Extension Definitions

Common Render Extension Options

The follow is a list of common options across all render extensions

- **x** - x position of the render extension relative to the control (number, optional, default: 0)
- **y** - y position of the render extension relative to the control s(number, optional, default: 0)
- **width** - width of the render extension, if -1 then it will set to the size of the object being rendered (number, optional, default: width of object it is attached to)
- **height** - height of the render extension, if -1 then it will set to the size of the object being rendered (number, optional, default: height of object it is attached to)
- **alpha** - Alpha value for this render extension (number, optional, 0-255, default: 255 (opaque))
- **rotation** - Rotation for the item, (0-360) (number, optional, default: 0)
- **valign** - Vertical alignment within the control (number, optional, default: 0)

0: default, top

1: top

2: center

3: bottom

- **halign** - Horizontal alignment within the control (number, optional, default: 0)

0: default, left

1: left

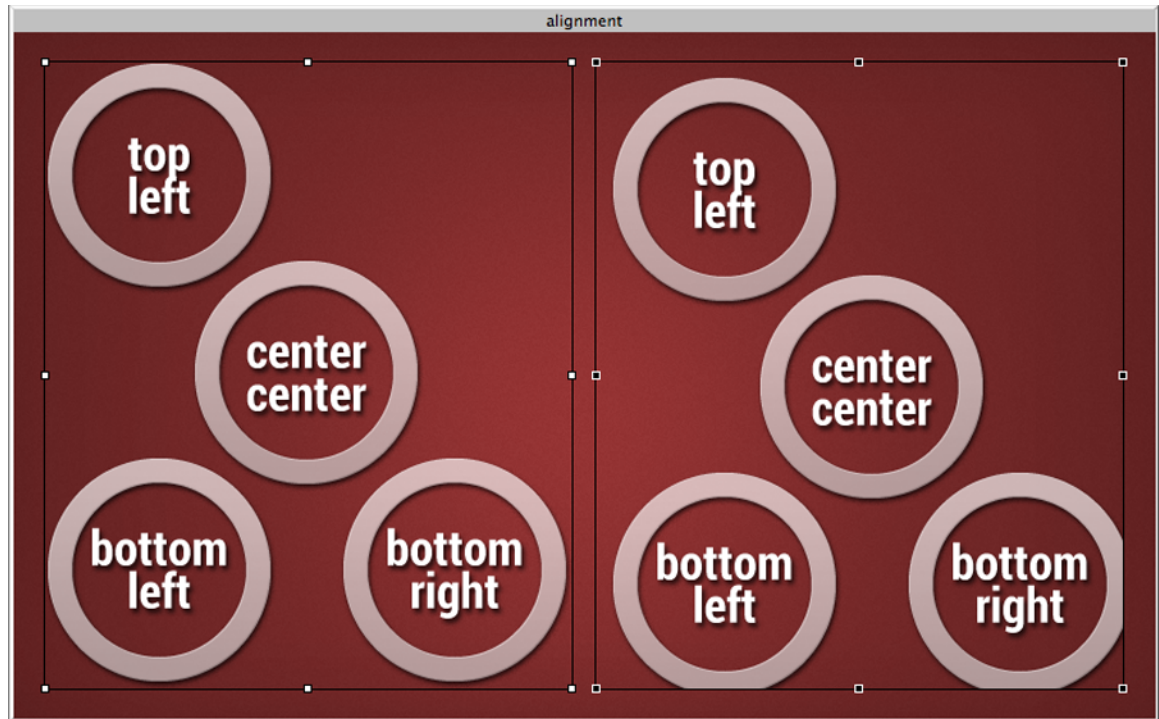
2: center

3: right

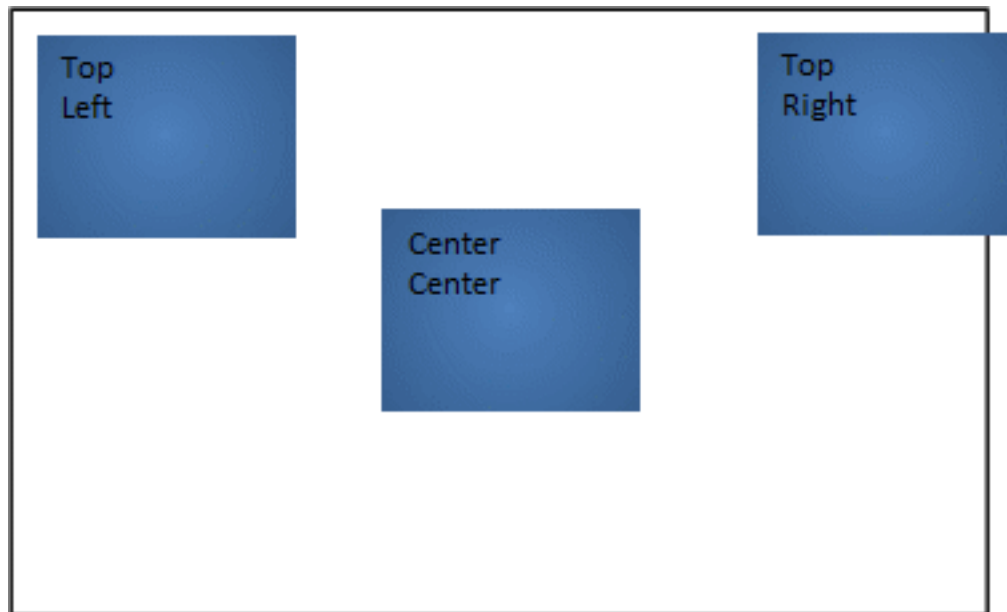
Render Extension Alignment

A render extension can have a vertical and horizontal alignment. This alignment is based on the control area and the render extension position and size. The following describes the effects of these parameters on alignment.

Width and height set, position set to (0,0)



Width and height set, position set to (5,5). The position functions as an offset to the alignment



Fill

The fill render extension draws a filled rectangle to the screen.

Fill Render Extension Options

- **color** - Color to use for item (number RGB format, optional)

Polygon

The polygon render extension draws a filled (convex) polygon to the screen. This extension is only available when the polygon plugin has been loaded.

Polygon Render Extension Options

- **style** - The style of polygon to render (string)
 - fill: draw a filled polygon using color attribute
 - line: draw a line using color attribute
 - lineloop: draw a line (with connected ends) in the using color attribute
 - filloutline: draw a filled polygon using color attribute, then outlined with outlinecolor attribute
- **antialias** - whether to antialias the polygon (number)
 - 0: do not antialias
 - 1: antialias

Single width outlined polygons are not anti-aliased on OpenGL ES 2.0 platforms. The multipass option for the OpenGL ES 2.0 Storyboard runtime controls polygon anti-aliasing.
- **points** - a list of points for the polygon (string)
 - List of points separated by a space. Each point looks like: x:y
- **color** - The color used to stroke the polygon. (color)
- **outlinecolor** - The color used to outline a filled polygon if the style is filloutline, this is currently a 1 pixel outline. (color)
- **linewidth** - Thickens with which to draw the outline

Rectangle

The rectangle render extension draws a single pixel outline rectangle to the screen.

Rectangle Render Extension Options

- **color** - Color to use for item (number RGB format, optional)

Image

The image render extension draws an image to the screen. Images can be scaled and tiled. When scaled, the image can also be set to maintain the aspect ratio.

Image Render Extension Options

- **name** - The name of the image to use (string, optional)

- **scale** - Enable scaling of the image if set to 1. The image will be loaded at full resolution and then scaled when rendered. If width and height are not set the image is scaled to the control size (number, optional)
- **loadscaled** - Enable scaling on load of the image if set to 1. This will load the image at the specified size and scale during the image decode. If width and height are not set the image is scaled to the control size (number, optional)
- **tile** - Tile the image, if width and height are not set the image is tiled to the control size (number, optional)
- **aspect** - If scaling maintain the images aspect ratio (number, optional)
- **Center Rotation** - If this value is turned on then any rotation applied to the image will happen around the center of the image and the values of **Center X**, **Center Y** will be ignored.
- **Center X, Y** - These values are only applied when the Center Rotation option is disabled and they specify the location of the rotation center point as a value relative to the control's upper left corner as 0,0 increasing as you go right and down. So to rotate around the center of a control whose width, height was 10, 20 you could specify a center point of X = 5 and Y = 10

Non-scaled, Scaled and Tiled



Image Alignment



Text

The text render extension draws a string to the screen. Strings can be wrapped on word boundaries and also rotated orthogonally. The following shows the effects of rotation on strings.

Text Render Extension Options

- **text** - The text string to display (string, optional)
- **font** - The font to use (string, optional)
- **size** - The point size of the string (number, optional)
- **underline** - Specifies if the string show display an underline

0: no underline

1: underline in text color

- **style** - Style to render in (number, optional)

0: Bitmap

1: Anti Aliased

- **wrap** - wrap text string to fit within render extension width

0: no text wrapping

1: text wrapping

External

The external render extension creates a buffer for other system applications or tasks to render into, things such as video players and web browsers. This extension is only available when the external plugin has been loaded.

External Render Extension Options

- **name** - The name of the external render extension. This information should be provided by the external render extension application provider and is used to allow the application to send update messages to the Storyboard Engine.
- **object** This is the path to a shared memory object which is created by the external application and is loaded by the Storyboard Engine. This information should be provided by the external render extension application provider.

3D Model

The 3d model render extension renders a 3d model into the control. Currently models in Wavefront Object (.obj) format are supported. This extension is only available when using OpenGL or OpenGL ES 2.0 based render managers, and requires the model3d plugin to be loaded.

The coordinate system in the render extension is the default OpenGL default coordinate system, with positive x to the right, positive y up, and positive z towards the viewer. The camera position defaults to (0, 0, 0), with the view direction along the negative z axis.

A Phong reflection model is implemented. A directional light source is present with white light coming from the (0, 1, 1) direction. The Phong model makes use of three terms:

- **Ambient** - The color of the material in the absence of direct light. The material will never appear darker than the ambient color.
- **Diffuse** - The color of light reflected from the material.
- **Specular** - The color of the highlights from the material. The specular exponent controls how large the highlight is.

For more details on the Phong reflection model refer to Phong Reflection Model [http://en.wikipedia.org/wiki/Phong_reflection_model] or to any book on computer graphics.

Rotations for the model are defined using Euler angles, with rotations applied around the z (psi), y (theta) and then x (phi) axes.

An OBJ file defines vertices and faces, and optionally normals and texture coordinates. If normals are not present, they will be calculated according to the convention that vertices in a face are specified in counter-clockwise order. If texture coordinates are not present, the model will not be rendered using a texture. Faces may be grouped together, and each group may be rendered with a different material.

Each OBJ file may also specify a Material (.mtl) file which allows for the material properties of the model to be specified. The following properties in a material file are currently supported:

- **d** - The transparency (alpha) of the material.
- **Ka** - The ambient lighting component of the material.
- **Kd** - The diffuse lighting component of the material.
- **Ks** - The specular lighting component of the material.
- **Ns** - The specular lighting exponent of the material.
- **map_Kd** - The texture specifying the diffuse color of the material. If the texture can be loaded, it will be used rather than the Kd parameter to when calculating the diffuse color.

If a material file is not present, the object will be rendered with a white color.

3D Model Render Extension Options

- **filename** - The name of the model to load.
- **camera_position_x** - The x position of the camera.
- **camera_position_y** - The y position of the camera.
- **camera_position_z** - The z position of the model.
- **azimuth** - The rotation of the camera around the y axis in degrees.
- **elevation** - The rotation of the camera around the x axis in degrees.
- **camera_field_of_view** - The field of view the camera in degrees. The field of view specifies how much of visual sphere is mapped to the control. A larger field of view is equivalent to using a wide-angle lens on a camera, and a smaller field of view is equivalent to using a zoom lens.
- **model_position_x** - The x position of the model.
- **model_position_y** - The y position of the model.

- **model_position_z** - The z position of the model.
- **model_orientation_phi** - The rotation of the model around the x axis in degrees.
- **model_orientation_theta** - The rotation of the model around the y axis in degrees.
- **model_orientation_psi** - The rotation of the model around the z axis in degrees.

Chapter 8. Scripting with Lua

The Storyboard Lua API (Lua API) gives developers access to the Engine through a Lua scripting interface. This API is a library of functions which allow interaction with the Engine by manipulating data and working with events and user interface components. Through the Storyboard Lua API developers can:

- Get and set data values from the model
- Inject application events
- Manipulate model objects such as controls/layers

The Storyboard Lua plugin is built on top of the standard 5.1 release of Lua available from www.lua.org. While the core Lua interpreter is unchanged from the standard release, two additional modules have been incorporated to facilitate development with Storyboard:

The bitwise manipulation module (bit32) from Lua 5.2 has been built-in to this Lua plugin. This module provides a native implementation of several standard bit operations, including those required for text conversion to/from UTF-8. The documentation for the bitwise functions available from this module can be found in the Lua 5.2 Reference Manual [<http://www.lua.org/manual/5.2/manual.html#6.7>]

The Storyboard module (gre) is included that provides function extensions to manipulate and work with the currently active Storyboard model. This module also incorporates the Storyboard IO communication API that can be used to send events to external programs.

Lua Function Parameters

Each Lua function invoked from the Storyboard is passed two arguments:

```
script_function_name( table mapargs, string allargs)
```

The first argument, mapargs, is a Lua table whose keys provide the context in which the action is being invoked along with any action specific argument and parameters. This context includes the application, screen and control the action was associated with, the currently focused control, any arguments provided to the action as well as all of the event data that cause the action to fire.

The following keys are always available inside the context's table:

context_app

The application context of the current action

context_screen

The screen context of the current action (the current screen)

context_layer

The layer context of the current action (the current layer)

context_control

The control context of the current action (the current control)

context_group

The control context of the current action (the current group)

context_row

If the context_control is a Table then this is the row index of the current cell

context_col

If the context_control is a Table then this is the column index of the current cell

context_target

The current context (app, screen, layer, or control) that the event was targeted at

context_event

The name of the event the triggered the action

context_event_data

A pointer to a Lua table containing any event data. The event data is different for each event and is defined in the event definition.

The second argument, allargs, provides a string containing the exact same string that was provided to the data arguments of the action.

Example of using context data:

```
function get_context( mapargs )
    print("triggered by event : "..mapargs.context_event)
    print("event was targeting : "..mapargs[mapargs.context_target])
end
```

Passing Extra Parameters to Functions

Lua actions are identified using an action type of Lua and setting the specific Lua function and extra parameters (if required) in the action arguments. Any extra parameters will be transferred directly to the Lua function through first argument (a Lua table) and the data can be accessed by using the parameter name as the table index.

```
function my_lua_func (mapargs , allargs )
    local p = mapargs.paramter1
    print("my_lua_func was passed : ".. tostring(p))
end
```

Storyboard Lua Integration

Since Lua has a tight integration with the Storyboard, there are some additional functions that have been added to facilitate access directly to data in the Storyboard model without having to use the Storyboard IO communication channel to access data and generate events.

The complete Lua API can be found in the Storyboard Lua API section of this document.

Lua Execution Environment

The Storyboard Engine Lua plugin provides a slightly different execution environment when compared to normal Lua script execution.

Normally a single Lua script serves as the starting point of script execution and all other scripts would be included using the `Lua require()` declaration. The Storyboard Lua plugin provides a slightly different loading behaviour in that it will pre-load all of the Lua scripts contained in the **scripts** directory at engine initialization time. The load ordering can be controlled by using the `require` statement to explicitly order dependencies. Since the `require` mechanism is used to perform the loading, any project files that use the same names as built-in Lua modules (ie `table.lua`, `string.lua` or `io.lua`) will generate a load time warning indicating the potential load time resource collision.

A side effect of this early module loading and execution is that any Lua script that is located outside of function blocks will have the opportunity to run before the application is fully initialized. This can be used to seed early execution environments or load preferences before the UI is in place and ready to render. Alternatively, this early initialization is possible by binding a callback to the `gre.init` event.

In addition to loading all of the script files in the **scripts** directory, the Lua plugin modifies the **package.path** variable and `;;` default search path to automatically search the scripts directory.

A convenience variable, **gre.SCRIPT_ROOT** is pushed into the execution environment that contains the path from the current working directory to the scripts directory. This variable can be used to locate additional resource files or to include extra script directories in a manner that is relative to the overall deployment bundle.

```
print("Script base directory: " .. tostring(gre.SCRIPT_ROOT))
-- Look for additional module files in the scripts/modules directory
package.path = package.path .. ";" .. gre.SCRIPT_ROOT .. "/modules/?.lua"
```

Asynchronous Lua Support

The asynchronous Lua support is provided in two fashions:

Lua Action: To create an independent Lua thread in response to an action, the user can add an 'async' parameter to any existing Lua script and it will automatically create and run that action outside of the main UI thread.

Lua Script There is a new Lua API call `gre.thread_create()` that takes a single parameter which is a function to execute. This function will be executed and scheduled to run in an independent thread of execution.

Threads are created using the system's underlying native operating system thread support. Operations are synchronized explicitly through locks in the Lua VM, however there is no explicit support for data synchronization (ie mutexes, condition variables)

The suggested communication pattern for inter-thread communication is to use Storyboard IO to inject event data into the system. This is similar to the idea behind `LuaLanes` or `WebWorkers` where inter-task communication is based on message passing.

Threads will be hard-terminated at exit. Clients should establish their own protocol where a soft shutdown is required to trigger any asynchronous threads to terminate.

Lua Debugger

Introduction

The Storyboard Lua Debugger enables the developer to monitor the flow of execution of the Lua scripts used by the Storyboard application. Using the debugger it is possible to step line by line through a Lua script while examining the variable values that are being used by the Lua functions.

Note

The Lua debugger is configured such that it can only be used with the simulator runtimes on the host platforms that support Storyboard Designer. For assistance in configuring the debugger for embedded targets, contact Crank Software support (support@cranksoftware.com).

Configuration

The Storyboard Lua debugger is only available with Storyboard Suite versions 1.1 and newer.

The Designer debugging environment communicates with the application's Lua script plugin using network sockets in a client/server model. The Storyboard application acts as the client and is controlled by the Designer debug environment which acts as the server.

In version 4.0, creating and launching the debug server is an automated process. As such, to configure Lua debugging it suffices to take the following steps:

1. Create a Storyboard application launch configuration
2. Include the Storyboard Lua plugin and enable debugging
3. Launch the Storyboard application

The first two steps are part of a one time configuration required to set-up the application and the debugger. After the initial set-up, only the last step needs to be performed in order to launch an application with the Lua debugger running.

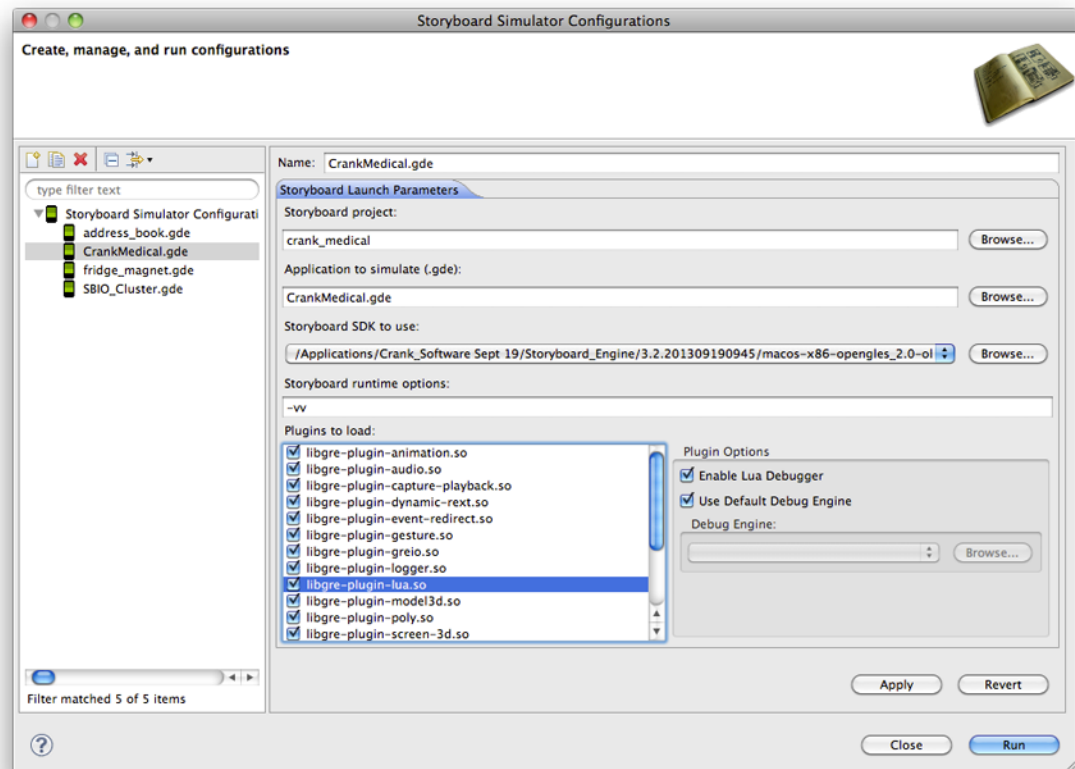
Create a Storyboard Launch Configuration

A Storyboard Launch configuration is automatically created when a Storyboard application is simulated via Run > Simulate Storyboard Project or by right clicking on the GDE file and selecting Storyboard Simulator.

In order to create a new launch configuration, or to customize an existing one, we need to open the Launch Configuration dialog. This can be done by selecting Run > Storyboard Simulator Configurations from the main menu. This will open a dialog where you can create, duplicate or delete simulator launch configurations. You can also open the Launch Configuration dialog via the icon on the toolbar.

Enable Lua Debugging

From the launch configuration dialog select the configuration that is to be used for the debug session. In the plugin listing ensure that the Lua plugin is selected and that the Enable Lua Debugger option has been selected.



Launch the Storyboard Application

With the Storyboard launch configuration created and the Lua plugin and debug mode enabled, you can now run the Storyboard application `Run > Storyboard Simulator Configurations ... > [Your Config]`.

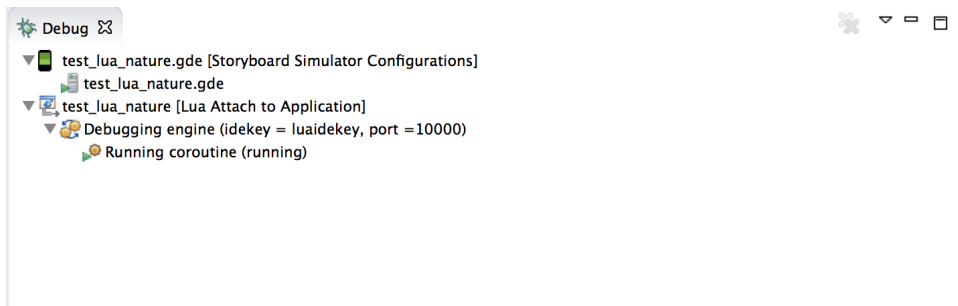
This will launch your application and an appropriately configured Lua Attach to Application debug configuration. The two will connect automatically and your application will immediately begin running. To confirm that the connection has taken place and the Lua debugger is running, check the output console for the following:

Debugger v1|.1.0

Debugger: Trying to connect to 127.0.0.1:10000 ...

Debugger: Connection succeeded.

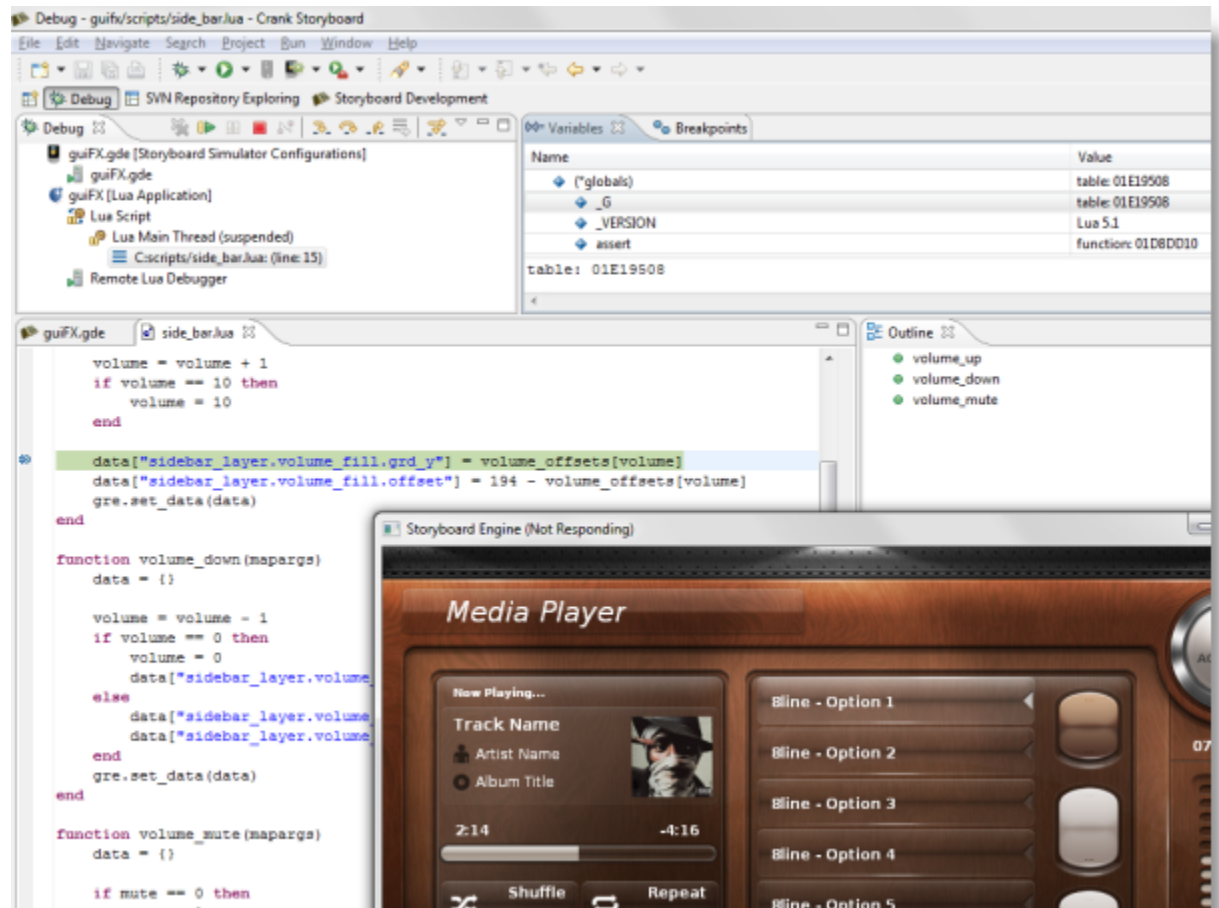
If you change to the Debugger perspective, the Debug view should look like this:



You are now able to start debugging your Lua code.

Debugging

The Lua debugger works like a traditional debugger. The Debugger perspective provides an alternative layout of views related to debugging activities.

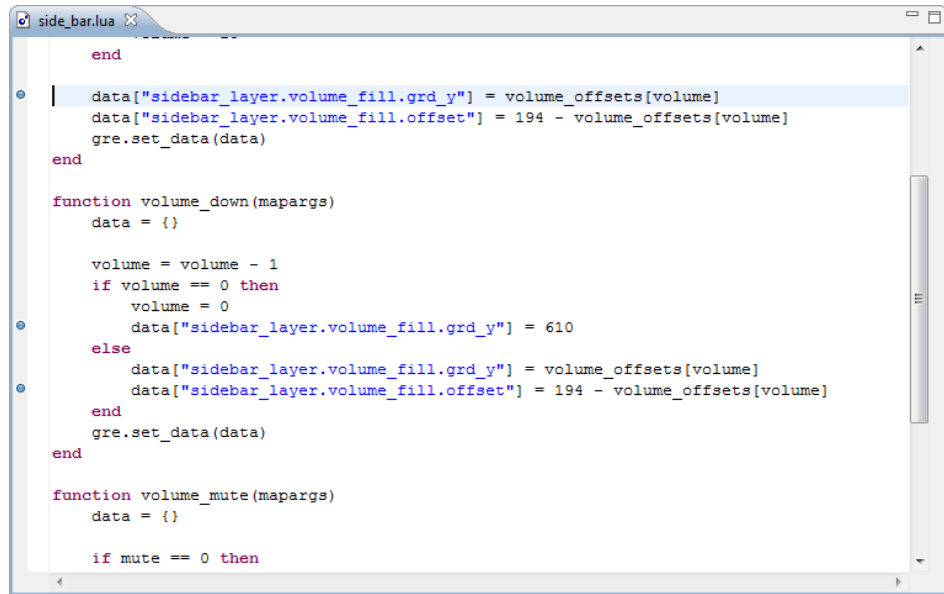


Breakpoints are listed in the Breakpoints view, local variables that are in scope for the selected frame are shown in the Variables view. The current stack trace and execution control (step, continue) are provided in the Debug view.

Breakpoints

Breakpoints can be placed directly in the editor for Lua script files. Breakpoints can be toggled on/off by double clicking in the margins of the Lua script file where the execution should be stopped. While it is possible to place breakpoints on all lines of a Lua script file, not all lines are breakable due to the manner in which the Lua script is executed. Declaration breakpoints may not resolve to an execution stop point in the script.

Breakpoints can also be enabled and disabled and removed by selecting them from the listing in the Breakpoints view and performing the appropriate operation. It is also possible to navigate to directly to the script source file from within the Breakpoints view by right clicking and selecting Go To File.

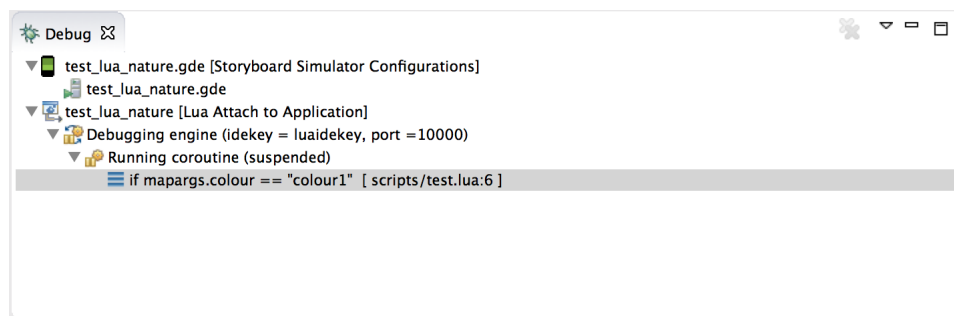


Variables

Variables are displayed in a hierarchical manner in the Variables view. Global variables are displayed in a special table and listed as `globals` while function parameters and local variables are displayed as top level elements. Strings and numeric values are displayed directly, while tables can be navigated by double clicking their nodes and driving down.

Stepping, Continuing and Terminating

The Debug view provides a list of active debug sessions and the execution stack trace when a session is at a breakpoint. Once a breakpoint is hit, then it is possible to single step to the next line of code, to continue the execution until the next breakpoint is encountered or to terminate the application all together using the view's toolbar commands.



Storyboard Lua API

gre.SCRIPT_ROOT

```
gre.SCRIPT_ROOT
```

This is a variable that is filled in by the Storyboard Engine to contain the value of the project's `scripts` directory.

The `gre.SCRIPT_ROOT` variable provides a convenient way to reference the file resources in a location independent, project relative, fashion. When used in conjunction with the `gre.env()` function, the `gre.SCRIPT_ROOT` can provide an effective way to configure the search path for extra Lua modules.

Example:

```
local data_file = gre.SCRIPT_ROOT .. "/input_data.csv"
```

gre.set_data

```
gre.set_data(  
    table  
)
```

Sets one or more items in the Storyboard application's data manager. Each index and value in the table passed in will be set in the data manager using the index's name as the key.

Parameters:

table A table containing the variable to change as the key and the value to change it to as that key's value.

Example:

```
function lua_func( mapargs )  
    local data_table = {}  
    data_table["variable_name"] = "variable data"  
    gre.set_data( data_table )  
end
```

gre.get_data

```
gre.get_data(  
    key  
    [, key2, ...]  
)
```

Gets one or more values from the data manager. Each argument to the function is interpreted as a data manager key whose value should be extracted from the data manager. This function returns a table using all the values as indexes and the corresponding value is the data returned from the data manager. A nil is returned for any values that do not match a key in the data manager.

Parameters:

key The key whose value should be extracted from the data manager.

Returns:

A table containing the passed in arguments as keys and the resulting data manager values as the values associated with those keys.

Example - Accessing Control Variables:

```
function get_data_fuc( mapargs )
  --When accessing control variables, use the following qualified
  --model path Layer.Control.Variable
  local data_table = gre.get_data("my_layer.my_control.variable_name")
  local value = data_table["my_layer.my_control.variable_name"]
  print("control_variable_name = " .. tostring(value))
end
```

Example - Accessing Control Width (Internal Variable):

```
function get_control_width( mapargs )
  --This will extract the width (grd_width) of the contro
  --'my_control' on the layer 'my_layer'
  local data = gre.get_data("my_layer.my_control.grd_width")
  local value = data["my_layer.my_control.grd_width"]
  print("The width of the control is " .. tostring(value))
end
```

gre.set_value

```
gre.set_value(
  key,
  value
  [, key2, value2, ...]
)
```

Set a variable in the data manager to a particular value. This function is a convenience function on top of `gre.set_data` that allows the key and value to be passed as a set of arguments to the function instead of having to create a table containing the key/value pairs.

Parameters:

key A string value containing the key to be set with the next following value
value The value to be assigned to the preceding argument (key)

Example:

```
function lua_func( mapargs )
    -- Assign the string 'variable_data' to the application variable
    -- 'variable_name'
    -- This example is the same as gre.set_data()
    gre.set_value("variable_name", "variable_data")
end
```

gre.get_value

```
gre.get_value(
    key
    [, key2, ...]
)
```

Get the value of a variable from the data manager. This function is a convenience function on top of `gre.get_data` that allows the value to be returned directly to the caller instead of a single table return value. A nil is returned for any values that do not match a key in the data manager.

Parameters:

key The key whose value should be extracted from the data manager.

Returns:

The value associated with the data manager entry for the key, or nil if no entry exists. If multiple keys are specified, then multiple return values will be generated matching the argument order.

Example - Accessing Control Width:

```
function get_control_width( mapargs )
    -- This will extract the width of the control 'my_control'
    -- on the layer 'my_layer'
    -- This is the same example as gre.get_data()
    local value = gre.get_value("my_layer.my_control.grd_width")
    print("The width of the control is " .. tostring(value))
end
```

gre.send_event

```
gre.send_event(  
    event_name,  
    [channel]  
)
```

Send an event to the application or to a Storyboard IO channel. `channel` is an optional parameter and if `channel` is not passed then the channel will be chosen as follows:

If the environment variable `GREIONAME` is set then it will be used otherwise the default channel is used.

Parameters:

`event_name` A string containing the event to send
`channel` An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global `GREIONAME` variable are set.

Example:

```
-- Send to the event to the application :  
gre.send_event("my_event")  
  
--To send the event to a Storyboard IO channel via parameters:  
gre.send_event("my_event", "io_channel_name")
```

gre.send_event_target

```
gre.send_event_target(  
    event_name,  
    target,  
    [channel]  
)
```

Send an event to a targeted model element (control, layer instance or screen) using the model's fully qualified path. The channel is an optional parameter.

Parameters:

`event_name` A string containing the event to send
`target` A string containing the object to target the event to (see Storyboard IO)
`channel` An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global `GREIONAME` variable are set.

Example:

```
-- Send to the event directed at a particular control target:
gre.send_event("my_event", "my_layer.my_control")
```

gre.send_event_data

```
gre.send_event_data (
    event_name,
    format_string,
    data,
    [channel]
)
```

Send an event with custom data to the application or to a Storyboard IO channel. The data parameter is a Lua table where the indexes match the values from the format string. channel is an optional parameter.

Special consideration is required for sending data that is to be formatted as an array (ie N[suf]M where M is greater than 0). In this case the data entry should be provided as a Lua table and not as a raw value parameter.

Parameters:

event_name A string containing the event to send
format_string A string format of the event data payload
data A table whose keys match up with the keys specified in the format_string
channel An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global GREIONAME variable are set.

Example:

```
-- Send a 'int32_update' event with a 32bit signed integer (int32_t)
-- payload to the 'controller' channel
function send_integer(value)
    local format = "4s1 value"
    local data = {}
    data["value"] = value
    gre.send_event_data("int32_update", format, data, "controller")
end

-- Send a 'int16_update' event with two 16bit signed integers (int16_t)
-- payload to the 'controller' channel
function send_two_integers(value1, value2)
    local format = "2s1 first 2s1 second"
    local data = {}
    data["first"] = value1
    data["second"] = value2
    gre.send_event_data("int16_update", format, data, "controller")
end
```



```
-- Send an 'array_update' event with an array of int32_t numbers (provided
-- as a table) to the client
function send_integer_array(values)
    -- Generate the format string dynamically based on the number of entries
    local count = #values
    local format = string.format("%s%d values", count)
    local data = {}
    data["values"] = values
    gre.send_event_data("array_update", format, data, "controller")
end

send_integer(12)
send_two_integers(10, 20)
send_integer_array({10, 20, 30, 40})
```

gre.receive_event

```
gre.receive_event(
    channel
)
```

Receive an event from a Storyboard IO channel. This is a blocking call and works best when used on a separate Lua thread.

Parameters:

channel A Storyboard IO channel to receive the event on.

Returns:

event A table containing the name, target, format and a data table from a received event.

Example:

```
-- Receive a Storyboard IO event with data payload x, y, z:
ev = gre.receive_event("my_channel")

if ev ~= nil then
    print(ev.name)

    for k,v in pairs(ev.data) do
        print(tostring(k).. " " ..tostring(v))
    end
end

--To disconnect from my_channel:
gre.greio_disconnect("my_channel", true)
```

gre.greio_disconnect

```
gre.greio_disconnect(  
    channel,  
    [is_receive_channel]  
)
```

This function forces any cached Storyboard IO channel connections to the specified channel to be closed. Subsequent calls using the same channel name will re-establish the connection to the channel if required.

Parameters:

`channel` The channel that is to be disconnected.
`is_receiving` An optional boolean parameter.
 -**True** if closing a receiving channel.
 -**False** or no argument if closing a sending channel.

Example:

```
-- Send an event to a custom channel  
gre.send_event("StoryboardRocks", "my_channel")  
-- Close the cached connection to that channel  
gre.greio_disconnect("my_channel")
```

gre.touch

```
gre.touch(  
    x ,  
    y ,  
    [channel]  
)
```

Send a touch event to the application at the co-ordinates passed in through the parameters. `channel` is an optional parameter

Parameters:

`x` The x position to simulate the touch event at
`y` The y position to simulate the touch event at
`channel` An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global GREIONAME variable are set.

Example:

gre.key_up

```
gre.key_up(  
    code,  
    [channel]  
)
```

Send a key_up event to the application with the scancode passed in the parameters. channel is an optional parameter

Parameters:

code The utf-8 character code to inject
channel An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global GREIONAME variable are set.

Example:

gre.key_down

```
gre.key_down(  
    code,  
    [channel]  
)
```

Send a key_down event to the application with the scancode passed in the parameters. channel is an optional parameter

Parameters:

code The utf-8 character code to inject
channel An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global GREIONAME variable are set.

Example:

gre.key_repeat

```
gre.key_repeat(  
    code,  
    [channel]  
)
```

Send a `key_repeat` event to the application with the `scancode` passed in the parameters. `channel` is an optional parameter

Parameters:

`code` The utf-8 character code to inject
`channel` An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global `GREIONAME` variable are set.

Example:

gre.redraw

```
gre.redraw(  
    x,  
    y,  
    width,  
    height,  
    [channel]  
)
```

Force a screen redraw. `channel` is an optional parameter. Specifying a `x,y,width,height` of 0 will result in a full screen refresh occurring.

Parameters:

`x` The x position of the redraw bounding box event
`y` The y position of the redraw bounding box event
`width` The width position of the redraw bounding box event
`height` The height position of the redraw bounding box event
`channel` An optional Storyboard IO channel to send the event on, if not specified the event is added directly into the current Storyboard application event queue if neither the environment variable or global `GREIONAME` variable are set.

Example:

gre.quit

```
gre.quit(  
    [channel]  
)
```

Send `QUIT` event to application to force shutdown. `channel` is an optional parameter.

Parameters:

`channel` An optional Storyboard IO channel to send the event on, if not specified the

event is added directly into the current Storyboard application event queue if neither the environment variable or global GREIONAME variable are set.

Example:

```
-- Send a quit message to the application
gre.quit()
```

gre.move_layer

```
gre.move_layer(
    layer_name,
    dx,
    dy,
    x,
    y
)
```

Move a layer to a new position. The `layer_name` is the name of the layer or a variable that is associated with the layer name. Setting `dx` or `dy` will move the layer by the specified delta from its current position. The `dx` and `dy` values can be 0 to set an absolute position using the `x` and `y` values only.

Parameters:

<code>layer_name</code>	The model full path of the layer to move
<code>dx</code>	A delta from the current x position or 0 to move using x
<code>dy</code>	A delta from the current y position or 0 to move using y
<code>x</code>	The x position to move to in absolute co-ordinates
<code>y</code>	The y position to move to in absolute co-ordinates

Example:

gre.move_control

```
gre.move_control(
    control_name,
    dx,
    dy,
    x,
    y
)
```

Move a controls to a new position. The `control_name` is the name of the control or a variable. Setting `dx` and or `dy` will move the layer by the specified delta from its current position. The `dx` and `dy` values can be 0 to set an absolute position using the `x` and `y` values only.

Parameters:

<code>control_name</code>	The model full path of the control to move
<code>dx</code>	A delta from the current x position or 0 to move using x
<code>dy</code>	A delta from the current y position or 0 to move using y
<code>x</code>	The x position to move to in absolute co-ordinates
<code>y</code>	The y position to move to in absolute co-ordinates

Example:

gre.clone_control

```
gre.clone_control(  
    reference_control_name,  
    new_control_name,  
    layer_name,  
    data  
)
```

Create a new control (`new_control_name`) on an existing layer (`layer_name`) by copying all of the properties of an existing control (`reference_control_name`). This new control will have all of the same actions, variables and it's current state will match the state of the reference control that is being copied.

The data argument is a table of control attributes that match the attributes available in the `gre.set_control_attrs()` function, for example `x`, `y`, `width`, `height`, `hidden`

Parameters:

<code>reference_control_name</code>	The name of the control that will be cloned. This may be a fully qualified name.
<code>new_control_name</code>	The name for the new control, this must be a unique name
<code>layer_name</code>	The name of the layer to place this control on, this layer must exist
<code>data</code>	Optional: A table containing control attribute tags as the keys with new values to be applied.

Example:

```
function create_new_control()  
    local data = {}  
    data["x"] = 10  
    data["y"] = 10  
    gre.clone_control("my_control", "my_new_control", "my_layer", data)  
end
```

gre.delete_control

```
gre.delete_control(  

```

```
        control_name,  
    )
```

Delete a control from the model. The control must be a control which was dynamically created with the `gre.clone_control()` function.

Parameters:

`control_name` The name of the control to delete

Example:

```
function delete_control()  
    gre.delete_control("my_control")  
end
```

gre.get_control_attrs

```
gre.get_control_attrs(  
    control_name  
    tags ...  
)
```

Get attributes for a control. Key name is the name of the control or a variable. Tags can be a list of the following values:

`x, y, width, height, hidden, active, zindex, findex`

A table with the results is returned.

Parameters:

`control_name` The model full path of the control to get information about
`tags` One or more tags as strings

Returns:

A table containing the tags as keys with the associated table value being the Storyboard value associated with that tag.

Example:

```
function check_if_hidden()  
    local dk_data = {}  
    -- check if my_control is currently hidden
```

```
dk_data = gre.get_control_attrs("my_control", "hidden")
if dk_data["hidden"] == 1 then
    print("my_control is currently hidden")
else
    print("my_control is currently visible")
end
end
```

gre.set_control_attrs

```
gre.set_control_attrs(
    control_name,
    tag_table
)
```

Set attributes for a control. The `control_name` is the name of the control or a variable. The `tag_table` contains the tags and values for the attributes to set.

x, y, width, height, hidden, active, zindex, findex, effect

In the case of the focus index (`findex`), the initial value set in Storyboard Designer must be non-zero in order for it to be changed dynamically at runtime

Parameters:

`control_name` The model full path of the control to change attributes on
`tag_table` A table with tags as the keys and the new values stored as the table's key values

Examples:

```
function set_control_hidden()
    local dk_data = {}
    dk_data["hidden"] = 1
    gre.set_control_attrs("my_control", dk_data)
end
```

```
function set_control_blur_effect()
    local dk_data = {}
    local effect = {}

    effect["name"] = "blur"
    effect["passes"] = 3
    effect["radius"] = 1
    effect["composite"] = true

    dk_data["effect"] = effect
```



```
gre.set_control_attrs("my_control", dk_data)
end
```

gre.get_layer_attrs

```
gre.get_layer_attrs(
    layer_name
    tags...
)
```

Get attributes for a layer instance associated with a particular screen. The `layer_name` specifies either the fully qualified name of a layer instance using the `ScreenName.LayerName` naming convention or, if only the layer name is specified, the name will refer to a layer instance associated with the current screen

The tags are a list of string attributes associated with the layer instance and can include one or more of the following values:

x, y, alpha, hidden, active, zindex, xoffset, yoffset

A table containing the keys and their respective values is returned or nil if the layer can not be found.

Parameters:

`layer_name` The model full path of the layer to get information about
`tags` One or more tags as strings

Returns:

A table containing the tags as keys with the associated table value being the Storyboard value associated with that tag.

Example:

```
function check_if_hidden()
    -- check if my_layer is currently hidden
    local data = gre.get_layer_attrs("my_layer", "hidden")
    if data.hidden == 1 then
        print("my_layer is currently hidden")
    else
        print("my_layer is currently visible")
    end
end
```

gre.set_layer_attrs

```
gre.set_layer_attrs(  
    layer_name,  
    tag_table  
)
```

Set attributes for a layer instance associated with a particular screen. The `layer_name` specifies either the fully qualified name of a layer instance using the `ScreenName.LayerName` naming convention or, if only the layer name is specified, the name will refer to a layer instance associated with the current screen

alpha, hidden, active, x, y, zindex, width, height, xoffset, yoffset

Note

Any change to the width and height values affect all layers.

Parameters:

`layer_name` The model full path of the layer to change attributes on
`tag_table` A table with tags as the keys and the new values stored as the table's key values

Example:

```
function set_layer_hidden()  
    local data = {}  
    data.hidden = 1  
    gre.set_layer_attrs("my_layer", data)  
end
```

gre.set_layer_attrs_global

```
gre.set_layer_attrs_global(  
    layer_name,  
    table  
)
```

Set attributes for a layer globally on all instances of the layer on all screens. The `layer_name` is the name of the layer. Table contains the tags and values for the attributes to set.

alpha, hidden, active, x, y, width, height

Parameters:

`layer_name` The model full path of the layer to change attributes on
`tag_table` A table with tags as the keys and the new values stored as the table's key values

gre.get_table_attrs

```
gre.get_table_attrs(  
    table_name,  
    tags  
)
```

Get attributes for a table. Key name is the name of the control or a variable. Tags can be any of the control tags mentioned in section 6.1.12 and any of the following values:

rows	The number of rows in the table
cols	The number of columns in the table
visible_rows	The number of visible rows in the table
visible_cols	The number of visible columns in the table
active_row	The active cell row
active_col	The active cell column
row	The row index of the upper left row
col	The column index of the upper left column
xoffset	The current scroll offset in the x direction
yoffset	The current scroll offset in the y direction

Parameters:

table_name	The model full path of the table to get information about
tags	One or more tags as strings

Returns:

A table containing the tags as keys with the associated table value being the Storyboard value associated with that tag.

Example:

```
function check_if_hidden()  
    local dk_data = {}  
    -- Get the active row/column  
    dk_data = gre.get_table_attrs("my_table", "active_row", "active_col")  
    print("Active Cell: " .. tostring(dk_data["active_row"]) .. ", "  
        .. tostring(dk_data["active_col"]))  
end
```

gre.set_table_attrs

```
gre.set_table_attrs(  
    table_name,  
    tag_table  
)
```

Set attributes for a table. The `table_name` is the name of the control or a variable. The `tag_table` contains the tags and values for the attributes to set.

`x, y, width, height, hidden, active, rows, cols, xoffset, yoffset`

Parameters:

`table_name` The model full path of the table to change attributes on
`tag_table` A table with tags as the keys and the new values stored as the table's key values

Example:

```
function resize_table()  
    local dk_data = {}  
    dk_data["rows"] = 5  
    dk_data["cols"] = 10  
    gre.set_table_attrs("my_table", dk_data)  
end
```

gre.get_table_cell_attrs

```
gre.get_table_cell_attrs(  
    table_name,  
    row,  
    col,  
    tags ...  
)
```

Get attributes for a table cell. `table_name` is the name of the table. Tags can be a list of the following values:

`x, y, width, height, hidden`

A table with the results is returned.

Parameters:

`table_name` The model full path of the table to get information about
`row` The row of the table to get information on
`col` The column of the table to get information on
`tags` One or more tags as strings

Returns:

A table containing the tags as keys with the associated table value being the Storyboard value associated with that tag.

Example:

```
function check_if_hidden()  
  local dk_data = {}  
  -- check if my_control is currently hidden  
  dk_data = gre.get_table_cell_attrs("my_table", 1, 1, "hidden")  
  if dk_data["hidden"] == 1 then  
    print("cell 1.1 of my_table is currently hidden")  
  else  
    print("cell 1.1 of my_table is currently visible")  
  end  
end
```

gre.get_string_size

```
gre.get_string_size(  
  font,  
  font_size,  
  string,  
  length,  
  width  
)
```

Calculate the area in pixels which the given string will occupy on the screen. Optionally calculate how many characters can fit into a predefined screen area.

Parameters:

`string` The string to render
`font` The name of the font to render in
`font_size` The size of the font to render in
`string_length` The length of the string to render or 0 for all (optional)
`width` A clipping width (in pixels) for the string, used to calculate how many characters fit (optional, by default there is no clip)

Returns:

A table containing the following entries:

"num_bytes" number of bytes that will fit in the clip

"width" string width in pixels as clipped by clip width

"height" string height in pixels

"line_height" height in pixels of the specified font

gre.poly_string

```
gre.poly_string(  
    x_values,  
    y_values  
)  
or  
gre.poly_string(  
    {{x=, y=}, ...}  
)
```

This is a higher performance function for generating a polygon string based on a set of numeric data points maintained in Lua table arrays.

In the two argument form, the function receives as inputs two Lua tables whose content represents the numeric x and y data points to be converted to a string. The tables are 1 based arrays and must be of the same length.

In the single argument form, the function receives as input a single Lua table whose array content are tables with an "x" and "y" member value.

The string returned is designed to be compatible with the Storyboard polygon plugin and is in the form of X1:Y1 X1:Y2 ...

Parameters:

x_values,
y_values An table containing numeric data for the x and y points respectively.

{{x=, y=}} A table containing tables with x and y members specifying the x and y points.

Example:

```
-- Create a triangle polygon in a 100x100 square  
local x_points = { 0, 50, 100 }           -- Left, Middle, Right  
local y_points = { 100, 0, 100 }          -- Bottom, Top, Bottom  
local xy_string = gre.poly_string(x_points, y_points)  
print("X Y String: " .. xy_string)  
  
-- Create the same triangle, but with x,y member variables  
local xy_points = { {x=0,y=100}, {x=50,y=0}, {x=100,y=100} }  
local xy_string = gre.poly_string(xy_points)  
print("XY String: " .. xy_string)
```

gre.resolve_data_key

```
gre.resolve_data_key(  
    key1  
    [, key2, ...]  
)
```

This function allows Lua scripts to resolve Storyboard context variables to a fully qualified name based on the current execution context.

Parameters:

key1 ... One or more string arguments containing the variable to resolve.

Returns:

A table containing the arguments provided on input as keys with the values being the resolved data value.

Example:

```
-- Resolve the application my_var to a fully qualified name  
local varname = "${app:my_var}"  
local dv = gre.resolve_data_key(varname)  
print("Full path for ${app:my_var} is " .. dv[varname])
```

gre.load_resource

```
gre.load_resource(  
    pool_name,  
    resource_name,  
    [pool parameters]  
)
```

This function will force the loading of a resource, such as an image or font, into the Storyboard application. This can be used in order to avoid load time delays that may be incurred as resources are lazy loaded into the application.

Parameters:

pool_name The name of the resource pool: image or font
resource_name The name of the resource that is to be loaded

The optional parameters vary depending on the pool being specified and are not required:

image pool:

w, h, background The width and height to cache the image at, and whether or not to load the image asynchronously the 'background'

font pool:

size, antialias The point size of the font and if anti aliasing is to be used

These options can be passed in as a table in the 3d parameter instead, which allows any or all of them to be specified. On completion of a 'background' loaded resource, the following event is sent:

gre.resource_loaded 1s0 resource

Example:

```
-- Call this on app init to pre-load the image and font into the cache
function on_app_init(mapargs)
    gre.load_resource("image", "images/tree.jpg")
    gre.load_resource("font", "fonts/DejaVu.ttf")
-- Call this on app init to pre-load the image and scale it to 100x100
    gre.load_resource("image", "images/scaledtree.jpg", 100, 100)
    local tbl = {}
    tbl["w"]=100
    tbl["h"]=100
    tbl["background"]=1
-- Call this on app init to pre-load the image and scale it to 100x100
-- asynchronously
    gre.load_resource("image", "images/scaledtreebg.jpg", tbl)
end
```

gre.load_image

```
gre.load_image(
    image_name,
    [optional table of parameters]
)
```

This function will force the loading of an image into the Storyboard application. This can be used in order to avoid load time delays that may be incurred as resources are lazy loaded into the application.

Parameters:

resource_name The name of the resource that is to be loaded

The optional parameters are as follows:

w The width to cache the image at

h The height to cache the image at

background Whether or not to load the image asynchronously the 'background'

On completion of a 'background' loaded resource, the following event is sent:

gre.resource_loaded 1s0 resource

gre.dump_resource

```
gre.dump_resource(  
    pool_name,  
    resource_name  
)
```

This function performs the opposite of the `gre.load_resource` call and removes a resource from the specified resource pool cache.

Parameters:

<code>pool_name</code>	The name of the resource pool: image or font
<code>resource_name</code>	The name of the resource that is to be removed

Example:

```
-- Force the tree.jpg image out of the cache, image will reload as required  
function flush_tree_image()  
    gre.dump_resource("image", "images/tree.jpg")  
    gre.dump_resource("font", "fonts/DejaVu.ttf")  
end
```

gre.walk_pool

```
gre.walk_pool(  
    pool_name,  
)
```

This function reports on the memory used by all of the resources loaded into a particular resource pool.

Parameters:

<code>pool_name</code>	The resource pool whose content should be reported
------------------------	--

Returns:

A table is returned with keys as the resources that are contained in the pool and values indicating the number of bytes that a particular resource is using within the system.

Example:

```
-- Display the content of the current image cache
function show_image_cache(mapargs)
    print("Images")
    local data = gre.walk_pool("image")
    for k,v in pairs(data) do
        print("  ".. tostring(k) .. "=" .. tostring(v))
    end
end
```

gre.timer_set_timeout

```
gre.timer_set_timeout(
    function,
    timeout
)
```

This function creates a one-shot timer which fires after "timeout" milliseconds and then executes "function"

Parameters:

function	The function to be called when the timer fires
timeout	The time in milliseconds before the timer should fire

Returns:

A piece of lightuserdata which serves as an identifier for the timer

Example:

```
local idval = {}
function cb_func()
    print("CB FUNC HAS BEEN CALLED")
end

--Call cb_func after 1 second
function cb_set_timeout()
    idval = gre.timer_set_timeout(cb_func, 1000)
end
```

gre.timer_set_interval

```
gre.timer_set_interval(
    function,
    interval
)
```

This function creates a repeating timer which fires every "interval" milliseconds and then executes "function"

Parameters:

function	The function to be called when the timer fires
interval	The time in milliseconds of how often the timer should fire

Returns:

A piece of userdata which serves as an identifier for the timer

Example:

```
local idval = {}
function cb_func()
    print("CB FUNC HAS BEEN CALLED")
end

--Call cb_func every 2 seconds
function cb_set_interval()
    idval = gre.timer_set_interval(cb_func, 2000)
end
```

gre.timer_clear_timeout

```
gre.timer_clear_timeout(
    id
)
```

This function stops an existing timer from firing

Parameters:

id	The userdata representing the timer
----	-------------------------------------

Returns:

Nothing

Example:

```
local idval = {}
```

```
function cb_func()
    print("CB FUNC HAS BEEN CALLED")
end

--Call cb_func after 5 seconds
function cb_set_timeout()
    idval = gre.timer_set_timeout(cb_func, 2000)
end

function cb_clear_timeout()
    local data

    data = gre.timer_clear_timeout(idval)
end
```

gre.timer_clear_interval

```
gre.timer_clear_interval(
    id
)
```

This function stops an existing timer from firing

Parameters:

id The lightuserdata representing the timer

Returns:

Nothing

Example:

```
local idval = {}
function cb_func()
    print("CB FUNC HAS BEEN CALLED")
end

--Call cb_func every 5 seconds
function cb_set_interval()
    idval = gre.timer_set_interval(cb_func, 2000)
end

function cb_clear_interval()
    local data

    data = gre.timer_clear_interval(idval)
end
```

gre.thread_create

```
gre.thread_create(func)
```

This function starts a new operating system thread of execution that is independent from Storyboard's main thread. The function provided as an argument indicates the starting context for this new thread of execution.

The Storyboard data and event (`get_data/set_data/send_event`) API are thread safe. However the execution of data changes outside of the main thread of execution can have a significant impact on performance of the application and the preferred way of synchronizing data obtained in a thread with the Storyboard UI thread is by using a Storyboard IO event and sending the data via `gre.send_event` or `gre.send_event_data`. There are no thread specific synchronization primitives, such as mutexes, for synchronizing Lua data access, serialize to the main thread using an event if this is a requirement.

In scenarios where a controlled shutdown and restart of a Storyboard application is required, separate threads of execution pose a synchronization challenge. In these situations all created thread(s) must have their execution interrupted and terminate in order for a clean shutdown to be observed. This can be accomplished nominally by intercepting the `gre.quit` event and then taking appropriate action to flag a shutdown variable or send an unblocking event.

This function is not available on all systems and is not available if `gre.thread_create` is set to nil.

Parameters:

`func` The Lua function to run in a separate thread of execution from the main UI thread.

Returns:

Nothing

Example:

```
-- Flag to indicate that we want our threads to quit executing
local quit_threads = false

-- Run a poll loop waiting for a file (a_file) to appear and
-- then send an event
function async_function()
    while(not quit_threads) do
        local fp = io.open("a_file")
        if(fp ~= nil) then
            fp:close()
            gre.send_event("file_created")
            return
        end
    end
end

-- Create the monitoring thread of execution
gre.thread_create(async_function)
```

gre.vfs_open

```
gre.vfs_open(filename)
```

This function provides read-only file access to resources packaged in the Storyboard virtual filesystem for embedded targets with no filesystem support. The function works similarly to the Lua `io.open()` and returns a FILE type object based on a project relative path. The returned object provides the `file:read()`, `file:lines()`, `file:seek()` and `file:close()` operations as described in the standard Lua documentation.

The use of this function can incur significant memory overhead if large files are read entirely into memory.

This function is not available on all systems and is not available if `gre.vfs_open` is set as nil.

Parameters:

filename The name of the resource to open as a project relative path (ie translations/french.csv)

Returns:

A file handle that contains read, seek, lines, close operations.

Example:

```
-- Use a simple CSV parser that assumes a single delimiter between
-- variable and value
function load_translation()
    local fp = gre.vfs_open("translations/french.csv")
    if(fp ~= nil) then
        local comma, var, value
        for l in fp:lines()
        do
            comma = string.find(l, ',', 0, true)
            if(comma ~= nil)
            then
                var = string.sub(l, 1, comma - 1)
                value = string.sub(l, comma + 1)
                gre.set_data({ [var] = value })
            end
        end
        fp:close()
    end
end
```

gre.mstime

```
gre.mstime()
gre.mstime(app_relative)
```

Retrieve the current time in milliseconds in the default, no argument flavour. This call provides a higher resolution than the standard Lua `os.clock()` or `os.date()` functions.

When `true` is passed in as an argument, then the time returned will be relative to the application start time and aligned with the timestamps that are generated by the Storyboard logging API.

Returns:

The current time in milliseconds in a system specific manner (`gre.mstime()`) or the time in milliseconds since the start of the application (`gre.mstime(true)`)

Example:

```
-- Time an operation
local s = gre.mstime()
my_function()
local e = gre.mstime()
print("my_function took " .. tostring(e - s) .. "ms")

-- Determine how long from app start to this point
local delta = gre.mstime(true)
print(string.format("Application start to now: %d ms", delta))
```

gre.env

```
gre.env(
    string_key
)
or
gre.env(
    table
)
```

Return information about the Storyboard runtime environment. The input can be either a single string containing the key to look up or a table of keys for variables to match. The following table describes the available keys:

version	The version of this engine as a string value. The format of the string is four version numbers separated by dots: major.minor.service.build.
target_os	The target operating system
target_cpu	The target processor
renderer	The name of the graphics rendering technology being used.
screen_width	The dimensions of the screen width
screen_height	The dimensions of the screen height
active_screen	The name of the currently active screen
render_caps	The rendering capabilities. Currently the only defined capability is "3d" if 3d rendering is supported

mem_stats Platform memory statistics for the engine. The results are returned as a table of key value pairs with two current keys defined. The key 'process_used' indicates the memory used by the sbengine process. The key 'heap_used' indicates only the heap (malloc) memory used by the sbengine process. Not all rendering engine platforms support all metrics, in which case the value will be set to 0 indicating no information.

Parameters:

Returns:

If a single string is provided as an input argument, just a single data value for that argument is returned

If a table is provided as an input argument, then a table with key/value pairs corresponding to the keys of the input argument and the results they provide.

Example:

```
-- Get the target OS for dynamic module loading
local os = gre.env("target_os")
print("Running on target OS: " .. tostring(os))

-- Report on the storyboard version and rendering technology
local info = gre.env({"version", "renderer"})
local msg = string.format("Storyboard version %s (%s renderer)", info.version, info.renderer)
print(msg)
```

gre.animation_create

```
gre.animation_create(fps, [auto_destroy], [end_callback])
```

Create a new animation at the desired frame rate (fps). The second parameter (optional), `auto_destroy`, tells if the animation should be released once completed. If you specify a value of 1 the animation will be released and the returned id is not valid once the animation has completed. The third parameter (optional) indicates a callback function to be invoked when the animation is complete.

Parameters:

`fps` The animation frame rate
`auto_destroy` Pass 1 in to release the animation once completed
`end_callback` Provide a Lua function to be called in the animation

Returns

An animation id to be used on future animation calls, nil on failure.

Example:

```
function animation_create(mapargs, fps)
    local id
```



```
id = gre.animation_create(fps)
end
```

gre.animation_add_step

```
gre.animation_add_step(id, data)
```

Add a step to a created animation. The id must be from a call to gre.animation_create. The data parameter defines the animation step values.

Parameters:

id The animation id

data A table of animation step values which can include:

key: The data key for the animation step to act upon

rate: The animation rate string: [linear|easein|easeout|easeinout|bounce]

duration: The length of the step (msec)

offset: The offset from animation start where this step begins (msec)

from: The value to start the animation at, if not specified the value is the current value of "key"

to: The end point for the animation

delta: The delta for the end of the animation from the start point. If both "to" and "delta" are given then the "to" value is used.

Example:

```
function create_animation(mapargs)
  local data = {}

  -- slide the x position 400 pixels over 2000 msec and auto-destroy
  -- it on completion
  id = gre.animation_create(60, 1)
  data["rate"] = "linear"
  data["duration"] = 2000
  data["offset"] = 0
  data["delta"] = 400
  data["key"] = "mylayer.mycontrol.grd_x"
  gre.animation_add_step(id, data)
end
```

gre.animation_destroy

```
gre.animation_destroy(id)
```

Destroy the animation associated with id.

Parameters:

id The animation to destroy

Example:

```
function create_animation(mapargs)
  local data = {}

  -- slide the x position 400 pixels over 2000 msec
  id = gre.animation_create(60)
  data["rate"] = "linear"
  data["duration"] = 2000
  data["offset"] = 0
  data["delta"] = 400
  data["key"] = "mylayer.mycontrol.grd_x"
  gre.animation_add_step(id, data)

  -- destroy it
  gre.animation_destroy(id)
end
```

gre.animation_trigger

```
gre.animation_trigger(animation_id, data)
```

```
gre.animation_trigger("animation_name")
```

Trigger an animation to run. If an `animation_id` is used to trigger the animation, then it must be the return value from `gre.animation_create()`. If a name is used to trigger an animation, then that name must be the name of the animation specified in Designer. This function can take an optional parameter, `data_table`. The `data_table` contains the tags and values for the extra arguments to set.

Parameters:

- `animation_id` The animation to trigger
- `data` A table containing the tags and values for the extra arguments to set
 - `id` The animation id used in the case of multiple animations with the same name
 - `context` The fully qualified name of an object in the model which will be used as the context for the animation

Example:

```
function create_animation(mapargs)
  local data = {}

  -- slide the x position 400 pixels over 2000 msec and auto-destroy
  -- it on completion
  id = gre.animation_create(60, 1)
  data["rate"] = "linear"
  data["duration"] = 2000
  data["offset"] = 0
  data["delta"] = 400
  data["key"] = "mylayer.mycontrol.grd_x"
  gre.animation_add_step(id, data)
```

```
gre.animation_trigger(id)
end

--Example of using gre.animation_trigger passing animation names.
function cb_toggle_cur_5day()
    if cur_5day_toggle == false then
        gre.animation_trigger("show_5day")
    else
        gre.animation_trigger("hide_mon_to_fri")
    end
end

--Example of using gre.animation_trigger with context.
function cb_toggle_cur_5day()
    local data = {}

    data["context"] = "Layer1.mycontrol"
    gre.animation_trigger("show_5day", data)
end
```

Storyboard Lua DOM Module

The Storyboard `gredom` Lua module provides a limited access to the hierarchical model used by the Storyboard Engine. This functionality is provided as an external Lua module and is located in the `lib/gredom.so` file of the Storyboard Engine target distributions.

The DOM module provides two sets of function interfaces. The first set of functions are associated with the `gredom` namespace and are used to lookup or access an Lua object (table) that contains a special set of methods (metatable entries) that are used to extract additional information about the object. In this documentation the object returned from the `gredom` namespace functions will be referred to as a `DOMOBJECT` object. Methods associated with the `DOMOBJECT` object must be invoked using the Lua colon (`:`) notation for example `DOMOBJECT:get_name()`

gredom.get_application

```
gre.get_application()
```

Get an object handle for the application root

Returns:

A `DOMOBJECT` object that represents the application.

gredom.get_object

```
gre.get_object(fqn)
```

Get an object handle for the

Parameters:

fqn The fully qualified name of the model entry (screen, layer, control), or a short name to autosearch for a match

Returns:

A DOMOBJECT object that represents the named model object.

DOMOBJECT:get_name

```
DOMOBJECT:get_name()
```

Gets the name of the specified DOM Object

Returns:

The name of the specified DOMOBJECT

DOMOBJECT:get_type

```
DOMOBJECT:get_type()
```

Gets the Storyboard type of the specified DOMOBJECT. The type may be one of gredom.APP, gredom.SCREEN, gredom.LAYER, gredom.LAYER_INSTANCE, gredom.GROUP, gredom.CONTROL, gredom.TABLE, gredom.TEMPLATE.

Returns:

A the type of the specified DOMOBJECT.

DOMOBJECT:get_parents

```
DOMOBJECT:get_parents()
```

Gets the parent DOMOBJECT objects for the specified DOMOBJECT. An array of parents is returned because in some cases, such as for a layer, there may be more than one parent representation.

Returns:

An array table containing the parent DOMOBJECT entries.

DOMOBJECT:get_children

```
DOMOBJECT:get_children()
```

Gets the child DOMOBJECT objects for the specified DOMOBJECT. This function returns only the model objects and does not include the variables.

Returns:

An array table containing the child DOMOBJECT entries

DOMOBJECT:get_variables

```
DOMOBJECT:get_variables()
```

Gets variables associated with the specified DOMOBJECT

Returns:

An array table containing the variables associated with this object.

Lua DOM Samples

```
require("gredom")

-- Print a list of all of the user variables associated with a specified
-- control
function print_variables(control_name)
    -- Get the DOM object for the control name passed in
    local domObject = gredom.get_object(control_name)
    if(domObject == nil) then
        print("Can't find name for " .. tostring(control_name))
        return
    end

    -- Get the variables defined on this DOM object
    local vars = domObject:get_variables()
    if(vars == nil or #vars == 0) then
        print("No variables for " .. control_name)
    else
        print("Variables for " .. control_name)
        for i=1,#vars do
            print("# " .. tostring(vars[i]))
        end
    end
end

-- Print out all of the screens where this control's container layer is
-- being used
function print_used_on_screens(control_name)
    -- Get the DOM object for the control name passed in
    local domObject = gredom.get_object(control_name)
    if(domObject == nil) then
        print("Can't find name for " .. tostring(control_name))
        return
    end

    -- Build up the full path to this object
```

```
-- Walk up the tree looking at all parents adding screens we find
local screen_list = {}
local parent_list = {}
table.insert(parent_list, domObject:get_parents())

local i = 1
while i <= #parent_list do
    local parents = parent_list[i]
    for p=1,#parents do
        -- If this was a screen, add it to our collection
        if parents[p]:get_type() == gredom.SCREEN then
            screen_list[parents[p]] = true
        else
            -- If this has parents of its own, then add them to the search
            -- list
            parents = parents[p]:get_parents()
            if (parents ~= nil and #parents > 0) then
                table.insert(parent_list, parents)
            end
        end
    end
end
i = i + 1
end

-- Print out all of the screens that we have identified
print(control_name .. " is used on the following screens:")
for screen,v in pairs(screen_list) do
    print("# " .. screen:get_name())
end

end

-- Invoke our DOM example functions with the context control
function CBDom(mapargs)
    print_used_on_screens(mapargs.context_control)
    print_variables(mapargs.context_control)
end
```

Storyboard Lua Android Integration

Storyboard Lua Android Integration

On Android target platforms Storyboard provides an additional level of platform integration. In order to access the native Java service API on Android platforms Storyboard has incorporated the LuaJava [<http://www.keplerproject.org/luajava/>] module to provide a bridge from Storyboard Lua script functions to the Android Java API.

Access to the LuaJava bridge is through the `luajava` Lua variable. On non-Android platforms, this variable will not be defined and this can be used to provide alternate or simulated behaviour.

```
function my_callback(mapargs)
    if(luajava == nil) then
```

```
    print("LuaJava bridge not available")
    return
end

-- LuaJava available for use ...
end
```

The general mapping of standard Lua/Java types such as strings and numbers is handled transparently so that Lua strings can be used in Java constructors and methods in the same way that the Java String class would normally be used and similarly for Lua numbers and vice/versa.

When a Lua variable is created that is a reference or proxy to a Java object, then access to the methods of that object are performed using the colon (:) notation with the Lua variable, i.e. `lua_variable:method_name()` notation. When accessing static member variables of an object, this can be performed using the traditional dot (.) notation `lua_variable.member_variable_name`. This is further demonstrated in the examples shown below.

In order to access a nested Java class for instantiation or binding, the dollar sign (\$) must be used as a separator. For instance, if the Java class Bar is a nested class of Foo, then binding would work as follows: `luajava.bindClass("Foo$Bar")`. This is further demonstrated in the examples below.

A description of the complete Android Java API is beyond the scope of this document. For a complete coverage of the Android API refer to <http://developer.android.com/reference/packages.html>. Depending on the functionality that your application is going to access, there may be additional restrictions that must be explicitly declared in the `AndroidManifest.xml` file. Permissions can be added in the Advanced Options section when exporting your Android project. The `android:debuggable` option has been changed to false by default. To change this, you will need to use your own custom manifest file. Export your manifest file to view it by clicking the Export button under the Manifest File tab. You can make changes to this file and then select it as a custom manifest file when exporting to ensure the manifest file is setup the way you want it to be.

Within the Android environment the Storyboard Engine execution takes place outside of the main Android/Java event loop. When integrating with the Android API's developers should always consider that they are using the Android API as if they were executing in a background thread and act accordingly. This may require the creation of additional `Looper` message event handlers if callback event handlers are being used. For more information on Android process model and multi-threading considerations, refer to the Android documentation: <http://developer.android.com/guide/components/processes-and-threads.html>.

Android Lua Java API

The mapping of Lua referenced objects to Android Java objects is relatively straightforward. All of the API functionality is accessed via the `luajava` Lua global variable. This variable provides four functions that can be used to access and manipulate standard Java objects and one variable that provides the Android `Activity` that is required.

`luajava.newInstance(className, ...)` This function creates a new Java object based on the fully qualified class name. Any additional parameters that are provided are passed through to the standard Java constructor.

The return value is a Lua variable that is a proxy to the Java object or `nil` if the class could not be instantiated.

```
-- Create an instance of a Java string tokenizer
```

```
local strTk = luajava.newInstance("java.util.
StringTokenizer", "a,b,c,d", ",")
while strTk.hasMoreTokens() do
    print(strTk.nextToken())
end

-- Create a new Android Intent object (unpopulated)
local intent = luajava.newInstance
("android.content.Intent")
```

`luajava.bindClass(className)`

This function creates a reference to a Java class based on a fully qualified class name. This is different from `newInstance()` in that a new Java object is not created and the constructor is not invoked, but simply a reference to the class is returned. Use this when you need access to static fields or methods of a Java object.

The return value is a Lua variable that is a proxy to the Java Class object specified or `nil` if the class could not be found.

```
-- Get the current system time
local sys = luajava.bindClass("java.lang.System")
print ( sys.currentTimeMillis() )

-- Parse a string into an Android Uri
local uriClass = luajava.bindClass("android.net.Uri")
local phoneURI = uriClass:parse("tel:6135951999")
```

`luajava.new(classObject,)`

This function is similar to the `newInstance()` function but rather than taking a fully qualified class name it takes an existing Class reference, generally obtained from calling `bindClass()`. Additional parameters can be passed to the Java constructor..

The return value is a Lua variable that is a proxy to the Java object or `nil` if the class could not be instantiated.

```
-- Create a new string instance
str = luajava.bindClass("java.lang.String")
strInstance = luajava.new(str)
```

`luajava.createProxy(interfaceNames, luaObject)`

If a Java API requires an interface to be implemented or provided as a set of callbacks, then it is where the `createProxy()` function can be used. The *interfaceNames* parameter is a comma separated list of fully qualified Java interfaces that will be implemented by the Lua variable *luaObject*. The names of the interface methods must be present in the `luaObject` variable.

The return value is a Lua variable that can be passed to any function or method that requires an implementation of that interface. If the creation of the proxy fails, then `nil` is returned.


```
-- Create a Lua variable with the same interface as an ActionListener
local button_cb = {}
function button_cb.actionPerformed(ev)
    -- I would do something interesting here ...
end

-- Map the Lua variable to the Java interface
buttonProxy = luajava.createProxy("java.awt.ActionListener", button_cb)

-- Use the newly created interface instance on a Java object
button = luajava.newInstance("java.awt.Button", "execute")
button.addActionListener(buttonProxy)
```

luajava.nativeActivity()

All significant interaction on an Android system involves working with an Activity (see <http://developer.android.com/reference/android/app/Activity.html>) Storyboard applications that are deployed to Android devices run as *native activities* which is a special class of the general Activity that allows those applications to interact directly with the graphics context and are generally C/C++ applications rather than pure Java applications.

The return value of this function is a Lua variable that is a proxy for the NativeActivity Java class used by this application or nil if the class could not be instantiated.

```
-- Start an activity specified by a previously created Intent object
local na = luajava.nativeActivity()
if(na ~= nil) then
    na.startActivity(intent)
else
    print("No Native Activity")
end
```

Storyboard Lua Android Example

This example demonstrates how a phone call could be invoked as part of a Lua callback. In order for this example to work, the AndroidManifest.xml file must be changed to give permission for calls to be made:

```
<uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
```

```
-- Log message routine to route diagnostic messages
local function lm(msg)
    print(msg)
end

-- Call a selected phone number using the Android API
-- Input is the string number value that is to be called
local function call_phone_number(number)
    if(luajava == nil) then
```

```
    lm("No luajava Lua object")
    return
end

local na = luajava.nativeActivity()
    if(na == nil) then
        lm("No native activity available")
        return
    end

local uriClass = luajava.bindClass("android.net.Uri")
if(uriClass == nil) then
    lm("No java.lang.String object")
    return
end

local phoneURI = uriClass:parse("tel:" .. tostring(number))
if(phoneURI == nil) then
    lm("No java.net.URI object")
    return
end

local intentClass = luajava.bindClass("android.content.Intent")
if(intentClass == nil) then
    lm("No intent class")
    return
end

local intent = luajava.newInstance("android.content.Intent",
    intentClass.ACTION_CALL, phoneURI)
if(intent == nil) then
    lm("No intent object")
    return
end

lm("Calling " .. number)
na:startActivity(intent)
end
```

This example demonstrates how to create a new instance of a nested inner class of a Java class. This example gets media metadata from the `android.provider.MediaStore.Audio.Media` class, which is a nested class of `android.provider.MediaStore.Audio`, which in turn is a nested class of `android.provider.MediaStore`.

```
-- In order to pass array's to any of the Android Java API's we must explicitly
-- a Java array from a Lua table and this function covers that work.
function make_array(dataClass, values)
    local arrayClass = luajava.bindClass("java.lang.reflect.Array")
    if(arrayClass == nil) then
        print("Can't get array class")
        return nil
    end
end
```

```
end

local newTypedArray = arrayClass:newInstance(dataClass, #values)
if(newTypedArray == nil) then
    print("Can't get array class")
    return nil
end

for i=1,#values do
    arrayClass:set(newTypedArray, i-1, values[i])
end

return newTypedArray
end

function get_album_files(album_id)
if (luajava == nil) then
    return
end

if (luajava.bindClass == nil) then
    return
end
local na = luajava.nativeActivity()

local mediastore = luajava.newInstance("android.provider.
    MediaStore$Audio$Media")
local externalURI = mediastore.EXTERNAL_CONTENT_URI
local columns = {}
columns[1] = mediastore.TITLE_KEY
columns[2] = mediastore.DURATION
columns[3] = mediastore.TITLE

local stringClass = luajava.bindClass("java.lang.String")
local array = make_array(stringClass, columns)
local where = mediastore.ALBUM_KEY .. "=?"
local what = {}
what[1] = album_id
local whatArray = make_array(stringClass, what)

local cursor = na:managedQuery(externalURI, array, where, whatArray, nil);
local res = cursor_to_table(cursor)
return res
end
```

Chapter 9. Storyboard IO

The Storyboard IO API provides a platform independent communication API that allows inter-task and inter-process queued message passing. This is primarily used to allow external communication with a Storyboard application.

Note

Storyboard IO was previously known as GREIO
The Storyboard IO API layers on top of native message passing and communication APIs, dependent on the operating system:

Linux	SysV message queue
MacOS	SysV message queue
QNX	QNX POSIX message queue (mqueue server or mq server/library)
Using the SBIO_MQ_PATH environment variable you can determine which message queue technology will be used. By default the standard mqueue server and corresponding C library mq_* functions will be used.	
If the SBIO_MQ_PATH environment variable is set to point at the libmq.so library (ie SBIO_MQ_PATH=/usr/lib/libmq.so) then the default binding for the message queue implementation can be changed to use the mq server and the corresponding mq library functions.	
If the SBIO_MQ_PATH environment variable is used, then it must be used consistently with all Storyboard IO clients and servers that want to communicate with one another.	
WinCE/WinCompact7	WinCE MSMQ
Win32	MS-Mailslots

The API provides transport delivery guarantees for messages that are placed into the queue regardless of the implementation. The maximum transport size of a message and the total queue capacity varies slightly from implementation to implementation however a 2K message size should be considered a design limit with the practical implementation limit around a 4K message payload size.

Storyboard IO integration with the Storyboard Engine is implemented as a plugin. It is possible to create alternate Storyboard IO implementations that take advantage of custom communication facilities available on a platform.

The Storyboard IO plugin provides a single communication channel which clients can used to inject events into the Storyboard application. These events will be queued and dispatched in the same manner as internally generated events.

Client applications can use Storyboard IO to create their own communication channels and then receive events from that channel from the Storyboard application or from any other Storyboard IO client.

Connecting to a Storyboard Application

In order to communicate with a running Storyboard application the external application must first attach to the application's Storyboard IO channel. By default this channel is named after the deployment

bundle file (ie [bundlename].gapp), however the name of this channel can be customized by specifying `-ogreio,channel=newname` as an option to sbengine. Alternatively, on some platforms, it is also possible to set a GREIONAME environment variable to the channel name.

Once the channel name is determined, the connection to the channel can be established with the `gre_io_open()` function. This will connect to the channel and return a handle that can be used for future communication.

Once the application has determined that no further communication is necessary the channel should be closed via the `gre_io_close()` function.

Sending Events to a Storyboard Application

Storyboard events contain string based names and a variable data field. For this reason the event data must be serialized into a buffer for communication. The Storyboard IO API provides the functions needed to both serialize your data and send the event. The event you wish to send must first be serialized via a call to `gre_io_serialize()`. This will allocate a serialized data buffer for your event. The event can then be sent via the `gre_io_send()` function. Once the event has been sent the buffer can be reused or freed via a call to `gre_io_free_buffer()`.

Note

Serialized buffers can be reused multiple times. The `gre_io_serialize_buffer()` function will resize or reallocate the buffer if the data being serialized is larger than the existing buffer. This is designed to cut down on repetitive memory allocation and deallocation churn.

Data parameters must be sent in order of descending alignment requirements. Example: 4u1 4u1 2u1 1s0 is good, 2u1 4u1 4u1 1s0 is not

```
gre_io_t                *send_handle;
gre_io_serialized_data_t *nbuffer = NULL;
const char              *event_data = "my event data"

/*
 * Connect to a channel to send messages.
 */
send_handle = gre_io_open("my_channel", GRE_IO_TYPE_WRONLY);
if(send_handle == NULL) {
    printf("Can't open send handle [%s]\n", argv[1]);
    return 0;
}

/*
 * Send a named event containing no data payload
 */
nbuffer = gre_io_serialize(nbuffer, NULL,
    "my_event_name",
    NULL,
    NULL,
    0);

gre_io_send(send_handle, nbuffer);
```

```
/*
 * Send a named event with an additional string payload
 */
nbuffer = gre_io_serialize(nbuffer, NULL,
                           "my_event_name",
                           "ls0 data",
                           event_data,
                           strlen(event_data)+1);

gre_io_send(send_handle, nbuffer);
```

Setting Application Data

The Storyboard IO plugin provides the capability to set application variable values using the Storyboard IO API, allowing external client programs to change data dynamically.

Clients can use the `gre_io_add_mdata()` function to serialize each variable value that is to be set. As values are added to the serialized buffer, it will be grown until it reaches a maximum size for the transport, at which point the `gre_io_add_mdata()` function will return -1 indicating it is full. The data can be then sent by using the `gre_io_send_mdata()` function which will send the change request to the Storyboard IO plugin and set the appropriate values in the Storyboard application.

Note

Data must be sent in order of descending alignment requirements. Example: 4u1 4u1 2u1 1s0 is good, 2u1 4u1 4u1 1s0 is not

```
gre_io_t                *send_handle;
gre_io_serialized_data_t *md_buffer = NULL;
uint32_t                x;
char                    *ptr;
int                     ret;

/*
 * Connect to the application channel
 */
send_handle = gre_io_open("my_channel", GRE_IO_TYPE_WRONLY);
if(send_handle == NULL) {
    printf("Can't open send handle [%s]\n", argv[1]);
    return 0;
}

/*
 * Add some values to be set in the data manager
 */
ptr = "my string";
ret = gre_io_add_mdata(&md_buffer,
                      "Test.String",
                      "ls0",
                      ptr, strlen(ptr)+1);
```

```
x = 1;
ret = gre_io_add_mdata(&md_buffer,
    "Test.Number",
    "4ul",
    &x, sizeof(uint32_t));

/*
 * Send the data to be set in the application.
 */
gre_io_send_mdata(send_handle, md_buffer);
```

Receiving Events from a Storyboard Application

In order to receive events the from a Storyboard application, a client program must first create a receive communication channel using the `gre_io_open()` function. This function takes the name of a channel to create and the mode in which to open the channel, for reading or writing. Receiving clients must open it for reading.

Once the communication channel is created, then the client program then must call `gre_io_receive()` in order to receive and process events.

The client communication channel can be created in either a blocking or non-blocking mode. By default the `gre_io_receive()` function will not return unless there is an event available or an error has occurred.

Once an event has been received the data can be unserialized into its standard components using the `gre_io_unserialize()` function.

```
char    *name = (char *)arg;
gre_io_t *rhandle;
gre_io_serialized_data_t *buffer = NULL;
int ret;
char *revent_name;
char *revent_target;
char *revent_format;
uint8_t *revent_data;
int offset, i, rnbytes;

rhandle = gre_io_open(name, GRE_IO_TYPE_RDONLY);
if(rhandle == NULL) {
    printf("Can't open IO channel %s\n", name);
    return 0;
}

printf("Waiting on channel [%s]\n", name);
while(1) {
    ret = gre_io_receive(rhandle, &buffer);
    if(ret < 0) {
        printf("Problem receiving data on channel [%s]\n", name);
```

```
        break;
    }

    rnbytes = gre_io_unserialize(buffer,
                                &revent_target,
                                &revent_name,
                                &revent_format,
                                (void **)&revent_data);
    printf("Event Received [%s] on channel [%s]:\n", revent_name, name);
    printf(" Event Target: [%s]\n", revent_target);
    printf(" Event Format: [%s]\n", revent_format);
    printf(" Event Data (%d bytes):\n", rnbytes);
}

gre_io_close(rhandle);
```

Storyboard IO Utilities

Included with Storyboard Suite are some command line utilities that can be useful tools when working with and configuring Storyboard IO. These utilities provide a very thin layer on top of the Storyboard IO C API and can be used to verify that Storyboard IO is working properly on your platform.

iogen

The `iogen` utility is used to generate Storyboard IO events from the command line. The utility's command line arguments closely mirror the arguments that would be provided to the `gre_io_open` and `gre_io_send` Storyboard IO API functions. Running `iogen` without any parameters will show a usage message:

Note

Data must be sent in order of descending alignment requirements. Example: 4u1 4u1 2u1 1s0 is good, 2u1 4u1 4u1 1s0 is not

Usage `./iogen channel_name [event_target event_name size data [size data]...]`

For example:

Send a 'gre.quit' event to a client on channel 'sb'

```
./iogen sb no_target gre.quit
```

Send a 'gre.press' event (int button, subtype, x, y) @ 100,150 to a client on channel 'sb'

```
./iogen sb some_target gre.press 4u1:button 0 4u1:timestamp 0 2u1:subtype 0 2u1:x 100 2u1:y
150 2u1:z 0 2u1:id 0 2u1:spare 0
```

Send a 'progress' event with an integer field 'percent' containing the value 50 on a channel 'sb'

```
./iogen sb no_target progress 4s1:percent 50
```

Send a 'greio.verbosity' event with an integer field 'verbosity' containing the desired level of engine debugging verbosity

```
./iogen sb no_target greio.verbosity 4s1:verbosity 4
```


The `iogen` utility can also be used to set variables in a Storyboard application. To set a variable, the `event_target` parameter should contain the fully qualified path for the Storyboard variable and the `event_name` parameter should contain the SBIO event `greio.iodata_set`. For example:

Set an integer application variable 'progress' with a number (50)
`./iogen sb progress greio.iodata_set 4s1 50`

Set the text variable 'myvariable' on the control 'mycontrol' on the layer 'mylayer' with a string (Hello)
`./iogen sb mylayer.mycontrol.myvariable greio.iodata_set 1s0 "Hello"`

The definition and format of standard Storyboard events such as `gre.press` and `gre.release` can be found in the Storyboard header file `iodefs.h`.

iorcv

The `iorvc` utility is used to receive Storyboard IO events. This utility takes an input channel name as a command line parameter and prints the events it receives. Running `iogen` without any parameters provides a usage message:

Usage `./iorcv [-s] channel_name`

By default, `iorcv` will loop around receiving messages until the program is terminated. By specifying `-s` you can cause `iorcv` to exit once it has received a single message.

When a message is received, a summary of the event contents is printed to the output:

```
% ./iorcv my_channel
Waiting on channel [my_channel]
Event Received [my_event_name] on channel [my_channel]:
Event Target: [no_target]
Event Format: []
Event Data (0 bytes):
```

This would be the response to an event generated by `iogen` with the following arguments:

```
% ./iogen my_channel no_target my_event_name
Connecting to Storyboard IO channel [my_channel]
```

Storyboard IO API

The details the functions available in the Storyboard IO library, `libgreio.a`, are also documented in the Storyboard IO header file `gre/greio.h`.

`gre_io_add_mdata`

```
int gre_io_add_mdata(  
    gre_io_serialized_data_t ** mbuffer,  
    const char * key_name,  
    const char * data_format,  
    const void * data,  
    int data_nbytes  
)
```

Add a data change key/value pair to a serialized buffer. This call can be used to serialize multiple data changes into a single Storyboard IO send operation to improve efficiency.

Once an multi-part data buffer is constructed, it can be sent using the `gre_io_send_mdata` function.

Parameters:

`buffer` The buffer containing the serialized data
`key_name` The data key which is to be set
`data_format` The format for the data to be set
`data` The data value to set
`data_nbytes` The number of bytes used for the data

Returns:

-1 on failure anything else is success

gre_io_close

```
void gre_io_close(  
    gre_io_t * handle  
)
```

Close an io connection. Any pending clients will return with an error on their pending actions.

Parameters:

`handle` A valid handle created with `gre_io_open()`

gre_io_free_buffer

```
void gre_io_free_buffer(  
    gre_io_serialized_data_t * buffer  
)
```

This de-allocates the memory associated with a buffer created through the Storyboard IO API.

Parameters:

buffer The buffer whose memory is to be de-allocated

gre_io_grow_buffer

```
void gre_io_grow_buffer(  
    gre_io_t*                handle,  
    gre_io_serialized_data_t * buffer  
)
```

This function attempts to expand the internal capacity of the Storyboard IO transport to ensure that the payload contained within serialized buffer can be transmitted.

Note

This call is not supported by all platforms and may fail if the transport buffer can not be resized.

Parameters:

handle The handle to the Storyboard IO channel to resize
buffer The buffer whose capacity is to be matched by the transport

Returns:

-1 on failure otherwise success

gre_io_open

```
gre_io_t* gre_io_open(  
    const char *    io_name,  
    int            flag,  
    ...  
)
```

Open a Storyboard IO communication channel using a named connection.

Parameters:

io_name The name of the io-channel to use
flags The mode you want to open the queue in
Flags define how the connection is opened. Possible flags are:
GRE_IO_TYPE_RDONLY: open read only, creating the channel if it doesn't exist
GRE_IO_TYPE_XRONLY: open for exclusive read, unlinking an existing channel and creating a new one
GRE_IO_TYPE_WRONLY: open write only
GRE_IO_FLAG_NONBLOCK: open non-blocking

Returns:

Returns a valid Storyboard IO handle or NULL if no channel can be created.

gre_io_receive

```
int gre_io_receive(  
    gre_io_t *                handle,  
    gre_io_serialized_data_t ** buffer  
)
```

Receive a serialized event from a channel. By default this call blocks until an event is received or until the channel is destroyed unless the GRE_IO_FLAG_NONBLOCK flag was passed to the `gre_io_open()` call.

In order to receive events, the handle must have been opened for reading using one of GRE_IO_RDONLY or GRE_IO_XRDONLY.

Parameters:

`handle` A valid handle created with `gre_io_open()`
`buffer` A pointer to a serialized buffer pointer. If the buffer is NULL then a new buffer is allocated otherwise the buffer provided is used to store the received event.

Returns:

The size of the message received in bytes or -1 on failure.

When a data buffer is successfully received, the event specific content can be extracted by making a call to `gre_io_unserialize`. The values returned by the unserialize call will be pointers directly into the memory allocated to the data buffer. Clients can read and write to the extracted values (such as event name, event format and data payload) directly as long as the serialized buffer is not re-used at the same time. Any data from the event that needs to be maintained across calls to `gre_io_receive` must be copied by the user before the serialized buffer is re-used.

gre_io_send

```
int gre_io_send(  
    gre_io_t *                handle,  
    gre_io_serialized_data_t *  buffer  
)
```

Send a serialized event buffer to a channel. In order to send events, the handle must have been opened for writing using GRE_IO_WRONLY.

Parameters:

handle A valid handle created with gre_io_open()
buffer A data buffer containing a serialized event

Returns:

-1 on failure otherwise success.

gre_io_send_mdata

```
int gre_io_send_mdata(  
    gre_io_t *                handle,  
    gre_io_serialized_data_t * md_buffer  
)
```

Send a serialized buffer of mdata (data manager key/value pairs) to the handle. The handle must have been opened for writing using GRE_IO_WRONLY.

Parameters:

handle A valid handle created with gre_io_open()
buffer A data buffer containing a serialized data

Returns:

-1 on failure anything else is success

gre_io_serialize

```
gre_io_serialized_data_t* gre_io_serialize(  
    gre_io_serialized_data_t *    buffer,  
    const char *                  event_addr,  
    const char *                  event_name,  
    const char *                  event_format,  
    const void *                  event_data,  
    int event_nbytes  
)
```

Serialize individual event items (see gre/io_mgr.h) into a single buffer for transmission using Storyboard IO.

Parameters:

buffer The buffer that will contain the serialized data or NULL if a new buffer should be allocated
event_addr The name of the event target, or NULL to send to the default target
event_name The name of the event to send, or NULL to send an empty event
event_format The format description of the data or NULL if no data is being sent

event_data A pointer do the data to transmit, or NULL if no data is transmitted
event_nbytes The number of data bytes to transmit, or NULL if no data is transmitted

Returns:

A buffer with the serialized data or NULL on error. It may be necessary for the internal buffer to be re-sized or re-allocated if the new data payload is larger than the previous one is being serialized.

gre_io_size_buffer

```
gre_io_serialized_data_t* gre_io_size_buffer(  
    gre_io_serialized_data_t *    buffer,  
    int                           nbytes  
)
```

This function ensures that the specified buffer has enough internal storage capacity for a payload of nbytes size. If the buffer is NULL or the existing capacity is not large enough then a new memory buffer will be assigned to the buffer object.

Parameters:

buffer The buffer to be sized, or NULL to allocate a new buffer
nbytes The number of bytes this buffer should be able to support

Returns:

A buffer with room for a message nbytes in size or NULL if the space could not be allocated

gre_io_unserialize

```
int gre_io_unserialize(  
    gre_io_serialized_data_t *    buffer,  
    char **  event_addr,  
    char **  event_name,  
    char **  event_format,  
    void **  event_data  
)
```

Transform a serialized buffer into individual event items (see `gre/io_mgr.h`). The pointers returned point back into the content of the serialized buffer so the buffer can't be de-allocated until clients are finished referencing the event items returned from this call.

Parameters:

buffer The buffer containing the serialized data

event_addr Location to store the event target
event_name Location to store the event name
event_format Location to store the event format
event_data Location to store the event data

Returns:

The number of bytes in the event_data structure

gre_io_zero_buffer

```
void gre_io_zero_buffer(  
    gre_io_serialized_data_t *  buffer  
)
```

This clears the internal byte count of the buffer, but does not de-allocate the buffer's memory.

Use this function to reset a buffer in between multiple calls to `gre_io_serialize`

Parameters:

buffer The buffer to have its byte count cleared

Chapter 10. Storyboard 3D Support

3D Rendering Fundamentals

At the most basic level, rendering of 3D content is accomplished by using matrix and vector mathematics to transform points and directions between various coordinate spaces.

Understanding a few of the underlying concepts will help a designer make informed decisions when configuring 3D Model render extensions in Storyboard Designer. Below we will explain the coordinate spaces that are applicable to 3D rendering in Storyboard and explain how they relate to the properties of the 3D Model render extension.

World space is a three dimensional space that serves as the basis for defining all the other coordinate spaces. The locations of the camera and model in the 3D Model render extension properties are coordinates in world space. It is important to note that each 3D Model render extension instance references its own model data and is effectively a 2D portal into a distinct three dimensional world.

In Storyboard, we define the default position and orientation of the camera to be at the origin of World space and looking down the World space negative z-axis. There are 2 primary camera modes which determine the effect of the Camera parameters on defining View space (also called Camera space).

In “Orbit” mode, the Azimuth and Elevation parameters first rotate View space around the World space y-axis and x-axis (respectively). The camera X, Y, Z position then position the camera in this rotated space. By defining View space using transformations in this order, we can achieve a neat effect. If we set only the Z position of the camera, Azimuth and Elevation now spin the camera around the World space origin, with the camera always looking toward the origin.

In “Fly” mode, the camera X, Y, Z position define the position of the origin (0,0,0) of View space within world space. Azimuth and Elevation now rotate the View space around the y-axis and x-axis (respectively) of View space. This allows a camera that can freely look “away” from the World space origin in any direction.

You may notice in the above description that the above descriptions of Azimuth and Elevation are in terms of y-axis and x-axis, and not the z-axis. In order to simplify rotations, Storyboard does not allow the camera to be “rolled” along the View z-axis.

The 3D Model render extension takes as a parameter a single model file per instance. Storyboard supports .obj and .fbx files as 3D model input. Since FBX file support is provided by a closed-source library maintained by Autodesk. This library has support for limited number of platforms and architectures. To help mitigate this limitation, and provide an opportunity for offline optimization of model data, FBX files are converted on import to SSG (Storyboard Scene Graph) files.

The Scene Graph and Transformations

We support a hierarchical scene graph for defining a 3D scene. We define the Node to be the basic building block. Currently a node may be a:

- Group
- Mesh
- Light

All nodes inherit the transform (coordinate space) of their parent.

Groups define a set of children nodes, and a coordinate space which all children nodes inhabit.

The complete order of transformations within a Group node is the following:

- Inherited transform from parent
- Local (bind) transform from scene graph
- Deformation transforms
 - Translation
 - Rotation (around X-axis, followed by Y-axis and finally Z-axis)
 - Scaling

Meshes and Lights are leaf nodes.

Meshes define:

- Geometry
- Material information related to portions of the geometry.

Lights may be one of 2 types:

- Directional, best used for modelling distant constant light sources, such as the sun
- Point (or omni-directional) lights, best used for lights that emanate from a position, such as a lamp, etc.

Material Support

We support the following attributes for a material applied to a section of geometry:

- Ambient color
- Diffuse color
- Specular color (and a specular exponent)
- Emissive color
- Alpha (transparency, 0.0 completely transparent, 1.0 completely opaque)

We also support a diffuse texture map, which is currently used as a texture source for both diffuse and ambient color.

We store the following additional information, but do not have any support for rendering at this time:

- Reflectivity coefficient
- Separate ambient map
- Specular map
- Emissive map

- Bump map
- Normal map
- Reflection map (expected to take the form of plane, cube or spherical mapping of reflection information)

Animation Support

Information on what is possible with the FBX file format is included below, but the bottom line is that almost all 3D Modeling DCC tools dispense with almost all of this structure and bake the movements down into a single take/layer, so in Storyboard, for simplicity, we define a 3D scene animation to have:

- n Animation Channels, containing:
 - n Animation Curves

Channels are defined as a node/transform pair, such as "FrontDriversSideDoor"/RX (x rotation). These map to rows in the animation timeline in Designer.

Curves are defined by key frames, and include a key frame time and value for the transform. These will map to the endpoints of Animation Steps in Designer.

Discussion on mapping FBX Animation data into meaningful structures

Animation data specified in an FBX file for a scene takes the following structure:

- n Animation Takes, containing:
 - n Animation Layers, containing:
 - n Animation Channels, containing:
 - n Animation Curves

Animation Takes (also called Stacks internally by FBX, but nowhere else it seems) define discrete animations that you might want to play. These quite easily map to our concept of animation clips in Storyboard Designer. Unfortunately, support for defining Animation Takes in many DCC tools is somewhat limited, see the note below. You can think of a Take in the film sense, "Action! ... do stuff, do stuff, do stuff... Cut!".

Animation Layers define a set of curves that you may want to play in parallel with another layer, allowing you to essentially modulate the defined motion of another layer. An example would be a sphere moving along a path (layer 1), while bouncing up and down (layer 2). These don't really map to anything in Storyboard, we would likely just import multiple layers of animation motion into a single clip.

Even though Animation Layers have little meaning to us, they are important because they are the container for a set of channels/curves.

Animation channels define what precisely we are deforming. These map to rows in our animation timeline. An example here would be "FrontDriversSideDoor, X rotation".

Each channel as mentioned above has a set of Curves, which basically map to the ends of Animation Steps in Storyboard. The curves are defined using key frames, with a time and a value.

In reality, most DCC tools (except MotionBuilder), will require any use of layers to be baked down into a single layer, and as mentioned above (and expanded on below), multiple takes are not natively supported either.

Support for Animation Takes

While FBX files can have multiple animation takes embedded in a single file, 2 of the most popular DCC tools, Maya and 3DS Max do not ship with the functionality to export the Animation Take data. These tools have a single animation timeline, and export the animation data a single take.

Artists desiring to specify multiple animations relating to a single model or scene have a few options, but all of them essentially defer defining this data to further down the asset pipeline.

The typical pipeline workflows are:

1. Export each separate animation into a separate FBX file. There are a whole bunch of problems with this idea.
2. Export modelling data to Autodesk MotionBuilder (previously called FilmBox, the origin of the FBX format) or another equivalent tool and use these to define the desired takes. These will import cleanly into separate takes.
3. Max and Maya have a paid plugin (fairly inexpensive - \$9 USD on TurboSquid as of the time of writing) allowing the artist to define multiple takes from the Maya and Max animation timeline. These are fairly simple tools, just defining a portion of the timeline to be each take, but are sufficient for most purposes.
4. Define all "takes" on a single timeline (with spacers between the desired takes) and export it as is. Use tools from the target middleware (as in Storyboard Designer in our case!), if they exist, to "slice" the animation into separate animations.

In order to support workflow 4, we would have to support the concept of slicing/splitting the incoming animations. As of Storyboard 4.2, this functionality is not supported.

Chapter 11. Optimizing Your Storyboard Application

Choosing the Right Image Format(s)

When creating an application the developer must define the target system screen resolution and color depth. This color depth information is used internally to decide how to create and render display elements in an efficient manner. When adding images to the user interface it is always preferable to create them in the desired color depth. If the application will be running in 16bit color then the most efficient image to render will be a 16bit image. If alpha blending/transparency is not required when this image is rendered then it is advisable to create images in the application color depth or at least remove the alpha channel in the image.

Frames Per Second

Setting your frames per second very high on a screen transition or animation might seem like a great idea that will make your UI look better. However if your target system can't keep pace with the requested frame rate the Storyboard Engine will simply drop the frames that it can't display and do extra work to achieve a lower fps that may look even worse. Usually a frames per second of 14 will look good on simple animations. The results can vary depending on what is being animated how long it is being animated for and what the content it is being overlaid to is composed of so the best plan is to evaluate your design on the target hardware and tune the settings appropriately.

Scaling Images

If you are only ever going to load an image once in your application don't scale the image, this is a performance hit at image render time. It's far better to use your favorite image editor and resize the image to exact size you intend to use it and turn the scale flag off.

Reducing Output Verbosity

Increasing the verbosity on sbengine is insightful when trying to track down behavioural issues and to gain a better understanding of the system behaviour. However, don't forget to turn off the verbosity for release since the process of outputting diagnostic messages to a console or serial terminal can cause significant slowdown due to the limited bandwidth of the output devices.

Adjusting Engine Rendering Options

The Storyboard Engine provides a number of different global rendering defaults that can be adjusted via command line options at execution time.

If your application contains a number of rotated images, then the `-orender_mgr,quality` option can be used to trade between higher execution performance (0) and a better visual interpolation (3)

If your application is using an OpenGL renderer, then the `-orender_mgr,multisample` option can be adjusted to favour less GPU consumption with less anti-aliasing (0) or choose a smoother visual presentation but longer to render (4+).

Memory

By default sbengine uses as much memory as it requires to load all the assets that the application requires (images, fonts, scripts,...) but this can be tuned to save memory. Here are some options to help with this.

- Remove any unused plugins from the plugins directory if you are simply setting a directory for the SB_PLUGINS environment variable.
- Set sbengine's `resource_mgr` options for image and font cache to appropriate values. Remember the caches must be large enough to fit all the images and fonts for your most resource intensive screen.
- Use the `Load Scaled` flag in image render extension options if you are loading a scaled version of an image (i.e. an image thumbnails screen). If you are only ever loading the image once you should resize the image before deployment to avoid the runtime cost of image scaling.

Measuring Performance

Using the Storyboard `logger` plugin it is possible to capture metrics detailing various aspects of a Storyboard applications performance. These metrics include screen, layer and control redraw times, action execution times and general event processing times. If a performance log file is captured as and saved with the file extension `.plog` (for performance log) then Storyboard Designer will automatically recognize it and open up a log file viewer that provides an organized display of the performance events.

For more information on the performance monitoring plugin and its capabilities, refer to the `Logger` plugin section of this document and the `gra.perf_state` action.

The Storyboard Embedded Engine runtime also provides a number of internal variables that can be used at runtime to display performance information. The following Storyboard variables, can be used to extract information from the runtime:

<code>grd_fps</code> (string, 1s0)	The frame rate of display updates averaged over the last 5 seconds of display. This value is only generated if the <code>-oscreen_mgr,fps</code> option is passed along to the sbengine binary.
	Storyboard display updates are entirely event driven, so unless the application that is being run is continuously changing content or generating redraw events such as is frequently done by benchmarking applications, this value may not reflect the true drawing performance of the system.
<code>grd.animation.name</code> (string, 1s0)	The name of the last completed animation.
<code>grd.animation.frames</code> (number, 4s0)	The number of frames rendered for the last animation run.
<code>grd.animation.duration</code> (number, 4s1)	The duration in milliseconds (ms) of the last animation run.

This sample demonstrates how you can use a Lua script to extract and print these values to the display.

```
function show_metrics(mapargs)
    local fps_key = "grd_fps"
```

```
local name_key = "grd.animation.name"
local frame_key = "grd.animation.frames"
local duration_key = "grd.animation.duration"

local msg
local data = gre.get_data(fps_key, name_key, frame_key, duration_key)

-- FPS generated every 5s, assuming: -oscreen_mgr,fps
if(data[fps_key]) then
    msg = string.format("Screen FPS: %d", data[fps_key])
    print(msg)
end

-- Animation data only available after animation complete
if(data[name_key]) then
    msg = string.format("Animation %s took %d ms @ %d fps", data[name_key],
        data[duration_key], data[frame_key])
    print(msg)
end
end
```

Chapter 12. Storyboard Software Updates

Automatic Updates

Storyboard Suite updates for both Designer and the Engine are provided through an software update installer within Storyboard Designer.

When Storyboard Designer starts it will automatically check for updates and then will probe every 4 hours while running. Users can also force a check for updates from within Storyboard Designer by selecting **Help > Check for Updates** in the main menu of Storyboard Designer.

If an update exists you will be notified to apply it and a wizard will guide you through download and installation process. If both a Storyboard Designer and a Storyboard Runtime update is found the installer will apply both updates in the same session.

Updates to Storyboard Designer will require a restart of your Designer application before they take affect.

Updates to Storyboard Engine will launch a secondary installer that will prompt you to install the new Storyboard Engine runtimes into a default installation directory. You can install these runtimes anywhere on your file system however the default is a labelled directory in the Storyboard Engine directory of your Storyboard Suite installation.

Part II. Storyboard Design Tutorials

Table of Contents

13. Creating a Storyboard Project from a Sample	228
Creating a New Application using the Storyboard Samples	228
Import	228
Import Sample	228
New Sample Project	229
14. Working with Multiple Application Design Files	230
Creating a Project	230
Resolving Conflicts	234
15. Creating a 3D Model Application	235
16. Creating a Multi-Touch Application	241
17. Adding Extra Libraries for Android	244
18. Adding Extra Libraries for iOS	246
19. Crank Public SVN	247
20. Exporting a Storyboard Project	255
21. Importing a Storyboard Project	257

Chapter 13. Creating a Storyboard Project from a Sample

There are a number of samples that are included with the Storyboard Suite distribution. These samples provide demonstration of various rendering plugins and action bindings available with Storyboard.

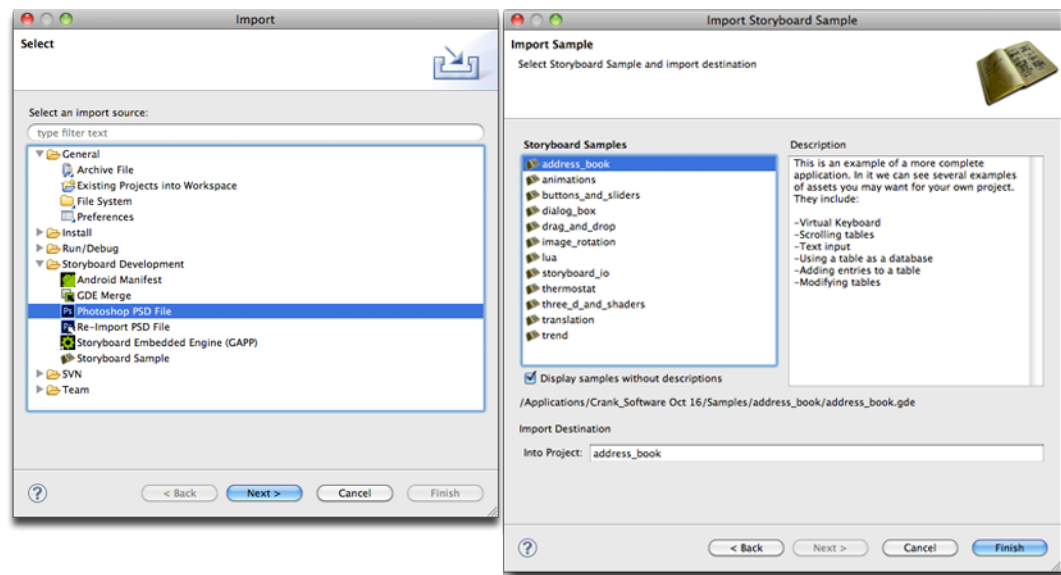
Creating a New Application using the Storyboard Samples

This quick start shows how to create a new Storyboard Application from a Storyboard Sample.

Import

Select File > Import. The Import dialog appears.

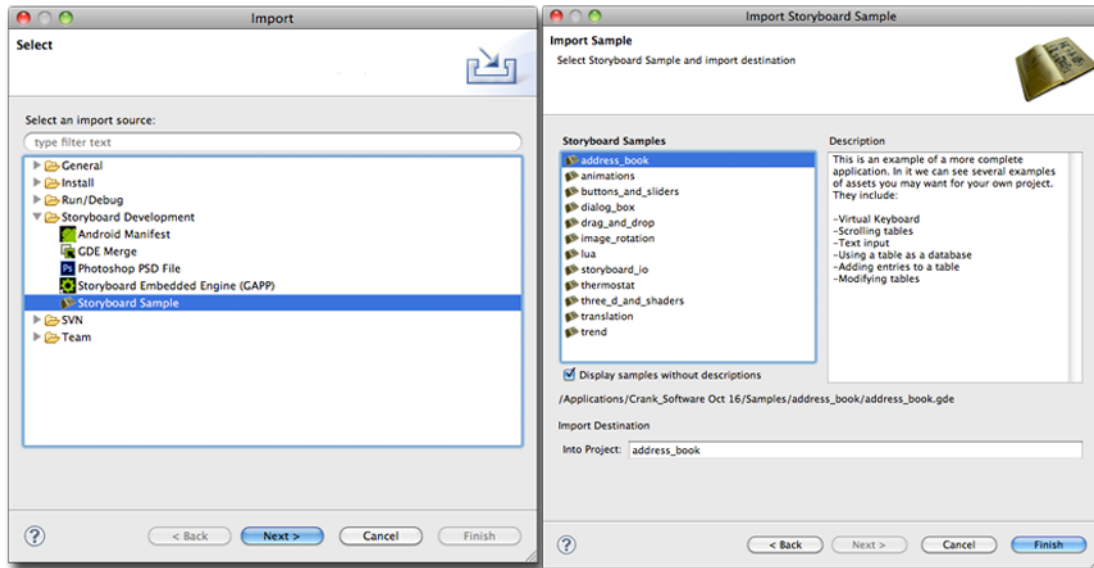
Select Storyboard Sample and click Next.



Import Sample

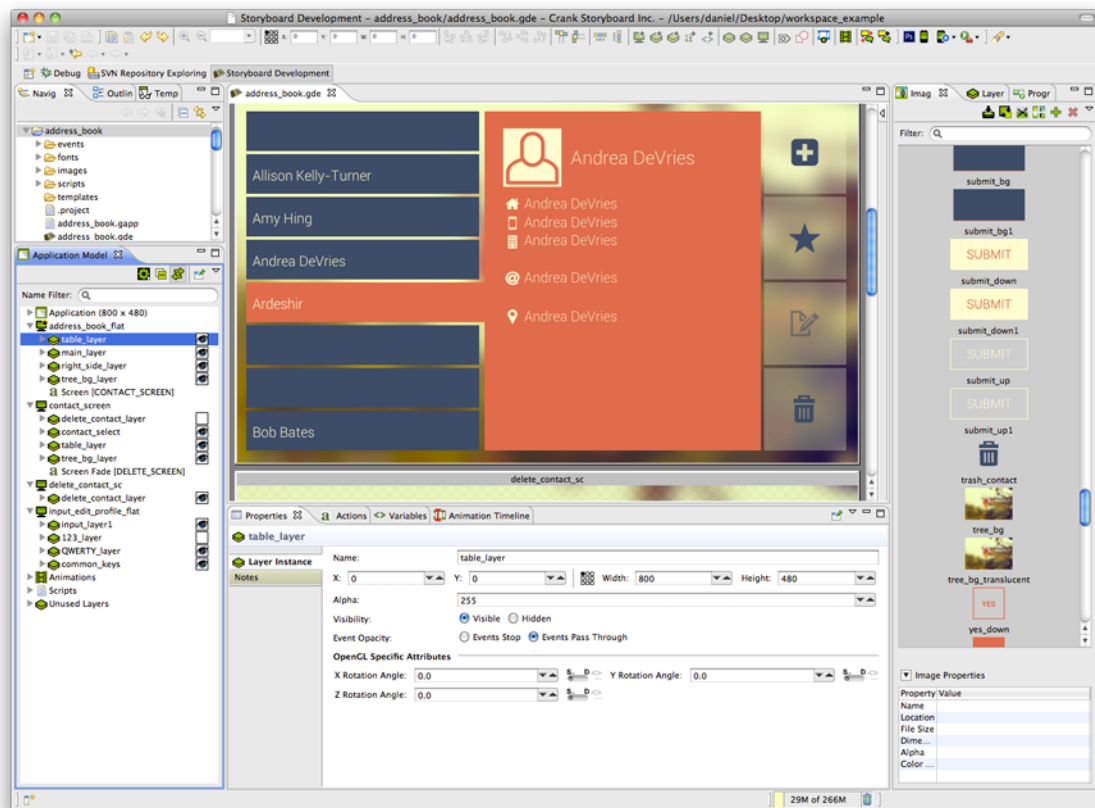
On the Import Storyboard Sample dialog, select the Sample to be used and click Finish.

Creating a Storyboard Project from a Sample



New Sample Project

The Sample Application opens in the Storyboard Development perspective.



Chapter 14. Working with Multiple Application Design Files

Storyboard Suite's collaborative features help multiple users develop applications faster. You can merge multiple files (*.gde) together and produce a single output during runtime. This tutorial explains how to create a project with multiple application files.

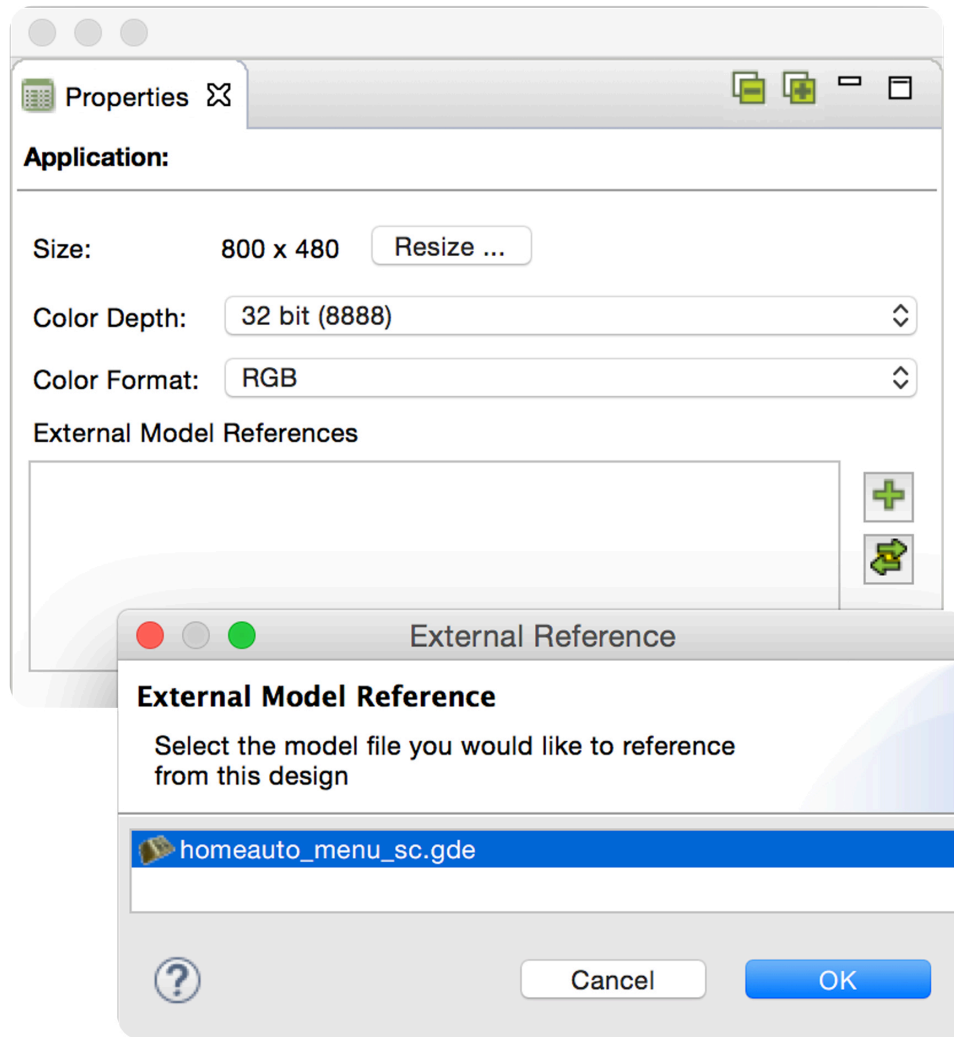
Creating a Project

A project with multiple application files starts the same as a single application file project. Select » **File » New » Storyboard Application** or create a new project using the **Photoshop Import** feature.

A project can accommodate multiple stand-alone applications that share project images and script resources. You can add an additional application file to an existing project in multiple ways:

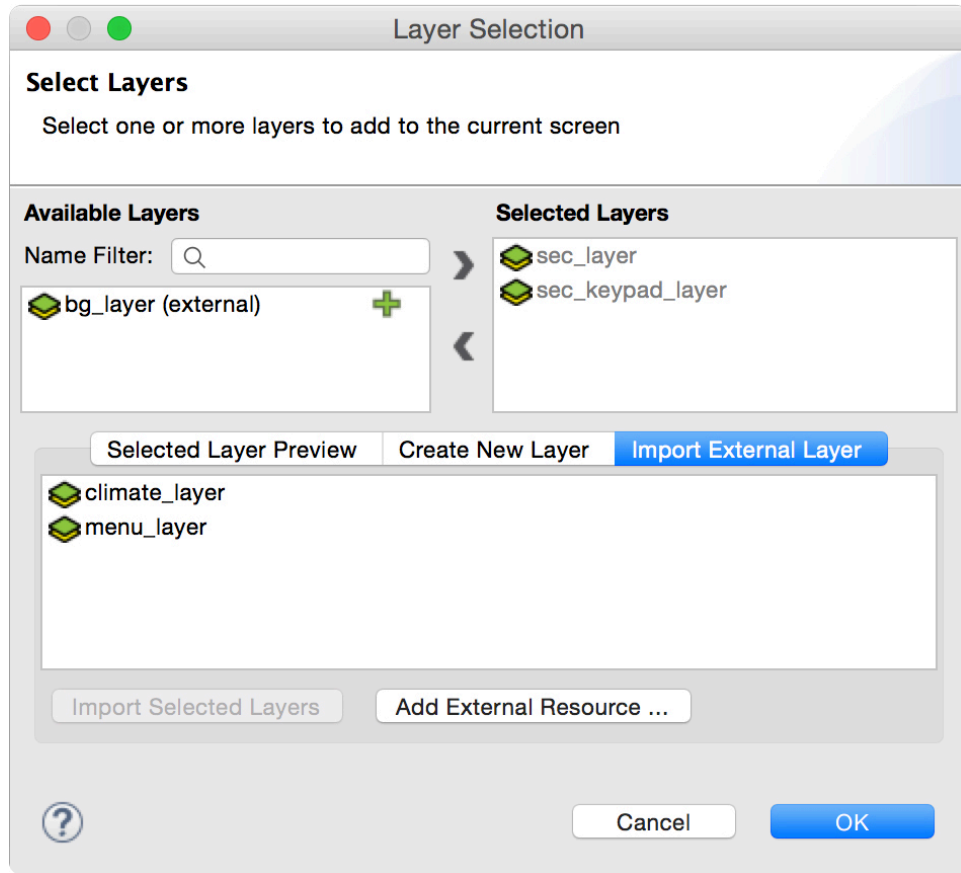
- Create a new file within an existing project by selecting » **File » New » Storyboard Application** and choose to create a **New Model in Existing Project**
- Create a new file using the Photoshop Import feature and select the option to import .psd **Into an Existing Project as a New File**.
- In the navigator view select and **copy an existing application**, then paste and rename the copied application file.

For a multiple application project to function as a whole, application files need to reference one another. Select the application from the Application Model View and in the Properties View, select **Add external model resource** button. In the next dialog select the .gde file that was created in **Step 2** and press **OK**.



To add layers from an external model to a local application:













- Select the **Import External Layer** tab and then select **Import Selected Layers**.
- Choose the layer(s) to add to the current screen. After adding an external layer, Storyboard Suite will recognize the external content and incorporate it to function like any other layer.



To create a transition from a source application screen to an external application screen, add actions to an application that perform a screen change.

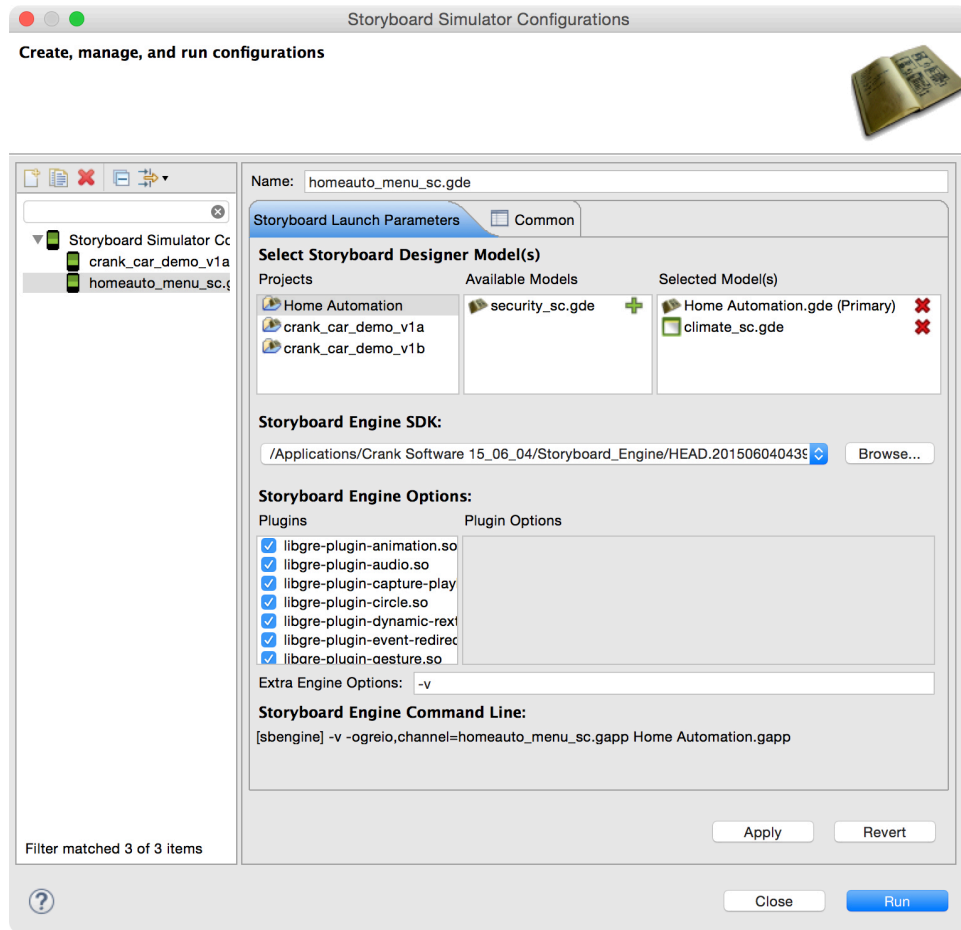
Step 3: Set Action Parameters

Screen Fade Parameters

screen	<input type="text" value="climate_sc (external)"/>		
duration	<input type="text" value="500"/>		
fps	<input type="text" value="30"/>		
rate	<input type="text" value="Linear"/>		
layers	<input type="text" value="All"/>		
direction	<input type="text" value="Top"/>		

Before launching a multi-file application with the simulator, the external .gde files have to be referenced in the Simulator Configurations dialog.

To manage the runtime configurations, select » **Run** » **Storyboard Simulator Configurations**. A list of available models that can be included in the runtime export used with Storyboard Engine is in the selected project folder. To apply changes, select » **Apply** and then » **Run**



If no conflicts occur within the selected applications, they are merged and converted into a single unified application at runtime. If conflicts exist, they must be resolved before the application can merge.

Resolving Conflicts

The application properties page provides an action to synchronize source content with referenced external content. Any differences are flagged as a conflict and the user is prompted to resolve the conflict based on the issue. Conflict types include:

Layers: If two or more layers have the same name their content needs to be identical.

Variables: Any application/global variables with the same name must have values that are the same.

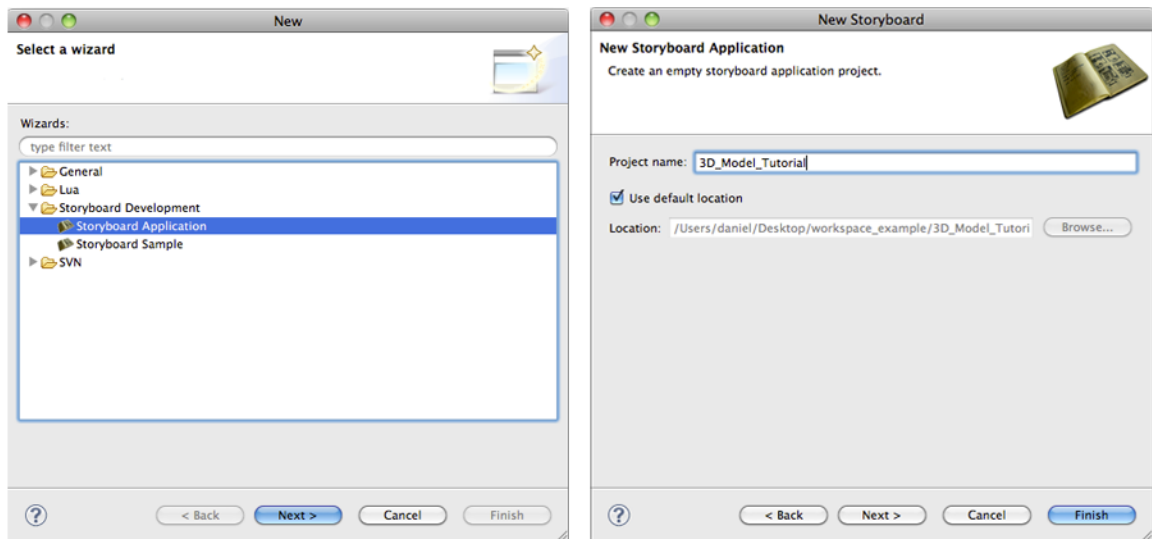
Animations: If two or more animations use the same name then the animation needs to be identical.

Screens: Screens from all applications are compared. Two or more screens with the same name prompt the user to resolve differences between the two.

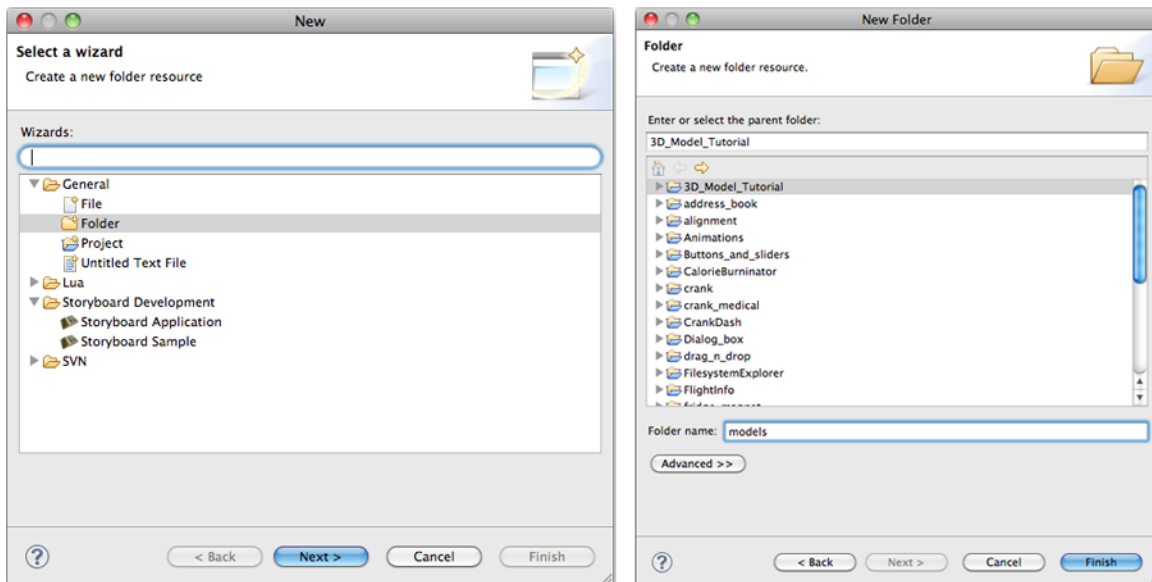
Chapter 15. Creating a 3D Model Application

This quick tutorial will show how to use the new 3D Model render extension with a new project

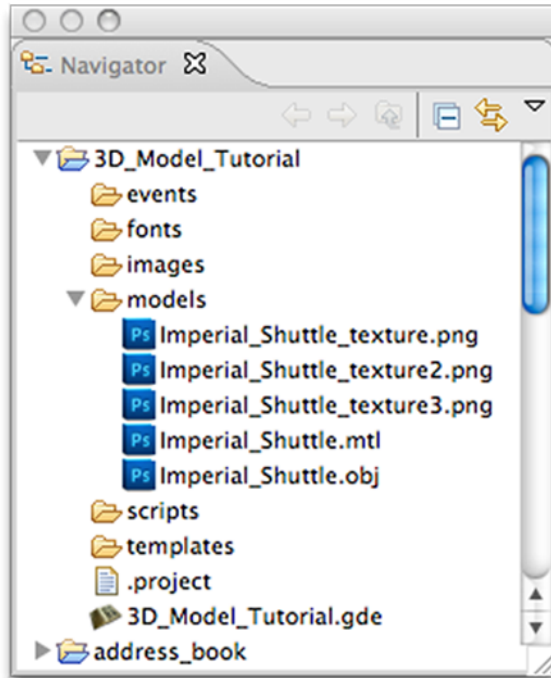
Select File > New > Storyboard Application.



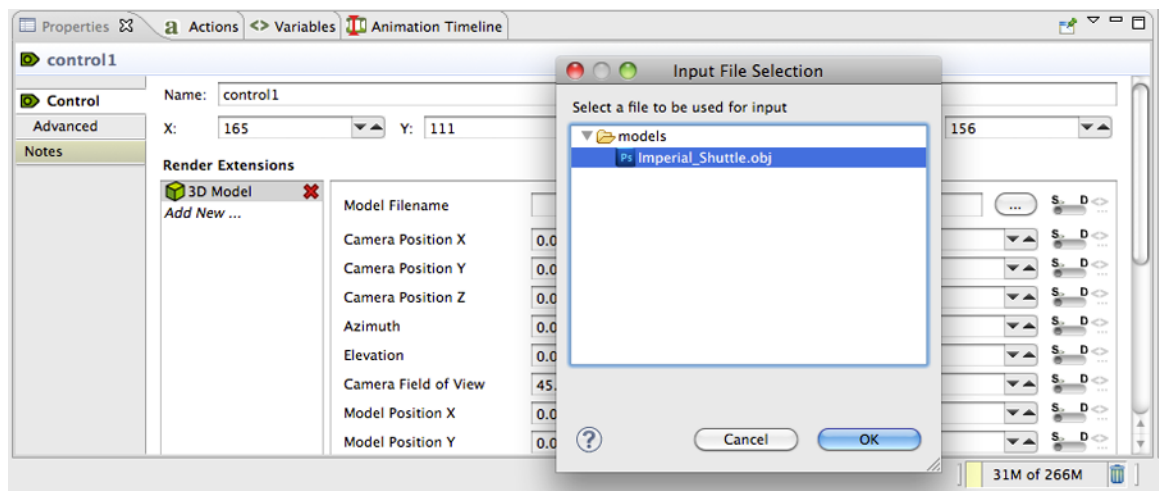
Select Project name and click Finish.



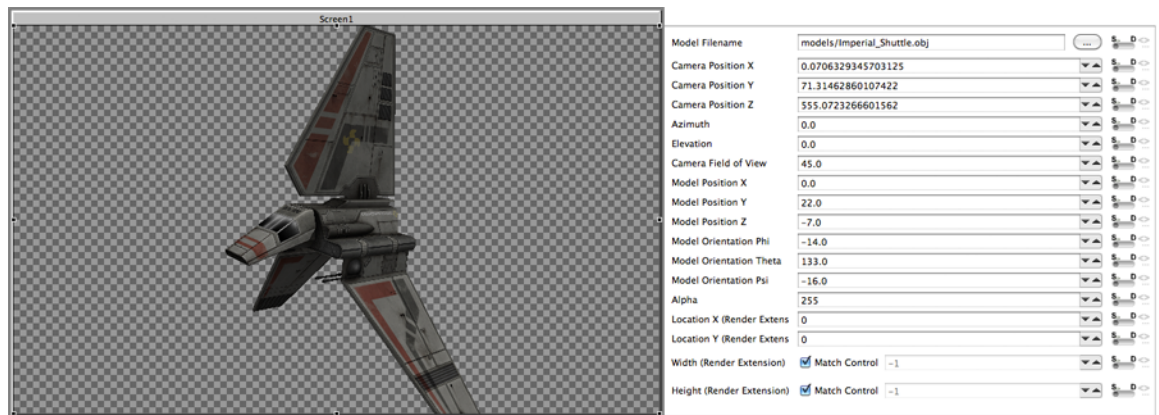
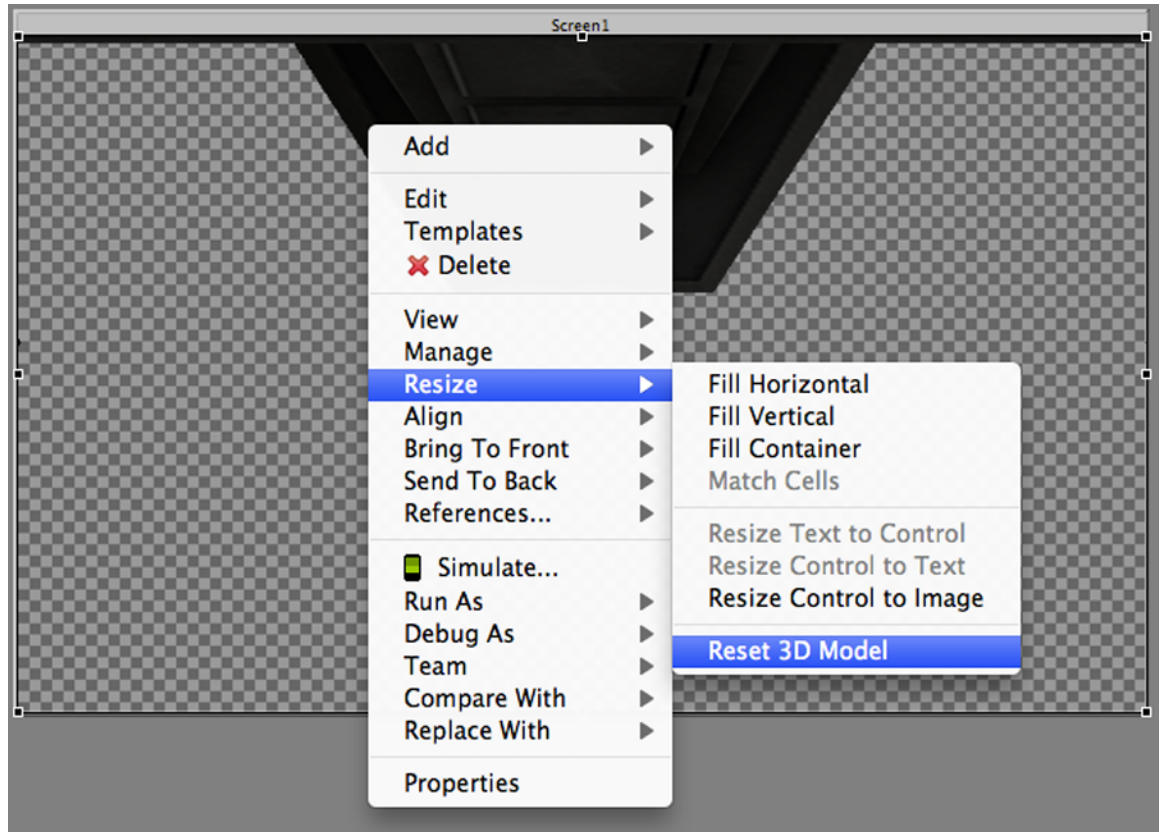
Once the empty project loads add a control with a 3D Model render extension to the screen. Nothing will show up in the render extension since a model file hasn't been selected.



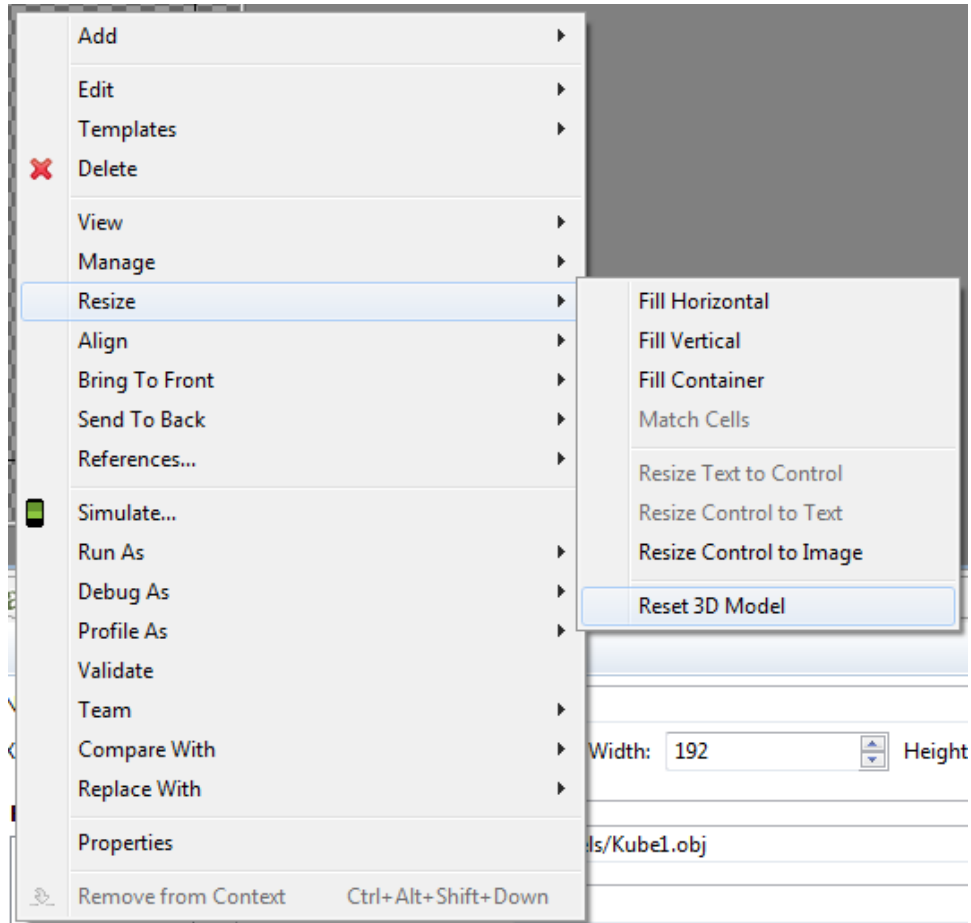
Copy an OBJ file anywhere within the project. (The models for this tutorial were placed in the user created models folder.)



Go to the 3D Model properties pane and either enter the location of the model or push the adjacent button to browse the project for any OBJ files.



Once the model loads it might not be immediately visible. Right click on the control and select Resize > Reset 3D Model. This sets some of the camera coordinates to make the model visible.



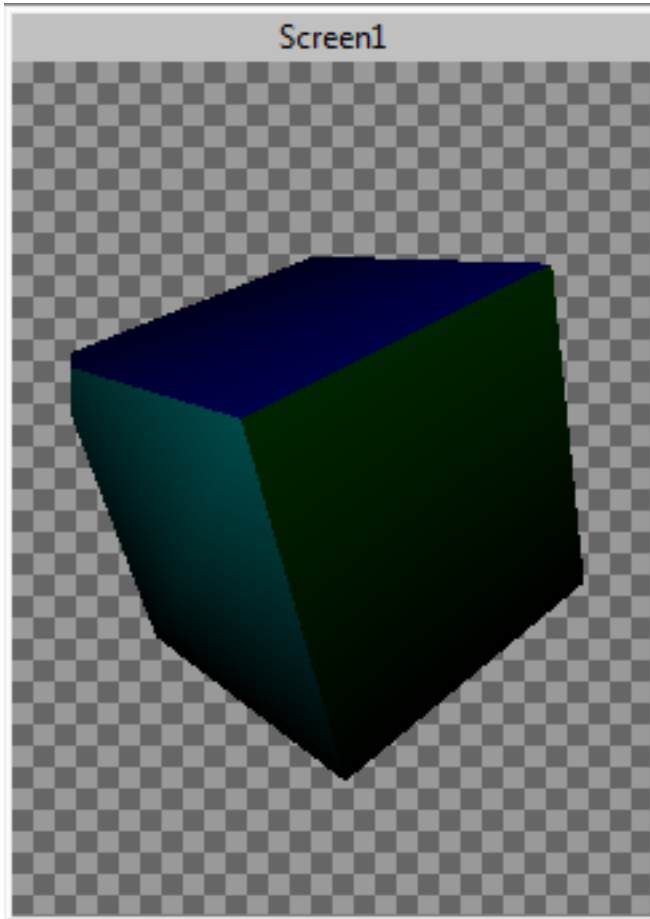
Further manipulation of the model properties may be needed in order to place the model in the desired position. The following are the definitions of each of the model's properties:

- **filename** - The name of the model to load.
- **camera_position_x** - The x position of the camera.
- **camera_position_y** - The y position of the camera.
- **camera_position_z** - The z position of the camera.
- **azimuth** - The rotation of the camera around the y axis in degrees.
- **elevation** - The rotation of the camera around the x axis in degrees.
- **camera_field_of_view** - The field of view the camera in degrees. The field of view specifies how much of visual sphere is mapped to the control. A larger field of view is equivalent to using a wide-angle lens on a camera, and a smaller field of view is equivalent to using a zoom lens.
- **model_position_x** - The x position of the model.
- **model_position_y** - The y position of the model.
- **model_position_z** - The z position of the model.

- **model_orientation_phi** - The rotation of the model around the x axis in degrees.
- **model_orientation_theta** - The rotation of the model around the y axis in degrees.
- **model_orientation_psi** - The rotation of the model around the z axis in degrees.

Model Filename	<input type="text" value="models/Kube1.obj"/>	...	<>
Camera Position X	<input type="text" value="0.00"/>	↑ ↓	<>
Camera Position Y	<input type="text" value="0.00"/>	↑ ↓	<>
Camera Position Z	<input type="text" value="0.00"/>	↑ ↓	<>
Azimuth	<input type="text" value="0.00"/>	↑ ↓	<>
Elevation	<input type="text" value="0.00"/>	↑ ↓	<>
Camera Field of View	<input type="text" value="45.00"/>	↑ ↓	<>
Model Position X	<input type="text" value="0.00"/>	↑ ↓	<>
Model Position Y	<input type="text" value="0.00"/>	↑ ↓	<>
Model Position Z	<input type="text" value="-0.50"/>	↑ ↓	<>
Model Orientation Phi	<input type="text" value="30.00"/>	↑ ↓	<>
Model Orientation Theta	<input type="text" value="30.00"/>	↑ ↓	<>
Model Orientation Psi	<input type="text" value="30.00"/>	↑ ↓	<>

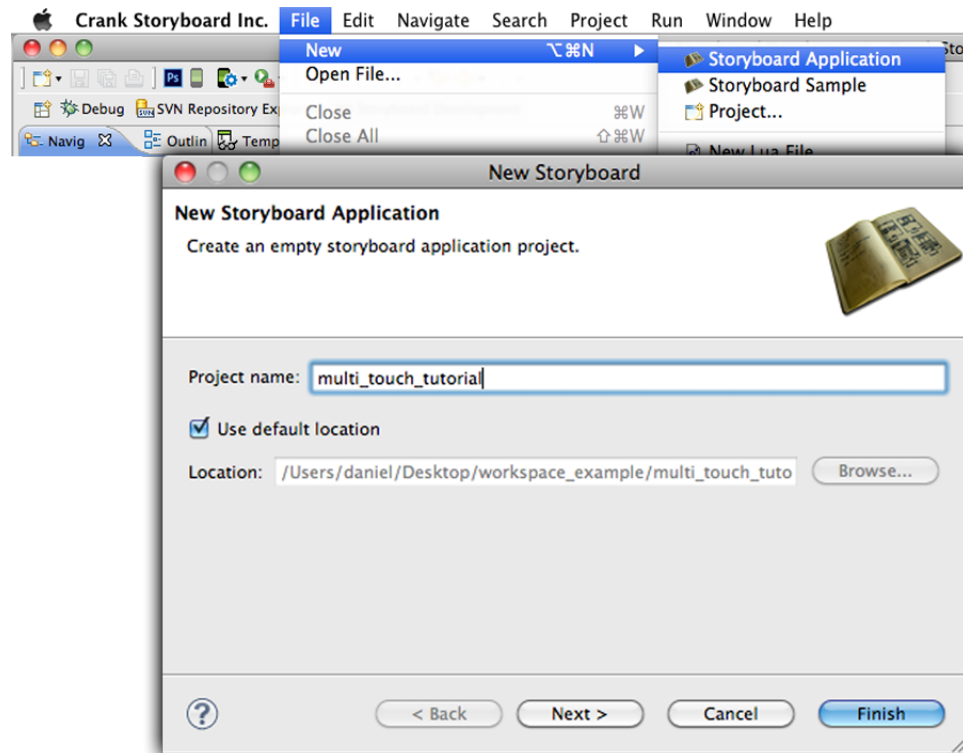
After setting the desired properties the model will automatically reposition itself.



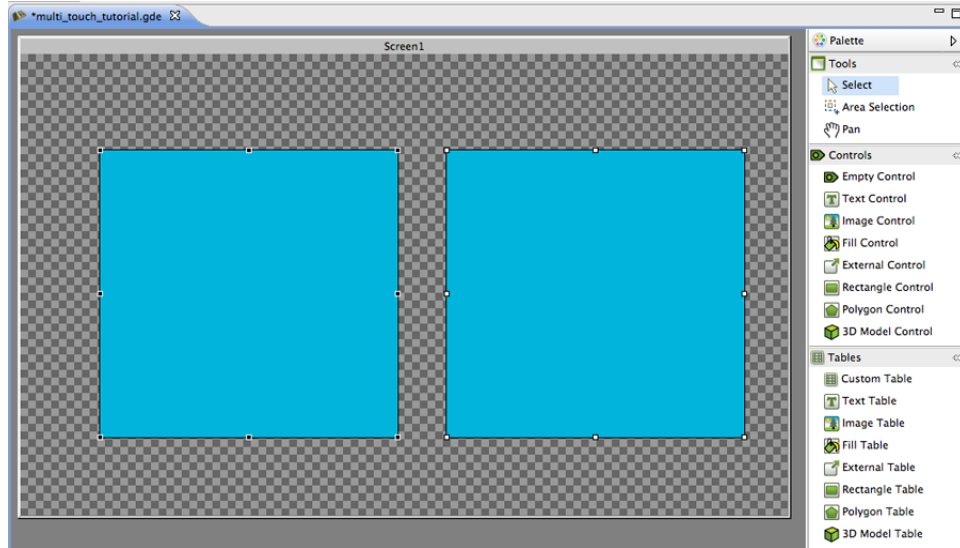
Chapter 16. Creating a Multi-Touch Application

This quick tutorial will show how to create a Storyboard Application that uses Android's multi-touch support.

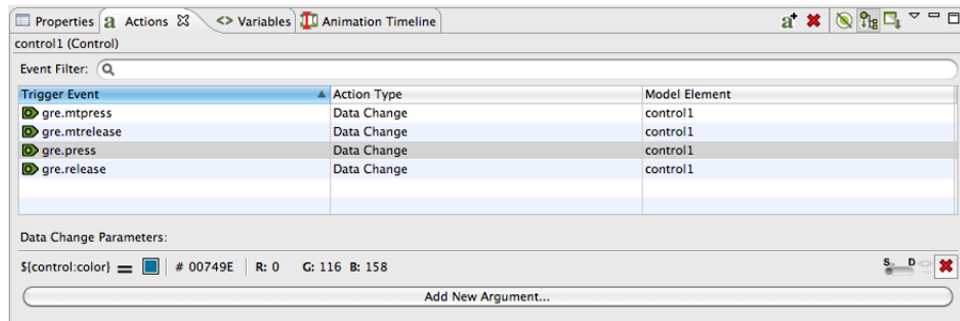
Select File > New > Storyboard Application.



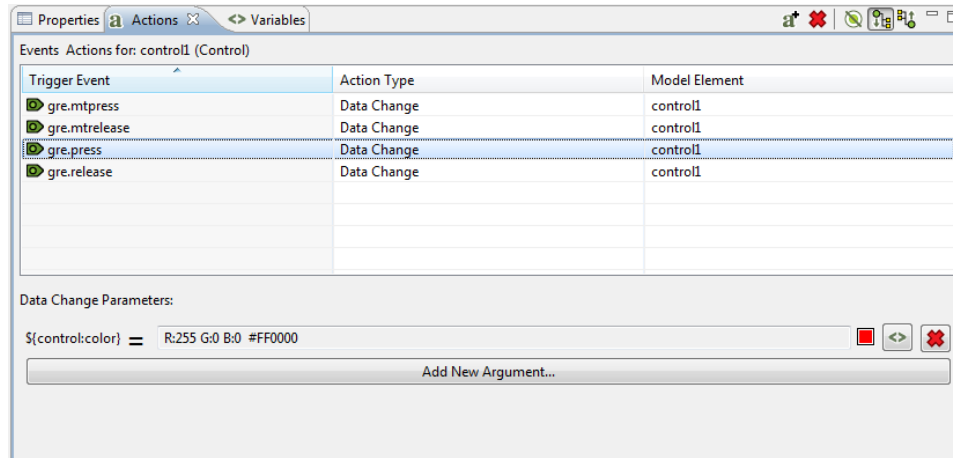
Select Project name and click Finish.



Once the empty project loads, add two controls each with a Fill render extension. Along with the Fill add a variable to both of the controls that stores a color. Set both of the Fills' color argument to that variable. (In this case blue is used as the default).



Time to add actions. In order for the multi-touch to work correctly one must add actions for both single touch and multi-touch. The single touch actions will be used when there is only one touch point, while the multi-touch events will be used when there are two or more touch points. In this example we'll use the gre.mtpress/gre.press and gre.mtrelease/gre.release events with a DataChange action where we change the color of the color variable for each of the controls (this way when you press and release the control its color changes). This example uses red for when the control is pressed and blue for when it's released.



Now you can export the example application as an Android APK and run it on a device. Press either of the controls and watch their color change. This is all that's required to getting multi-touch events to work with Storyboard on Android.

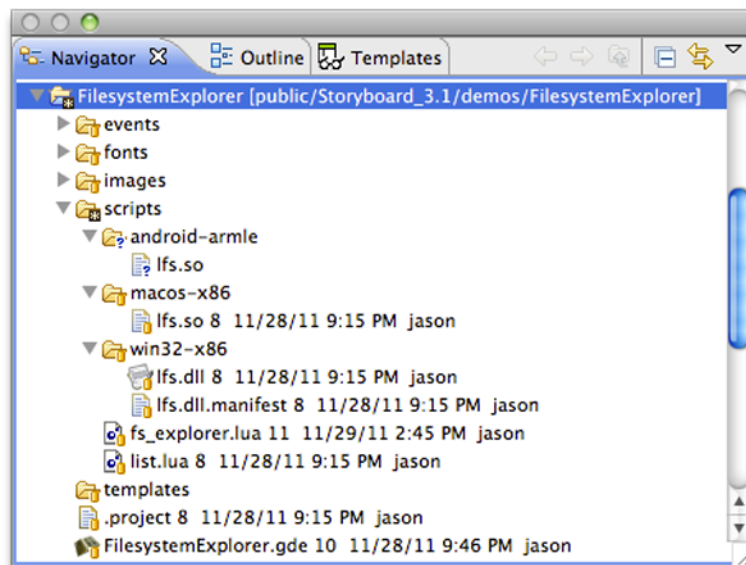
Enabling Multi-Touch

The `-ogesture,mode=multi` option needs to be passed to the Storyboard Runtime Options when exporting an Android apk from Storyboard Designer.

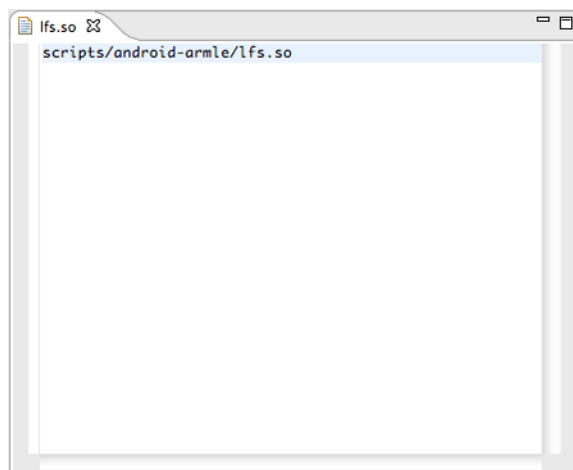
Chapter 17. Adding Extra Libraries for Android

Sometimes a user creates a Storyboard app that requires a library that isn't included with the Storyboard Runtime. When exporting for Android we need to tell the exporter which libraries to preload. We do this by giving the exporter a text file with a list of libraries. Make sure that the paths to these libraries are relative to the Storyboard app's directory. As well, the order of the libraries in the list determine the order they get loaded in, therefore if one of the libraries has a dependency on another library make sure to have the dependent library higher in the list.

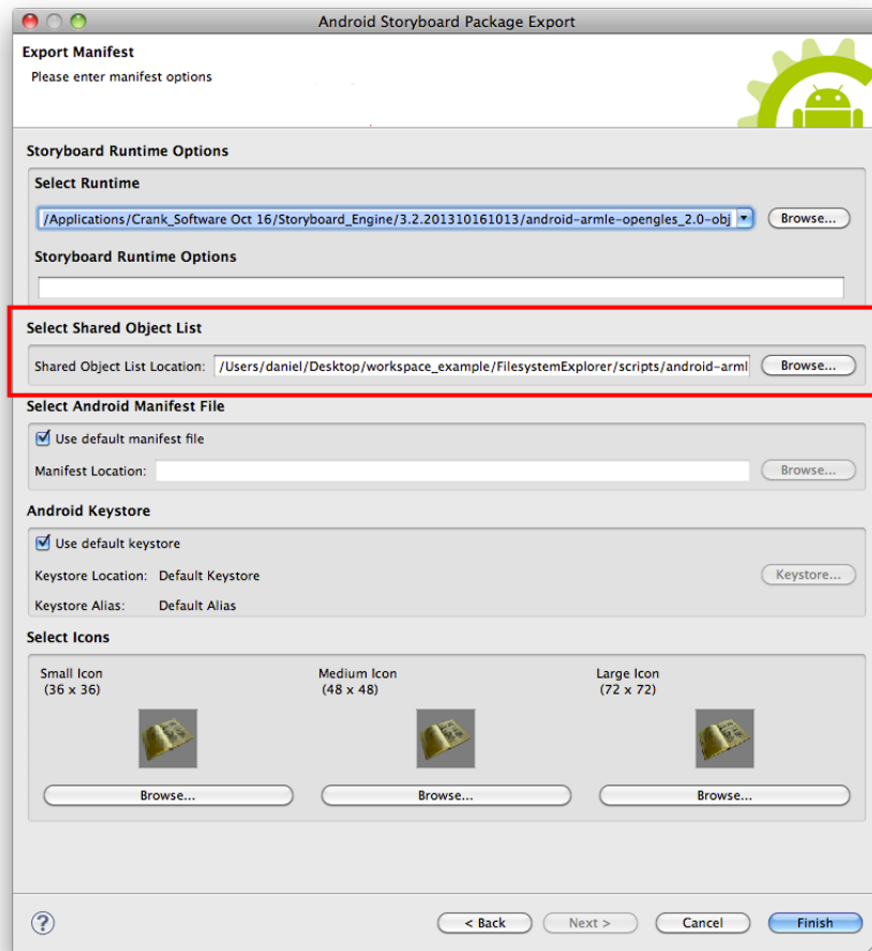
The example we'll use is modifying the FilesystemExplorer app from the Crank Software public repository. In order for this app to work on android we'll need to include the LuaFileSystem module that's been compiled for android (the lib's name is lfs.so). We'll add this file in scripts/android-armle.



Create a text file, which we'll call user_libs.txt, with the following contents:



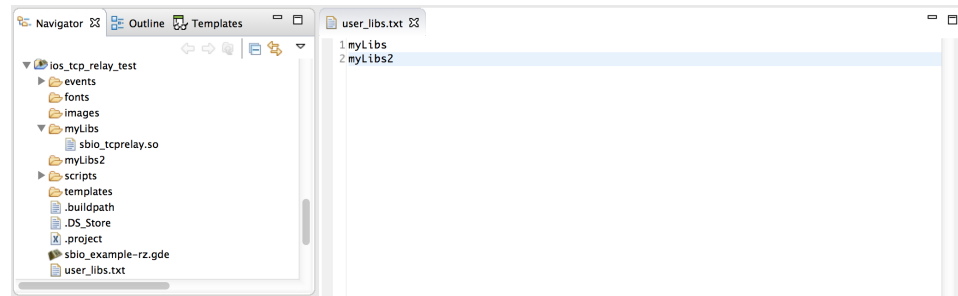
When exporting the app make sure to include the path to this file and hit finish.



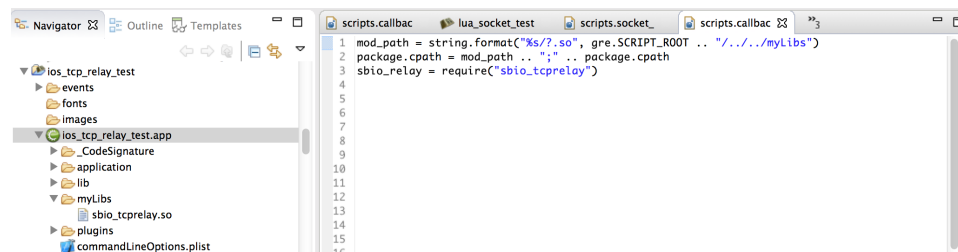
Now the app can make use of the functionality included in the new library.

Chapter 18. Adding Extra Libraries for iOS

Users can also add additional libraries that aren't included in the runtime to an exported iOS application. Similar to Android, Storyboard needs to know about them beforehand so that we can pass them through the code signing process. To do this, create a text file called `user_libs.txt` that contains any folder names as strings that lead to added libraries that we need to check for signing. The folders should be placed at the project root. This text file should also be placed in the root of the project. A valid setup looks like this:



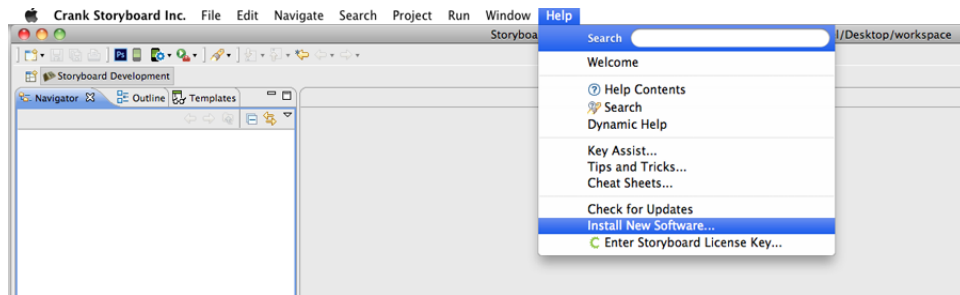
Now that your libraries have been signed and are available for use, if you need to access them through Lua you can do so using the helper variable `gre.SCRIPT_ROOT`. You can look at the generated `.app` file to determine the script root relative path to your library folders, which will be located at the root of the `.app` folder. That path needs to then be appended to the `package.cpath` variable in Lua using a semicolon. In the above example, it would look like this:



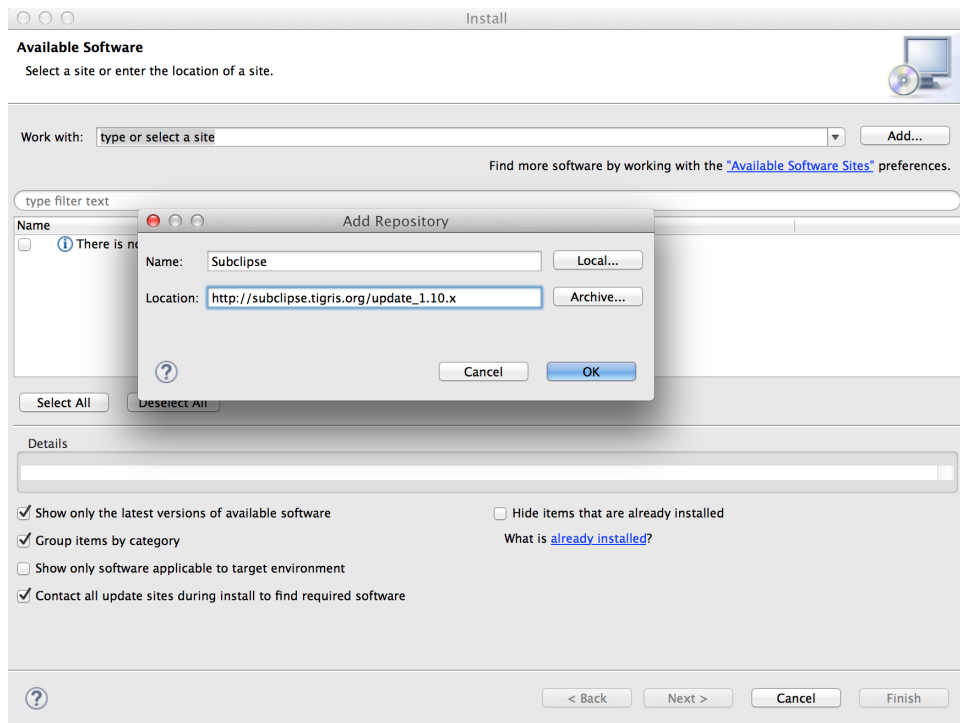
Chapter 19. Crank Public SVN

Installing Subclipse and connecting to the Crank Software public repository

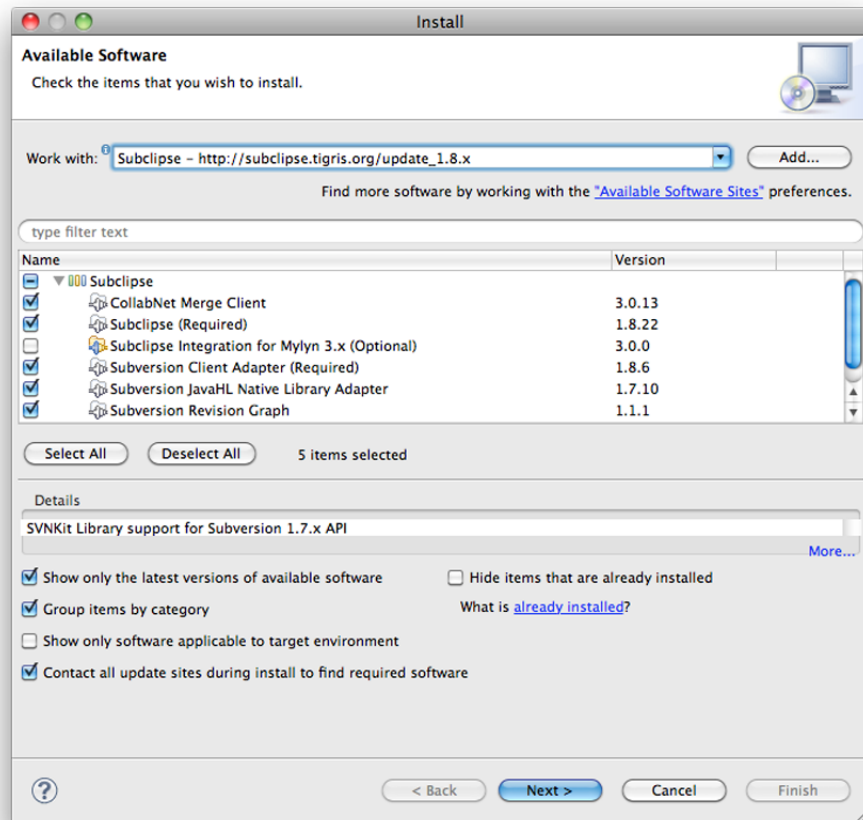
Before we are able to start using the demos from the public Crank code repository we will first need to install a SVN client. To do so in Storyboard Designer we go to Help -> Install New Software.



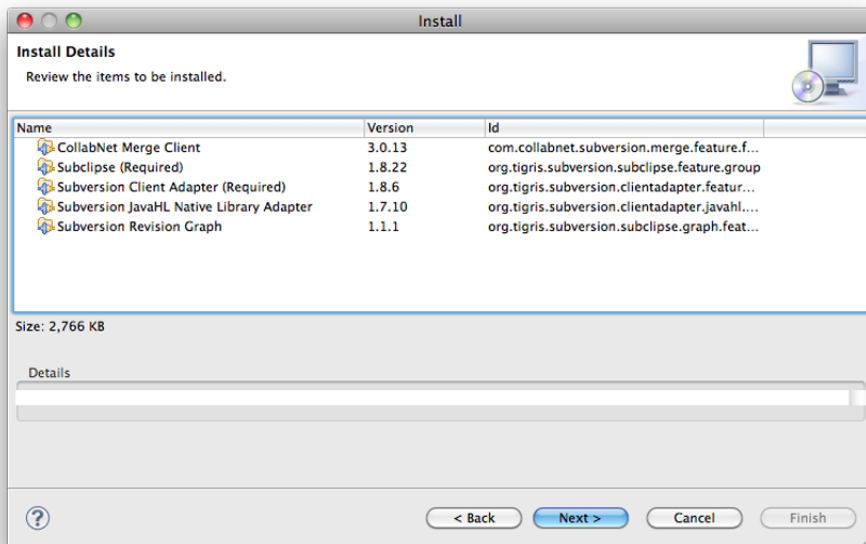
Here at Crank we use Subclipse. Click on Add and for Name you can enter anything you please. To keep it simple we will use Subclipse. For Location enter http://subclipse.tigris.org/update_1.10.x. Click OK.



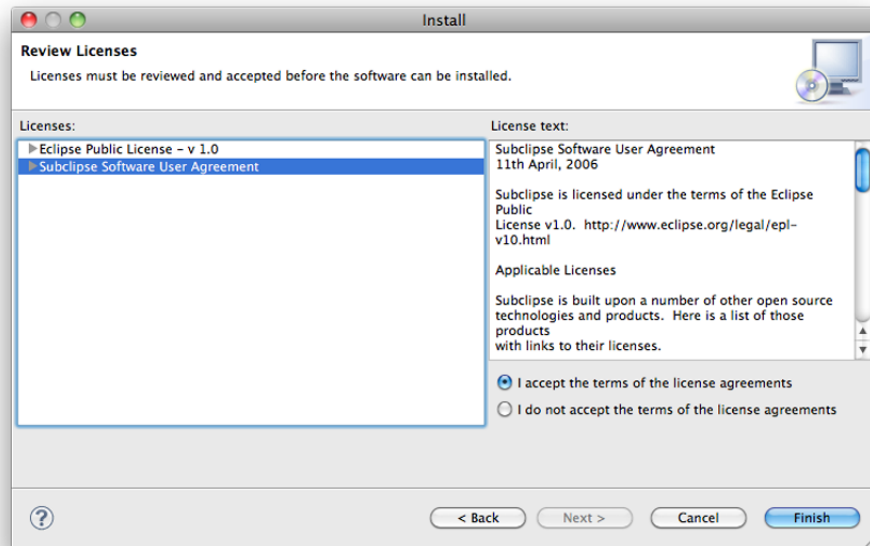
After you click OK you will be presented with a list of software to install. There is no need for Mylyn so you can uncheck it. Click Next.



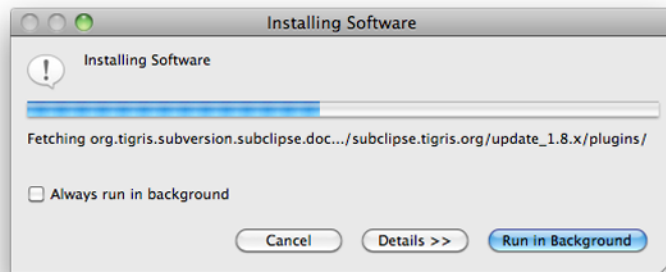
Details of the software to be installed. Click Next.



Accept license agreements. Click Finish.



Software installation progress.



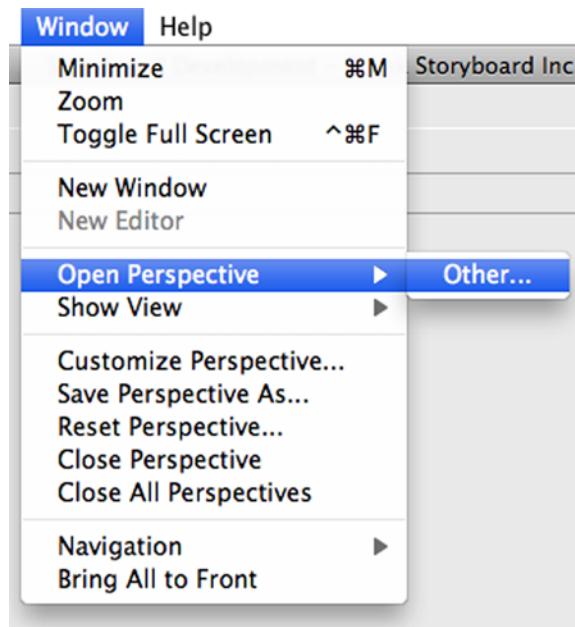
Generic warning that the software being installed is not signed. Click OK.



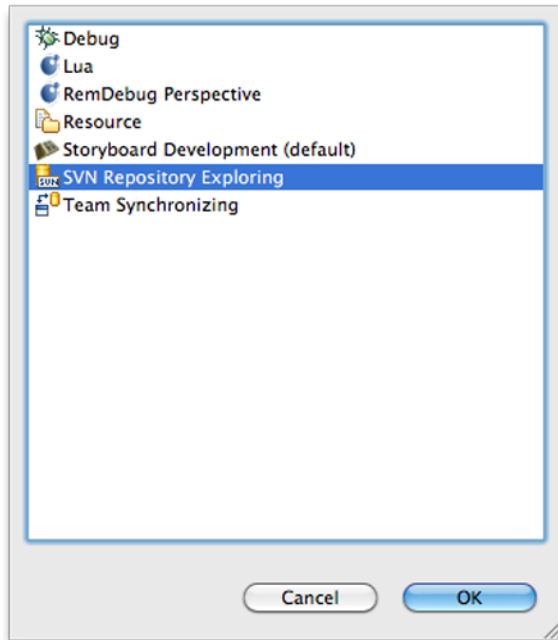
After the Subclipse svn client is installed you will need to restart Storyboard Designer. Click Restart Now.



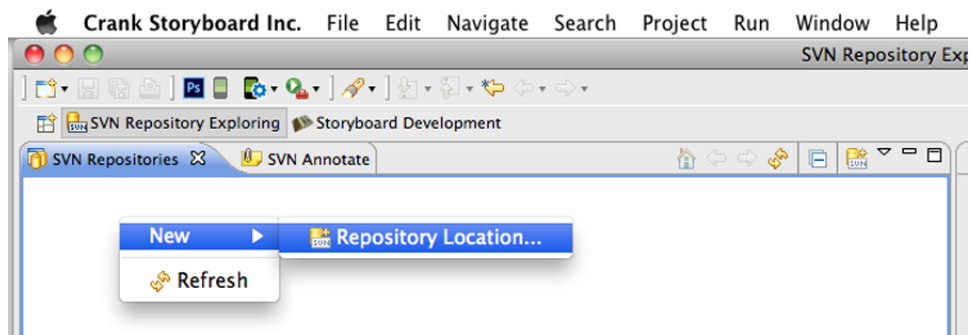
Now that Subclipse is installed we need to go to that perspective to add a repository.



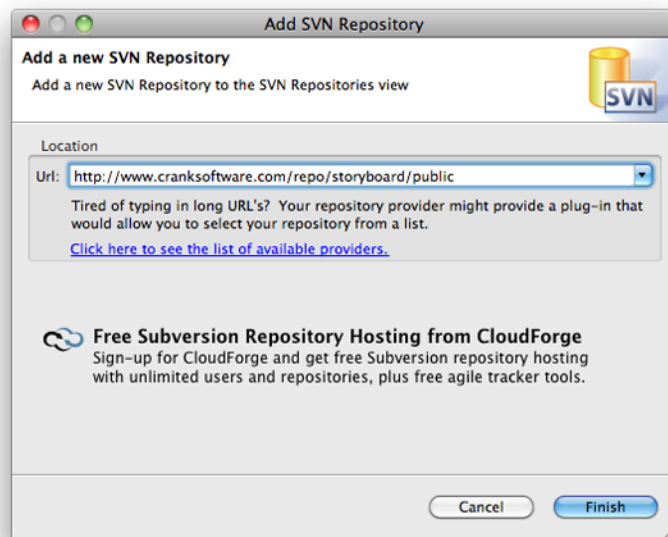
Select SVN Repository Exploring. Click OK.



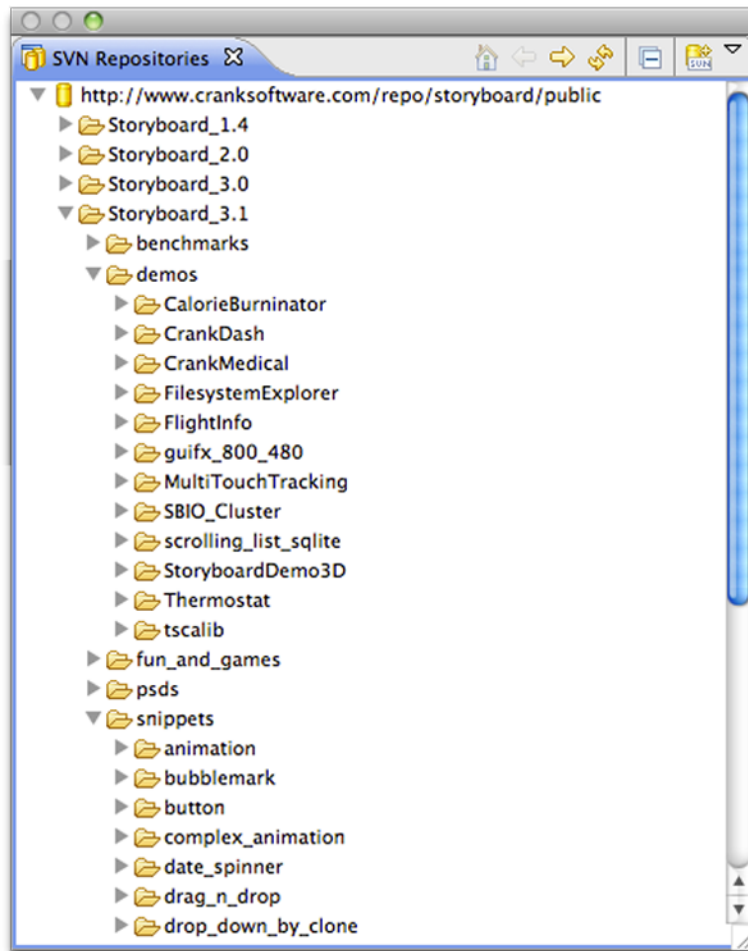
To add a repository we simply right click in the SVN perspective window and select New -> Repository Location.



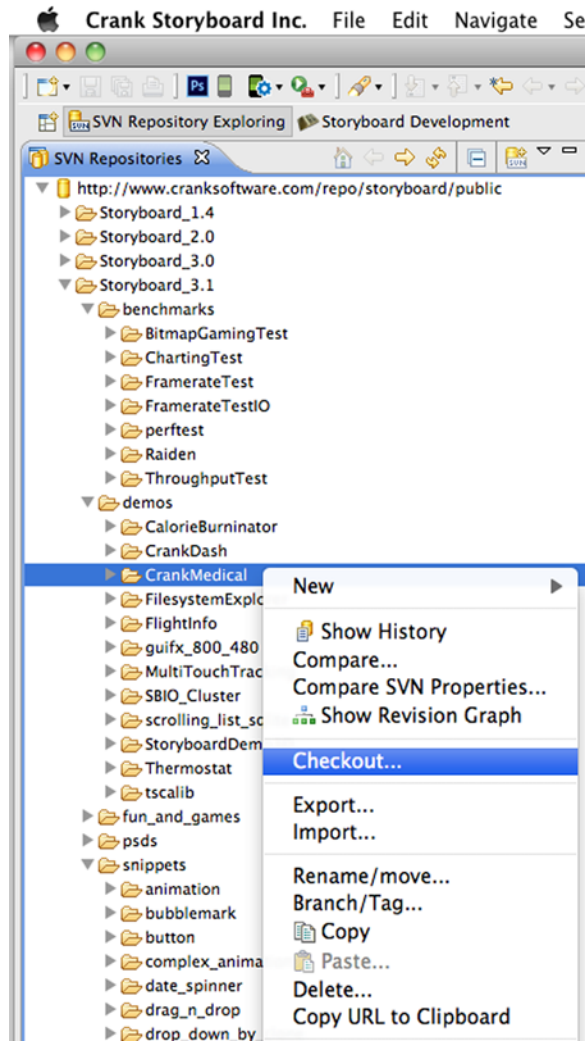
Enter the Crank public code repository URL <http://www.cranksoftware.com/repo/storyboard/public> Click Finish. When you are prompted for a username and password use storyboard and crankrocks.



You are now connected to the Crank Public Repository. By expanding the directories you can see the different demos available for checkout.



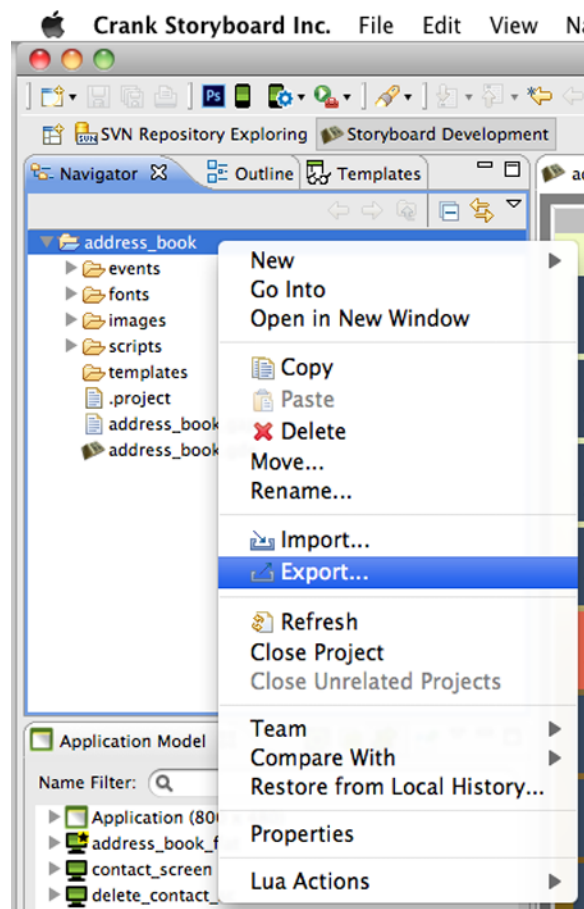
To checkout from the repository you right click on the demo and select Checkout. Once completed click on the Storyboard Development tab to see the application in your workspace.



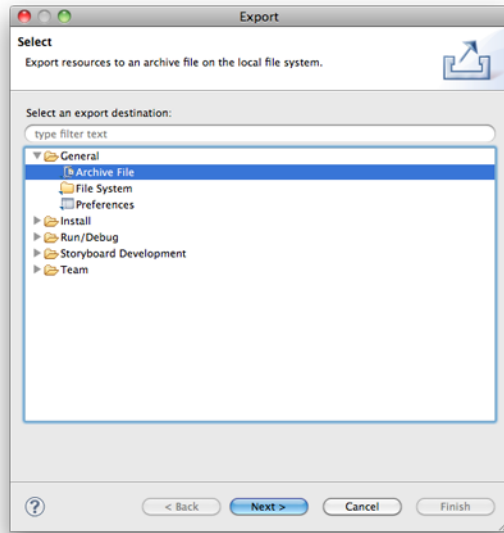
Chapter 20. Exporting a Storyboard Project

Storyboard Designer gives you the ability to export your project for either archiving, sharing or demo purposes. Here are a couple of easy steps showing you how to do that.

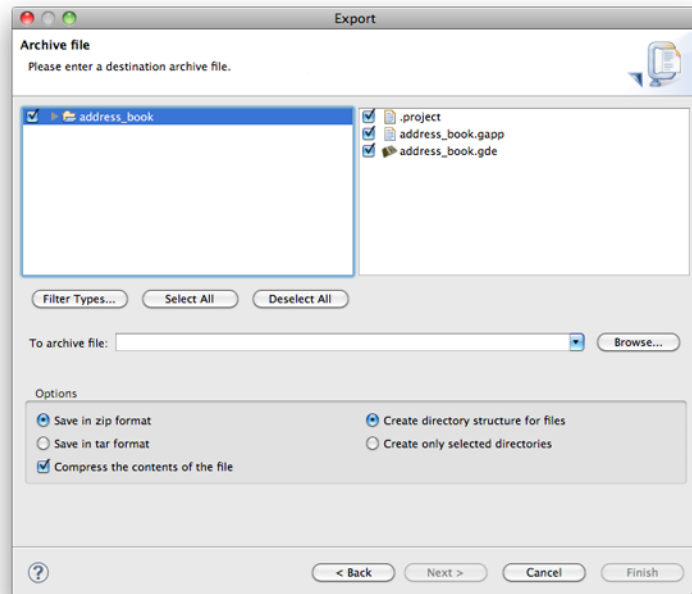
When your Storyboard Project is complete and you are ready to export, right click on the project folder and select Export.



Next you will be presented with the Export Selection dialogue. Expand General by clicking on the triangle to the left of the folder. Select Archive File and then click Next.



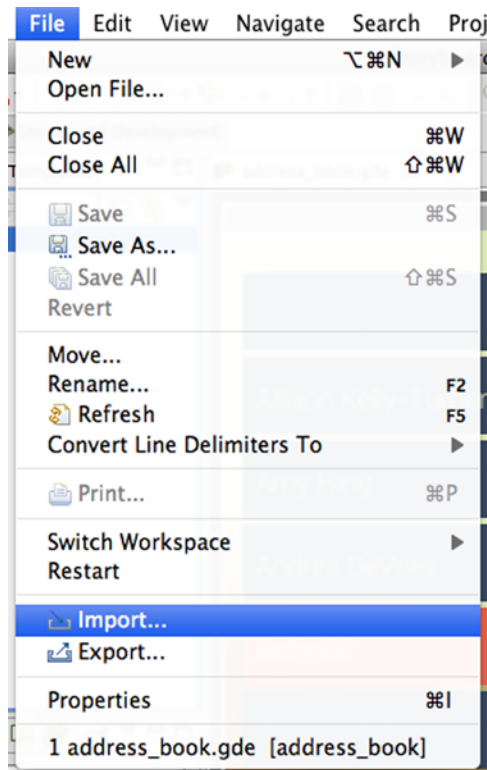
Next you will see the Export Archive file box. Here you will see all the folders and files that will be included in the archive you are about to create. Browse to the location where you want to save and then provide a name for your archive. Review and verify your Options and then click Finish.



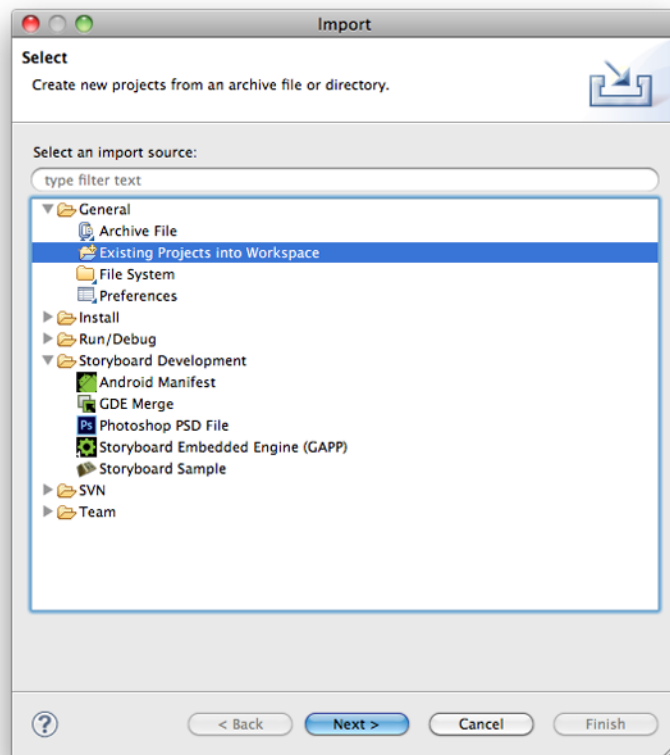
Chapter 21. Importing a Storyboard Project

Here are a couple of easy steps to import a Storyboard Project Archive into Storyboard Designer.

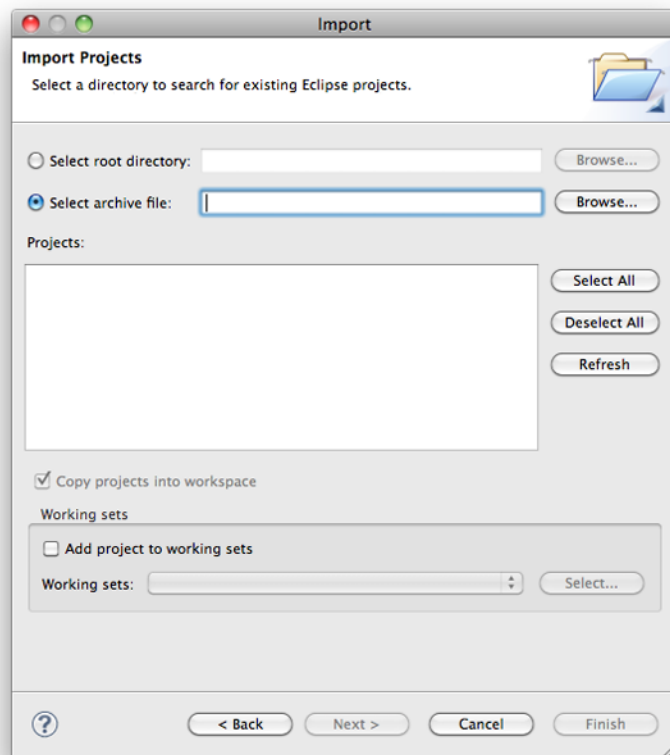
Right click within the Navigator view or left click File from the top menu and select Import.



Next you will be presented with the Import Selection dialogue. Expand General by clicking on the triangle to the left of the folder. Select Existing Projects into Workspace and then click Next.



Check the Select archive file: radio button and then Browse to the Storyboard Project Archive. Click Finish to import the archive into your workspace.



Part III. Storyboard Target Tutorials

Table of Contents

22. Linux	262
TI AM355 Starter Kit	262
Step 1: Importing A Storyboard Sample	262
Step 2: Exporting A Storyboard Application	264
Step 3: Selecting The Storyboard Embedded Engine	266
Step 4: Configuring The Target Platform	267
Step 5: Running The Storyboard Application	267

Chapter 22. Linux

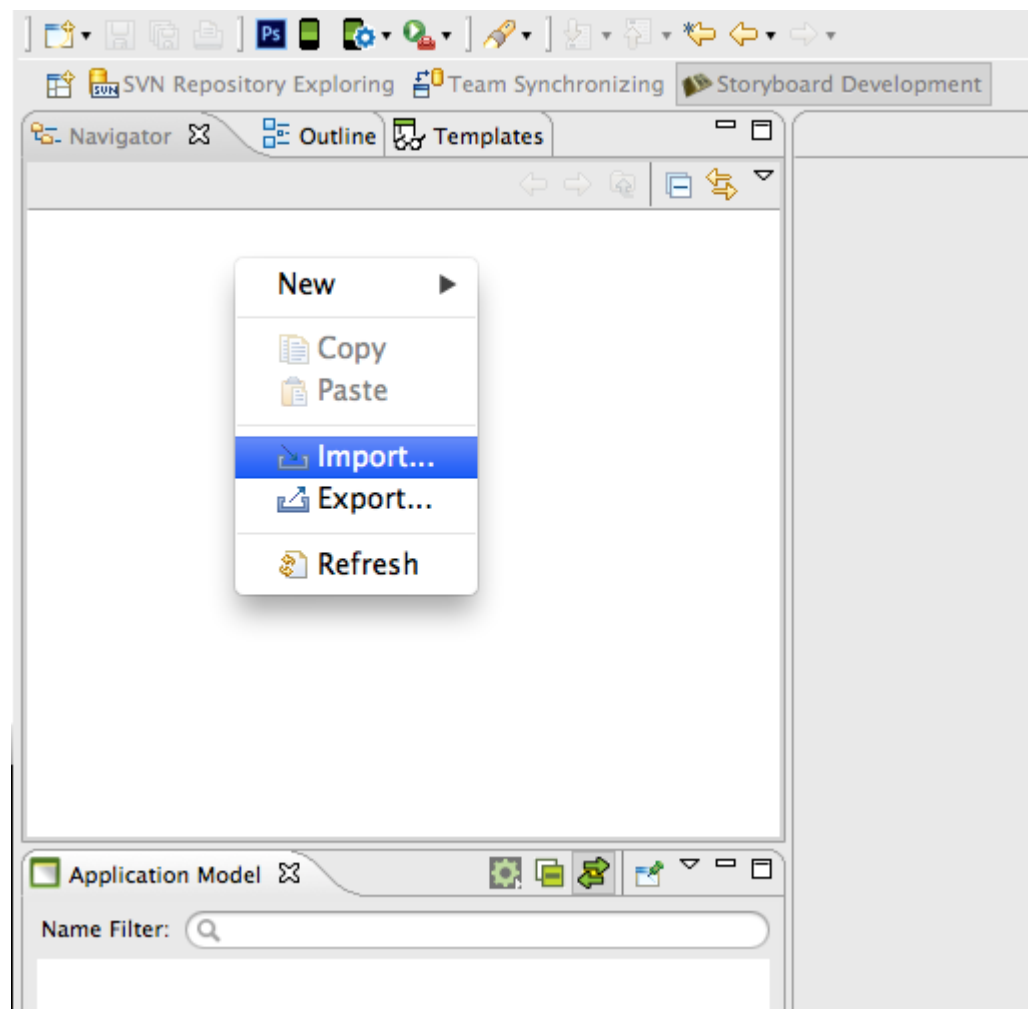
TI AM355 Starter Kit

The AM335x Starter Kit (EVM-SK) provides a stable and affordable platform to quickly start evaluation of Sitara™ ARM® Cortex™-A8 AM335x Processors (AM3352, AM3354, AM3356, AM3358) and accelerate development for smart appliance, industrial and networking applications. It is a low-cost development platform based on the ARM Cortex-A8 processor that is integrated with options such as Dual Gigabit Ethernet, DDR3 and LCD touch screen.

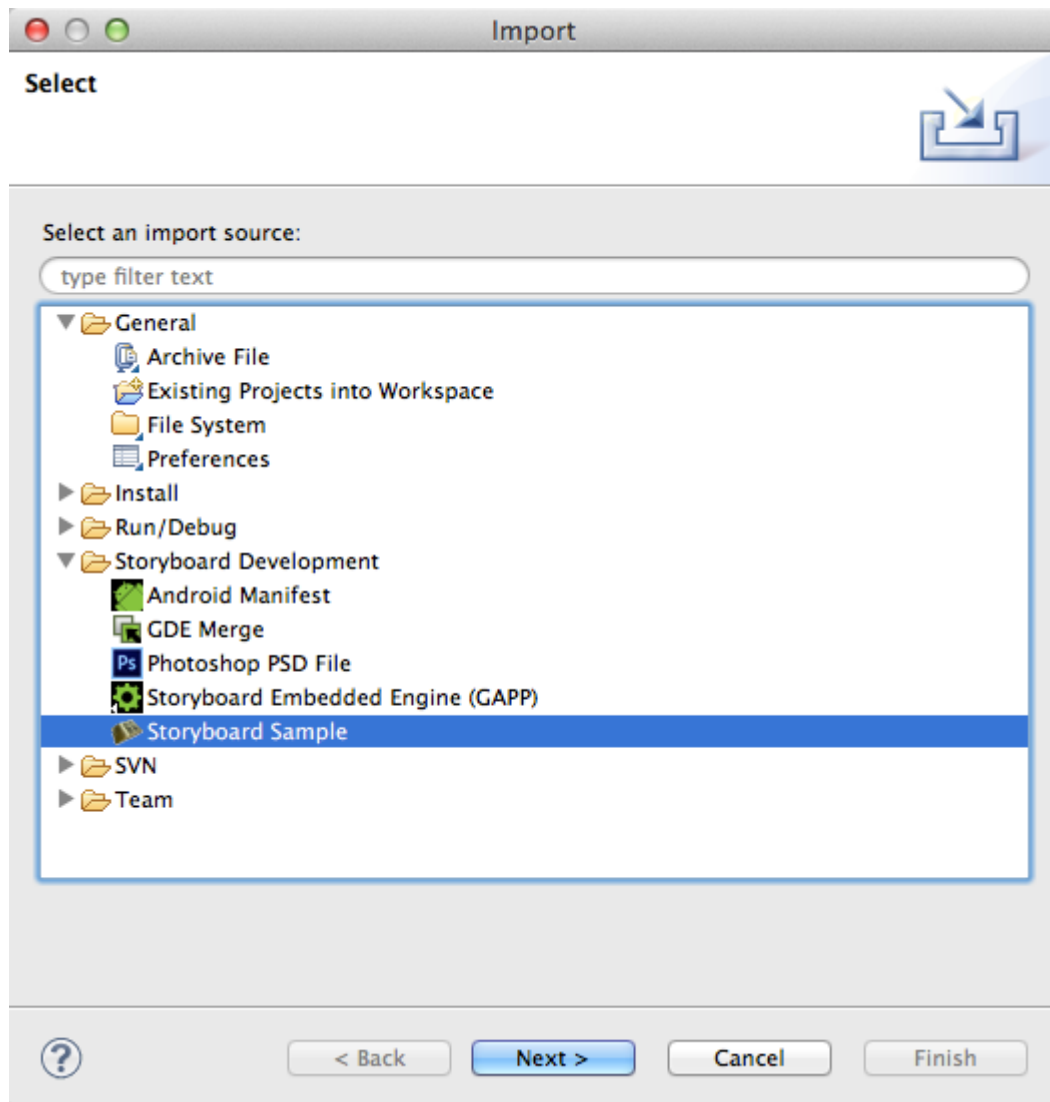
The following steps describe how to take a Storyboard sample and place it down on the TI AM335 board. It is assumed that the TI AM335 board has been setup correctly running Linux and that it is connected via a serial cable to either a laptop or desktop computer.

Step 1: Importing A Storyboard Sample

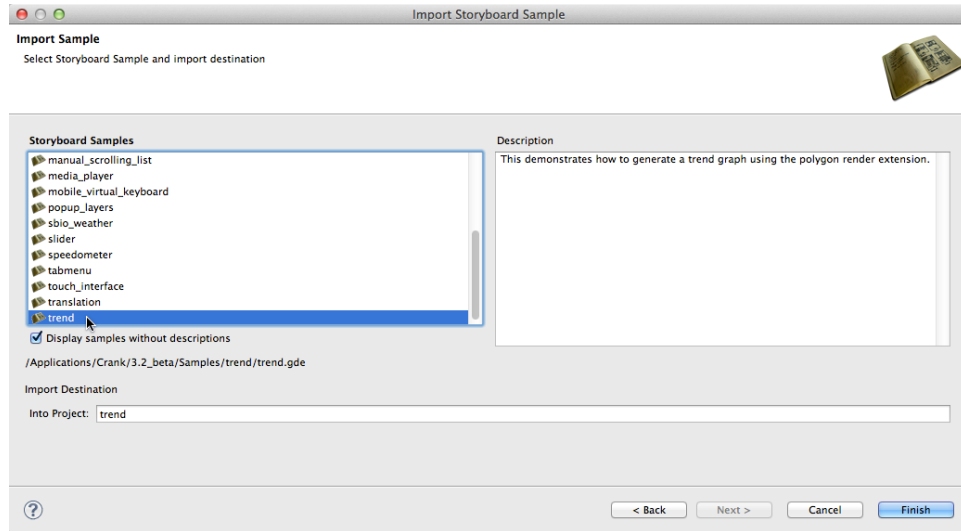
To import a Storyboard sample the user can right-click within the navigator view and select Import.



In the Select dialog expand the Storyboard Development folder and select Storyboard Sample.

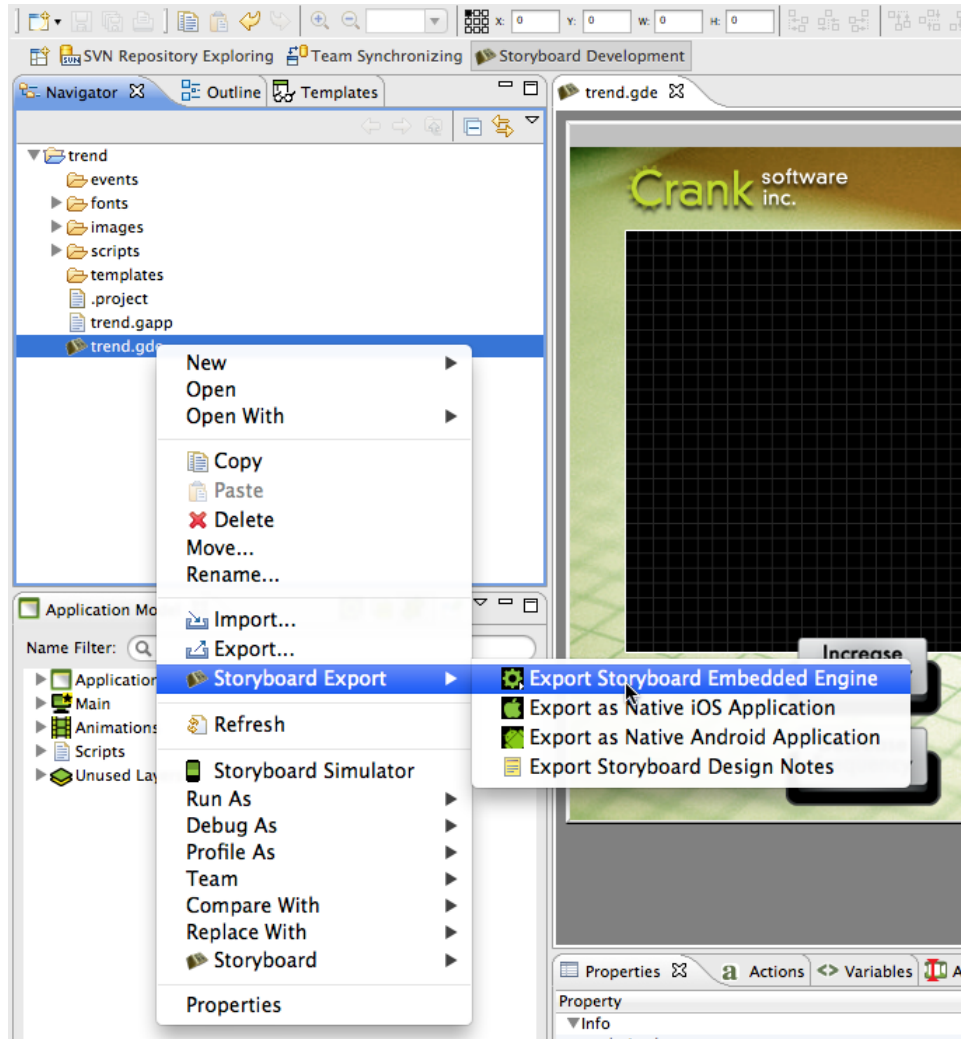


In the Import Sample dialog any sample can be used but for the purposes of this example the Trend sample has been selected.

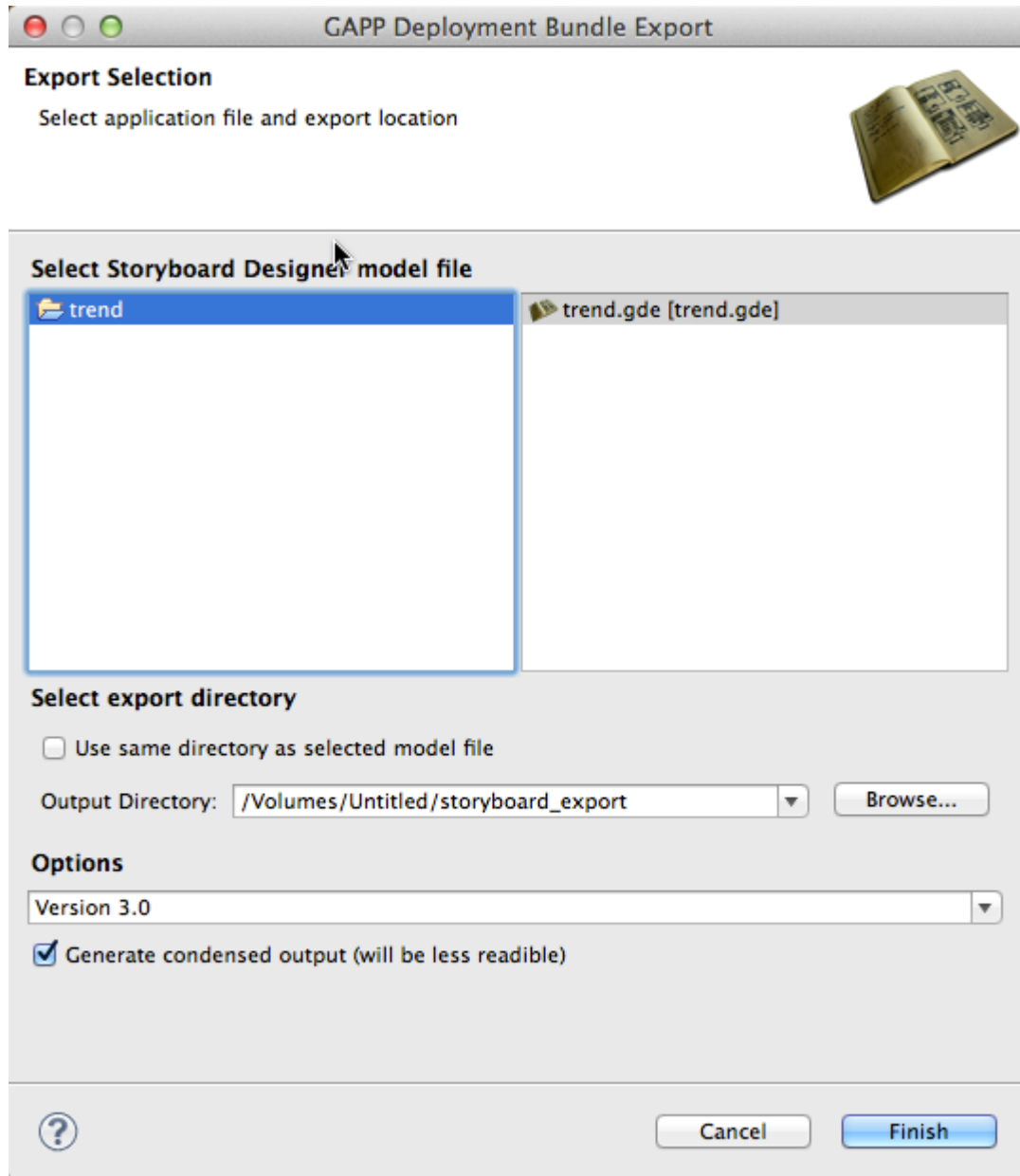


Step 2: Exporting A Storyboard Application

Once a Storyboard application is complete and ready to be placed on a target platform, it needs to be exported from Storyboard Designer in a format that the Storyboard Embedded Engine can use. Right-click the Storyboard applications's .gde file, located in the project folder in the Navigation View, and select Storyboard Export -> Export Storyboard Embedded Engine.



The Export Selection dialog is used to tell Storyboard Designer where to export the selected Storyboard Application. Leaving the "Use same directory as selected model file" option checked will place the data bundle, for Storyboard Embedded Engine, into the applications project directory. Deselecting the option enables the data bundle to be placed else where. i.e. - usb drive, NFS mount etc. For the purpose of this example the Storyboard application will be exported to a usb drive attached to the laptop/desktop.

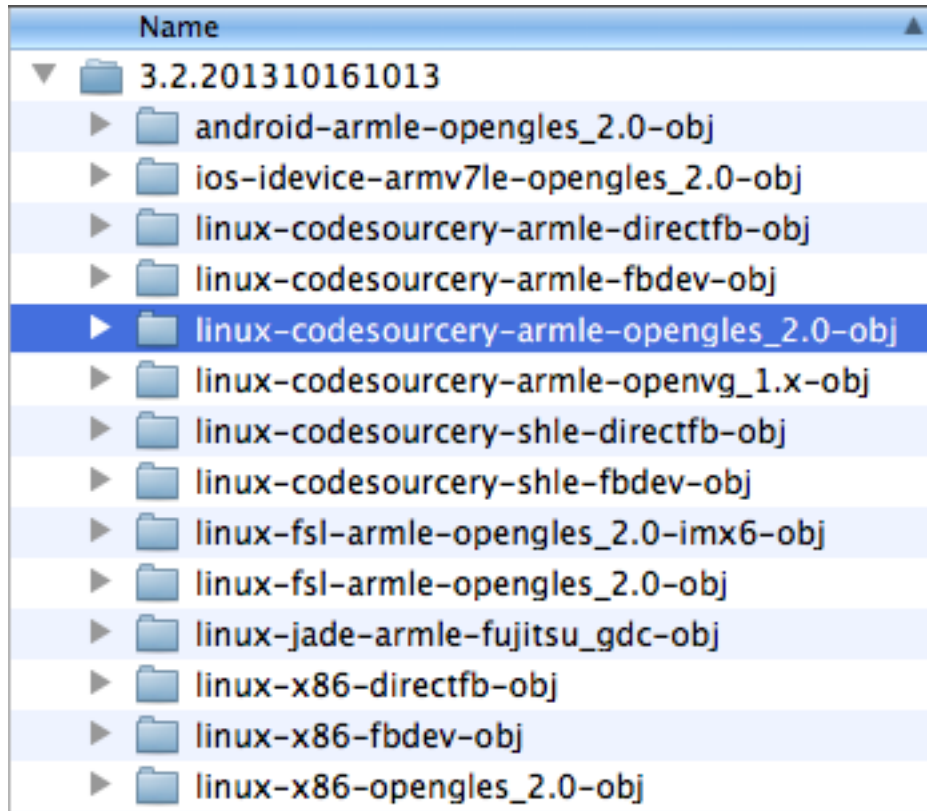


Step 3: Selecting The Storyboard Embedded Engine

The Storyboard Embedded Engine is the optimized runtime component that resides on the target platform that interprets the data bundle to display the Storyboard application. The Storyboard Embedded Engine is categorized by operating system, system architecture, and rendering technology.

All the supported Storyboard Embedded Engines are shipped with Storyboard Suite and are located under the Storyboard_Engine directory.

To run the Storyboard application on the TI AM335 a Linux, Armle, OpenGL ES runtime is required. Copy the linux-codesourcery-armle-opengles_2.0-obj Storyboard Embedded Engine and place it on the USB drive along with the Storyboard application that was just exported.



Step 4: Configuring The Target Platform

The USB drive with the Storyboard application along with the correct Storyboard Embedded Engine can now be ejected from the laptop/desktop and connected to the TI AM335.

The following commands are to be executed within the serial terminal connected to the TI AM335.

Login to the TI AM 335 and mount the USB drive:

```
login: root
```

```
mount /dev/sda1 /mnt/usb
```

Now, the Storyboard SB_PLUGINS specific environmental variable and an addition to the LD_LIBRARY_PATH environment variable need to be made:

```
export SB_PLUGINS=/mnt/usb/linux-codesourcery-armle-opengles_2.0-obj/  
plugins
```

```
export LD_LIBRARY_PATH=/mnt/usb/linux-codesourcery-armle-opengles_2.0-  
obj/lib:$LD_LIBRARY_PATH
```

Step 5: Running The Storyboard Application

With the needed environmental variables now set, the next step is to run the Storyboard application by passing it to the Storyboard Embedded Engine.

```
/mnt/usb/linux-codesourcery-armle-opengles_2.0-obj/bin/sbengine /mnt/  
usb/storyboard_export/trend.gapp
```

Part IV. Release Notes

Table of Contents

23. Release Notes 4.0	270
Introduction	270
Storyboard Designer	270
Changes	270
Known Issues	270
Storyboard Engine	271
Changes	271
Known Issues	272
24. Release Notes 4.1	273
Introduction	273
Storyboard Designer	273
Changes	273
Known Issues	274
Storyboard Engine	274
Changes	274
Known Issues	274
25. Release Notes 4.2	276
Introduction	276
Storyboard Designer	276
Changes	276
Known Issues	277
Storyboard Embedded Engine	277
Changes	277
Known Issues	278
26. Release Notes 4.2.1	279
Introduction	279
Storyboard Designer	279
Changes	279
Known Issues	279
Storyboard Embedded Engine	280
Changes	280
Known Issues	280

Chapter 23. Release Notes 4.0

Introduction

Welcome to the release of Storyboard 4.0

Storyboard Designer

The Storyboard Suite 4.0 release includes stability and performance enhancements. Storyboard 4.0 will automatically convert Storyboard 3 and earlier workspaces and project files. Once converted to Storyboard 4.0 format, Storyboard projects are no longer compatible with Storyboard 3 and earlier projects.

In addition to these enhancements, the following changes have been made to Storyboard Designer and Storyboard Embedded Engine.

Changes

Eclipse 4.x Update

Storyboard Designer is now using the Eclipse 4.x framework.

Lua Editor Updates

The Lua editor has been updated to include a number of productivity enhancements, including code auto completion, code folding, jump to variable and function definitions, and documentation tool tips.

Lua debugging has been streamlined to allow one-click application debugging.

Control Rotation

Control rotation now supports alpha.

Relaxed Naming Conventions

We removed the restriction in Storyboard Designer that requires the name creation of elements to be unique within the entire model space to only requiring name uniqueness at a peer level.

We've also provided the capability to name your render extensions.

Known Issues

Error When Updating or Installing Plugins

When Eclipse tries to update/install plugins, it doesn't request an escalation in privileges which causes the update to fail with a strange error. Always run Storyboard Designer as administrator when doing an update or installing plugins.

Require Mac OS X 10.8 or greater

Storyboard Designer requires Mac OS X 10.8 or greater due to the requirement by the Java 8 runtime from Oracle.

Resource Cleanup tool

Images that are not referenced directly, but are pulled in at runtime-based on a Lua script action, will not be identified as used by the Resource Cleanup wizard and may be removed accidentally.

Storyboard Engine

Storyboard Engine continues to move forward with the goal of providing our customers with a broad list of supported targets.

Changes

Asynchronous Lua scripts

The enablement for OS thread level locking has been added to the Storyboard Lua distribution.

Control Groups

Control Groups has been added to rid the situation of placing many render extensions in the same control. You now have the ability to group individual controls together.

Scrolling Layers

You now have the ability to scroll multiple controls on a layer with the same ease as table scrolling.

Layer Attributes

A layer's width and height can now be dynamically altered using `gre.set_layer_attrs`.

Circles and Arcs

Circles and arcs are now easier to create and manipulate in Storyboard.

QNX 6.6 Screen

Storyboard Embedded Engine has added support for QNX 6.6 screen.

Freetype Font Update

Our freetype font support has been updated to the latest version.

Green Hills Integrity

Storyboard now supports Green Hills Integrity Real-Time Operating System.

9-patch Image Support

9-patch support has been added to make scaling images on embedded applications easier.

Known Issues

Windows 8

Windows 8 will scale a Storyboard application to its resolution regardless of what the app was created to be displayed at. Setting the "Disable display scaling on high DPI settings" in the compatibility section for the Storyboard runtime will resolve the issue.

iOS 8.x

In order to export for iOS 8.x devices, users will need to have installed Xcode 6.1.1 and its command line tools.

Chapter 24. Release Notes 4.1

Introduction

Welcome to the release of Storyboard 4.1

Storyboard Designer

The Storyboard Suite 4.1 release includes stability and performance enhancements. Storyboard 4.1 will automatically convert Storyboard 3 and earlier workspaces and project files. Once converted to Storyboard 4.1 format, Storyboard projects are no longer compatible with Storyboard 3 and earlier projects.

In addition to these enhancements, the following changes have been made to Storyboard Designer and Storyboard Embedded Engine.

Changes

Animation Timeline Editing Enhancements

There have been a significant number of positive changes to the animation timeline to improve its usability and aesthetics which include ...

- The ability to copy and paste entire rows of animation steps
- Use alignment controls on animation timelines
- Editing within the properties view
- Updated color coding to easily distinguish between animation elements

Application View Enhancements

Minor enhancements to the application model view have been made to improve productivity such as ...

- Double click to edit Lua Scripts
- Change Lua Script action label to include the Lua function name
- Double click the Action for editing to do something contextual

Properties View Enhancements

The properties view has been cleaned up to be less cluttered. Tabs have replaced the expanding shelves making it easier to see what render extension is active and currently being modified.

Highlighting Groups

Groups are now highlighted upon selection in the Application Model or if they are ALT-selected in the editor. The selection displays the outline of the group contents and allows for resizing. Groups are also shown in the wireframe and outline mode.

Update PSD Workflow (aka PSD merge, update and sync)

A new page has been added to the PSD re-importer that allows the user to review and accept location changes of re-imported images.

Color Picker Overhaul

The platform specific color selectors have been replaced with a new generic SWT based color picker.

Known Issues

Error When Updating or Installing Plugins

When Eclipse tries to update / install plugins it doesn't request an escalation in privileges which causes the update to fail with a strange error. Always run Storyboard Designer as administrator when doing an update or installing plugins.

Require Mac OS X 10.8 or greater

Storyboard Designer requires Mac OS X 10.8 or greater due to the requirement by the Java 8 runtime from Oracle.

Resource Cleanup tool

Images that are not referenced directly, but are pulled in at runtime-based on a Lua script action, will not be identified as used by the Resource Cleanup wizard and may be removed accidentally.

Storyboard Engine

Storyboard Engine continues to move forward with the goal of providing our customers with a broad list of supported targets.

Changes

Windows Multi-Touch and Gesture Support

Multi-touch support has been added via the winevent plugin that extends the normal windows event processing loop to support multi-touch event processing.

Gesture Plugin

A configurable threshold to the gesture plugin has been added. This will allow users to configure the sensitivity of the motions to the resolution of their displays.

Known Issues

Windows 8

Windows 8 will scale a Storyboard application to its resolution regardless of what the app was created to be displayed at. Setting the "Disable display scaling on high DPI settings" in the compatibility section for the Storyboard runtime will resolve the issue.

iOS 8.x

In order to export for iOS 8.x devices, users will need to have installed Xcode 6.1.1 and its command line tools.

Chapter 25. Release Notes 4.2

Introduction

Welcome to the release of Storyboard Suite 4.2

This new version of Storyboard Suite introduces several major enhancements and a large number of improvements and efficiencies:

Please note

Once a Storyboard Project has been converted to the Storyboard 4.x format, Storyboard projects are no longer compatible with Storyboard 3 and earlier projects.

Storyboard Designer

Changes

New Installer

Storyboard Suite has a new installer. Fancy isn't it? We think so :)

3D Multi-Mesh Model Support via Autodesk FBX Content Importing

The Autodesk FBX file format is now supported as a 3D content import format in addition to the previous support Storyboard had for Wavefront OBJ files. Scene objects can be manipulated direct by importing the animations directly from the FBX file or by manually selecting individual node attributes.

3D Diagnostic Output

Added a new argument to the 3D render extension (-omodel3d,diag=n) that can be used to enable various levels of diagnostic output.

Storyboard Designer Animation Preview

Animations can be previewed and single stepped within the Designer environment without having to launch the complete application simulation. Animations can be designed and viewed in a variety of different execution contexts to ensure proper applicability in all environments.

Animation Snapshots

Users have the ability to take snapshots (set keyframes) during animation recordings.

Designer Group Template Support

Groups can now be incorporated into Designer templates opening up a broad new range of functionality that can be pre-packaged and used out of the box. Animations, images, 3D model content, timers and Lua scripts can all be incorporated into template blocks and moved from project to project.

9-Patch Image Editing and Refactoring

Quickly analyze and convert existing large or scaled image content to nine-patch format to achieve immediate memory and runtime performance improvements. Design and edit nine-patch images directly in the Storyboard Design environment.

Integrated Search and Property Editing

Fast project refactoring through the use of customized search criteria and an expanded set of multi-selection based property changes.

Android MultiTouch Support

The `-ogesture,mode=multi` option needs to be passed to the Storyboard Runtime Options when exporting an Android apk from Storyboard Designer.

Re-order Screen Layout

Ability to re-order the layout/presentation of screens. Place associated screens close to one another so that customers can move back and forth between them easily, compare them, etc.

Copy/Paste Layers

Copying and pasting layers is much easier by using the new Duplicate Layer action.

PSD Re-import Update

Added a new page to the re-import wizard that allows the user to review and import newly added controls that were not included in the original import.

Known Issues

Require Mac OS X 10.8 or greater

Storyboard Designer requires Mac OS X 10.8 or greater due to the requirement by the Java 8 runtime from Oracle.

Resource Cleanup tool

Images that are not referenced directly, but are pulled in at runtime based on a Lua script action, will not be identified as used by the Resource Cleanup wizard and may be removed accidentally.

3D Model's Axis

3D model's axis no longer shows up when in the direct editing mode.

Storyboard Embedded Engine

Changes

Simplified Data Access

Write less glue logic and express your intent more succinctly with our improved Lua API's for UI variable access and Storyboard IO array data transfer support.

Multitouch

The runtime options `-omtdev,max_x` and `-omtdev,max_y` have been deprecated. The max touch x and y values are now automatically detected.

Known Issues

Texture Memory

"ERROR (1):ERROR: 505 : create_image_texture@2791" relates to running out of texture memory. Since the texture is purged when the image resource gets released you can use the resource manager "image" pool size option, i.e `sbengine -oresource_mgr,image=4 your_app.gapp`, to get around this issue. This option enables you to manually set the image pool size. (By default sbengine will use all available memory) When the image pool size is reached, older images are released to create room for newer images.

3D Layer Rotation

3D layer rotation does not respect alpha. When a 3D layer, that has transparency, is rotated it gets rendered with a white fill.

9-Patch

9-Patch images currently can't be rotated.

Android

Storyboard Android crashes when accessing array values via the dot notation from the luajava bridge.

Cloned Control

Data does not get updated automatically in a cloned text control on a scrolling layer that is created off screen using the win 32 runtime.

3D Models

3D models do not show up on some versions of Linux.

Linux Wayland Runtimes

The Linux Wayland runtimes do not support keyboards.

Flickering Graphics

Using the Yocto Jethro linux kernel (3.14) with the boundary devices branch for the nitrogen6x you might encounter flickering graphics. If so, executing the following line resolves the issue ... `echo 10 >/sys/devices/soc0/backlight_lvs0.17/backlight/backlight_lvs0.17/brightness`

Chapter 26. Release Notes 4.2.1

Introduction

Welcome to the release of Storyboard Suite 4.2.1

This release is a maintenance update to the 4.2 release focusing primarily on the runtime engine.

Storyboard Designer

Changes

Resize Dialog

The resize dialog has been re organized from a visual perspective to make it less cluttered.

Designer Scroll bars

Scroll bars have returned to the main editor.

Lua Debugger

Resolved an issue where the Lua Debugger would present errors when using the csv.lua script.

Designer Default Memory

Designer's default memory has been increased from 512M to 1G to help performance when working with large images and 3D models.

Known Issues

Require Mac OS X 10.8 or greater

Storyboard Designer requires Mac OS X 10.8 or greater due to the requirement by the Java 8 runtime from Oracle.

Resource Cleanup tool

Images that are not referenced directly, but are pulled in at runtime based on a Lua script action, will not be identified as used by the Resource Cleanup wizard and may be removed accidentally.

3D Model's Axis

3D model's axis no longer shows up when in the direct editing mode.

Storyboard Embedded Engine

Changes

Performance

A number of enhancements have been made to Storyboard Embedded Engine to increase overall performance. Circle controls were one of the areas that benefited from these changes as we saw an order of magnitude in performance on some target platforms.

Memory Management

Memory management was another focus in this release. Changes were made that resolved some potential memory leaks in areas such as Lua animations and cloned polygon controls.

General Bug Fixes

Many bug fixes are included with this release including Text wrapping, PNG decoding, Table resizing and accessing array values in Android just to name a few.

Known Issues

Texture Memory

"ERROR (1):ERROR: 505 : create_image_texture@2791" relates to running out of texture memory. Since the texture is purged when the image resource gets released you can use the resource manager "image" pool size option, i.e sbengine -oresource_mgr,image=4 your_app.gapp, to get around this issue. This option enables you to manually set the image pool size. (By default sbengine will use all available memory) When the image pool size is reached, older images are released to create room for newer images.

3D Layer Rotation

3D layer rotation does not respect alpha. When a 3D layer, that has transparency, is rotated it gets rendered with a white fill.

9-Patch

9-Patch images currently can't be rotated.

Linux Wayland Runtimes

The Linux Wayland runtimes do not support keyboards.

Flickering Graphics

Using the Yocto Jethro linux kernel (3.14) with the boundary devices branch for the nitrogen6x you might encounter flickering graphics. If so, executing the following line resolves the issue ... echo 10 >/sys/devices/soc0/backlight_lvs0.17/backlight/backlight_lvs0.17/brightness

Part V. Licensing

Table of Contents

27. END-USER LICENSE AGREEMENT	283
28. Crank Software Third Party License Guide	293
Introduction	293
Storyboard Designer	293
Lightweight Java Game Library	293
Storyboard Engine	294
Lua	294
SOIL	294
Option Parsing	295
XML Parsing	295
Imagination OpenGL libraries	296
FreeType library	296
Scanline edge-flag algorithm for antialiasing	298
General IFF format	299
GNU LESSER GENERAL PUBLIC LICENSE	299
Storyboard Engine Platform Specific Dependencies	301
All Simple Direct Media Layer (SDL) renderers	302
All Simple Direct Media Layer (SDL), OpenGL ES 2.0, and Fujitsu Jade renderers.....	302
Fonts	302
Bitstream Vera	302
Bitstream Deja Vu	304
Liberation	305
Roboto	307
Lato	307

Chapter 27. END-USER LICENSE AGREEMENT

The software and related documentation that you are about to access ("Software", as further defined below) is offered to You (either an individual or a legal entity) by Crank Software Inc. ("Crank") of 4017 Carling Ave, Suite 302, Ottawa, Ontario, Canada K2K 2A3 (voice: +1.613.595.1999) for use only in accordance with the terms of the Storyboard End User License Agreement. Some Software components have supplementary or alternative end user license terms, as noted below.

BY ANSWERING "I ACCEPT" DURING THE DOWNLOAD AND/OR INSTALLATION OF THE SOFTWARE, OR OTHERWISE ATTEMPTING TO DOWNLOAD, COPY, INSTALL OR USE ANY PART OF THE SOFTWARE, YOU ARE REPRESENTING THAT YOU HAVE READ, UNDERSTOOD AND AGREE TO BE BOUND BY THESE TERMS AND AGREE TO PAY ALL ASSOCIATED FEES. NOTHING ELSE GRANTS YOU PERMISSION TO COPY, USE OR MODIFY THE SOFTWARE. THESE ACTIONS ARE PROHIBITED BY LAW IF YOU DO NOT ACCEPT THESE TERMS, UNLESS YOU HAVE AN ALTERNATIVE SIGNED AGREEMENT WITH CRANK. DO NOT PROCEED UNLESS YOU ARE ABLE AND WILLING TO ENTER INTO THESE AGREEMENTS AND COMPLY WITH THESE TERMS. IF YOU HAVE ANY QUESTIONS CONTACT CRANK BEFORE YOU ATTEMPT TO COPY, INSTALL OR USE ANY PART OF THE SOFTWARE.

THE SOFTWARE MAY INCLUDE PRODUCT ACTIVATION AND OTHER TECHNOLOGY DESIGNED TO PREVENT UNAUTHORIZED COPYING. THE ACTIVATION TECHNOLOGY MAY PREVENT YOUR USE OF THE SOFTWARE IF YOU DO NOT FOLLOW THE ACTIVATION PROCESS DESCRIBED IN THE SOFTWARE AND DOCUMENTATION.

If you do not agree to these terms and conditions, please click "I Decline" and promptly return or, if received electronically, certify destruction of the Software and all accompanying items within five (5) days after receipt of Software, and receive a full refund of any license fee paid.

Storyboard End User License Agreement

This Storyboard End User License Agreement (comprising Part A – Background , Part B - Standard Terms and Conditions, and any documents incorporated by reference, collectively "this Agreement") is a legal agreement between You and Crank, and is made effective as of the date of Your acceptance of this Agreement, as defined above. The parties agree as follows.

Part A-Background

A1. Crank has developed and licenses Storyboard® Suite ("Storyboard Suite"), a software development toolset for designing and executing graphical user interfaces for embedded systems. Storyboard Suite consists of a number of individual software products and related collateral, including Storyboard Designer and Storyboard Engine, Storyboard Browser and optional Storyboard software development kits (sometimes referred to as a "SDK"). Storyboard Suite includes a variety of software development tools, including debuggers, libraries, headers, utilities, sample source code, a simulation engine, etc.

Embedded system developers, including any of its graphical designers, systems engineers or other personnel will typically build the graphical user interface for Target System(s) using Storyboard Designer. Storyboard Designer aids in the design, development, simulation and testing phases of this development. Developers may add to their suite by selecting the desired "Software Development Kit" (sometimes referred to as "SDK"), which provide a greater range of technology options and customization capabilities.

A2. This Agreement is intended to detail Your license rights to the Storyboard Designer and to any SDK products that You order, to support Your Target System development, testing, support, maintenance

END-USER LICENSE
AGREEMENT

and enhancement efforts. Each individual from Your organization using the Software in any way must be licensed to have an individual copy of the Software, regardless of whether the Software is used on individual workstations or in a networked environment. The license fees for the Storyboard Designer products are generally on a per developer basis. SDKs license fees are determined on a variety of bases, depending on a customer's requirements. The specific license rights will be detailed on the License Certificate provided to You for the Software. The Storyboard Engine is bundled with the Storyboard Suite for Target System design purposes; the Storyboard Engine must be separately licensed for redistribution within Target Systems.

A3. You may require one or more license keys or passwords from Crank to install and use the Software ("License Keys"). License Keys for evaluation or beta licenses may be time limited. All License Keys are to be treated as Confidential Information of Crank in accordance with the provisions of Section B5 [Confidentiality].

Part B – Standard Terms and Conditions

B1. Definitions. In this agreement,

- a. "Agreement" or "EULA" means this Storyboard End User License Agreement.
- b. "Authorized Workstations" means in relation to the Software or a part of the Software, those Workstations on which You have been authorized to install the Software or that part of the Software,
- c. "Commercially Released" means formally released, generally available, and fully supported by Crank. It explicitly does not include any software and/or collateral that Crank may make available from time-to-time that has been noted as any of "experimental", "engineering", "beta", "unsupported" or similar terminology.
- d. "Crank", "we", or "us" means Crank Software Inc.,
- e. "Contractor" means an independent contractor performing services for your development project that are substantially similar to those performed by You or Your employees;
- f. "Derivative Work" means any work made by You, or for You by a Contractor pursuant to this Agreement, that is a revision, modification, translation, expansion, extension, collection, condensation or abridgement of any Software provided by Crank in source code form;
- g. "Documentation" means any developer documentation, read-me files, release notes and License Guides (see B2(g)) that are provided by Crank in or for the Software;
- h. "License Certificate" means a Crank issued document which authenticates software licensed under this Agreement. It will: include a License Key; specify the number of Authorized Workstations if other than 1; and provide other details contemplated by this Agreement.
- i. "License Key" means license keys and/or passwords from Crank required to install and use the Software.
- j. "Software" means the object code and source code included in the Storyboard Suite for which You received a valid License Certificate and that You license pursuant to this Agreement. It includes associated Documentation and corresponding Software updates or supplemental releases that You are entitled to receive and use under one of Crank's support plans.
- k. "Target System" means any product into which any portion of the Storyboard Engine has been wholly or partially integrated, and which: (1) significantly enhances the function and value of the Storyboard Engine, and (2) has a substantially different principal purpose than that of the Storyboard Engine;

END-USER LICENSE
AGREEMENT

- l. "Workstation" means an individual developer's workstation, laptop and/or home computer used to perform Storyboard development, provided the Software is only used on one computer at a time;
- m. "You" or "Your" means to the entity for whose benefit You act, which may be yourself as an individual, a corporate entity or some other form of legal entity;
- n. Other capitalized terms defined in any part of this Agreement will have their indicated meaning throughout this Agreement.

B2. Software Development License, Restrictions and Requirements.

- a. **License Rights.** Subject to the terms of this Agreement (including without limitation those specific to third party software - see Section B2(g) [Third Party Software] below) - and payment of all applicable license fees, Crank hereby grants to You for each applicable license purchased from Crank (or from one of its authorized distributors) for Software to be used pursuant to this Agreement, a non-exclusive, personal, non-sublicensable and non-transferable license to:
 - 1. copy the Software as required to install it on and to follow normal back-up and archiving practices for Authorized Workstations;
 - 2. use, execute, display and perform the Software on the Authorized Workstations in accordance with associated Documentation, for the purpose of developing, testing and maintaining Target Systems;
 - 3. create Derivative Works of Software source code and, subject to the provisions of Section B5 [Confidentiality], copy, compile, link, use, execute, display and perform such Derivative Works on Authorized Workstations in accordance with associated Documentation, for the purpose of developing, testing and maintaining Target Systems; and
 - 4. copy, link, use, execute, display and perform the Storyboard Engine, and the object code of any Derivative Works created pursuant to (3) above, as required to install and use them: (i) on a reasonable number of Target Systems solely for internal Target System development and testing purposes; and (ii) on one Target System for demonstration, promotion, evaluation or training purposes, provided that such copy is not left with third parties.
- b. You may authorize Your Contractors to exercise any of the license rights in B2(a), provide that You remain responsible to Crank for the performance of any obligations, and compliance with any restrictions, required by this Agreement,
- c. **Time and other Limited Licenses.** If You received Software under an evaluation, beta or other time-limited license, Your rights in the Software may be further limited as contemplated in this EULA, on the License Certificate You receive, or as otherwise specified at the time of download and Your license rights in the Software will end when the term of Your license expires. Crank may, at its discretion, include with the Software capabilities to remind You of the time limitations and to prevent You from continuing to use the Software at the end of the term.
- d. **Educational License.** If You received Software under an educational license, then Your license rights in the Software
 - 1. are limited to educational, academic, research, instructional, teaching and training purposes ("Educational Purposes) only and expressly exclude rights for any commercial purposes;
 - 2. under B2(a)(4)(ii) are amended to ten Target Systems for Educational Purposes and expressly not for any commercial purposes;
 - 3. may be further limited (including by time as indicated in section B.2(c)) as contemplated in this EULA, on the License Certificate You receive, as otherwise specified at the time of download or as otherwise specified in writing by Crank.

END-USER LICENSE
AGREEMENT

- e. **Ownership and Use Restrictions.** Crank and its suppliers retain all right, title and interest in and to the Software, including all intellectual property therein. All copies will be considered Software for the purpose of this Agreement and shall remain the property of Crank and its suppliers. Without restricting the generality of the foregoing, unless expressly permitted by this Agreement, by applicable law, or by Crank in writing, You agree not to:
1. alter, remove, or cover any trademark, logo, proprietary or licensing notices, labels or marks in or on any part of the Software, including in any "about" box, "flash" / "splash" screen or Documentation. You agree to use reasonable efforts to ensure that all copies of the Software bear any notices, labels or marks contained in or on the original;
 2. copy, reproduce, publish, rent, lease, loan, or distribute the Software except as expressly provided in this EULA;
 3. use unauthorized license keys;
 4. decompile, disassemble, decrypt, extract, unbundle, translate or otherwise attempt or assist others to reverse engineer any part of the Software, including circumventing any License Key activation or evaluation period expiry mechanisms, except as necessary, when permitted by an applicable law, to correct defects or achieve interoperability with complementary programs, for Your purposes only, but only if Crank has refused to provide the necessary information or assistance;
 5. directly or indirectly export, import or transmit the Software to any country in contravention of the laws of that country or the laws of Canada or the United States;
 6. use the Software in a High Risk Application (see also Section B9[No High Risk Applications]).
- f. **Other Agreements / Products Required.** For certainty, this EULA does not provide You with any rights to distribute the Software, or the files and data Storyboard generates, on any Target Systems. Any such right / entitlement would be the subject of a separate agreement with Crank. Any right to obtain support for the Software is subject to You purchasing the applicable support products and the terms of Parts C and/or D of this Agreement.
- g. **Ownership of Derivative Works.** Subject to any underlying rights in the Software, and subject to any Feedback provided under Section B2(f) [Feedback], You retain all right, title and interest in and to any Derivative Works and application software that You develop pursuant to this Agreement.
- h. **Feedback.** At Your option, Crank would like to get suggestions, comments or other feedback about its products (i.e., regarding their utility, reliability, performance and Your user experience, as well as any bug-fixes, features, functionality or enhancements You would like to see in future versions; collectively "Feedback"). You agree that all Feedback is and shall be given entirely voluntarily and, even if designated as confidential, will not create any confidentiality obligations for Crank. You agree not to provide any Feedback that is subject to any third party intellectual property rights. If You desire to license any of Your intellectual property to Crank You will not provide the intellectual property information to Crank as Feedback, but rather, we will discuss the necessity of entering into a separate agreement. In the absence of such an agreement, and in order to incorporate Feedback that You provide, Crank requires, and You hereby agree, to assign and waive all right, title and interest (if any) in and to any Storyboard-specific Improvements (as defined below), including any associated intellectual property and moral rights, to and on behalf of Crank. In this paragraph "Storyboard-specific Improvements" means any work-arounds, bug-fixes, features, functionality, enhancements or other suggested improvements to the Software that You provide to Crank.
- i. **Third Party Software.** Parts of the Software may contain third party code. When permitted such Software is sublicensed to You under the standard terms of this Agreement or otherwise may be licensed to You under amended or alternative terms. Those terms, and any Software authorship attribution and like

END-USER LICENSE AGREEMENT

notices that Crank is obliged to provide to You, are referenced in the corresponding "License Guide", which is included with the Software Documentation and is available through www.cranksoftware.com.

B3. Limited Rights

- a. **Evaluation Rights.** Notwithstanding Section B2(a)[License Rights], Software provided under an evaluation or time limited license ("Evaluation License") may only be used for determining the suitability of the Software for your intended Target System application. An Evaluation License does not allow you to use the software for commercial development purposes.
- b. **Beta Code.** As specified to You by Crank, the Software (or parts of it) may be code intended for experimental testing and evaluation ("Beta Code"). For any code specified as Beta Code by Crank, Crank grants to You a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Crank. This grant and Your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Crank may choose not to release commercially in any form. You agree to evaluate and test the Beta Code under normal conditions. You are encouraged to contact Crank periodically during Your use of the Beta Code to discuss any malfunctions or suggested improvements and upon completion of Your evaluation and testing, to send to Crank a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements and this will be treated as Feedback pursuant to Section B2(f) [Feedback].

B4. Activation, Audit and Reporting.

- a. The Software require activation in order to install and certain machine-specific information is sent ("Activation Information") to Crank at the time of activation and/or periodically thereafter. This Activation Information may include but is not limited to software identification number, MAC address, UUID, IP address, identification numbers set by manufacturers of hardware and/or identification numbers related to the host operating system. During some instances of activation, you may be asked for certain information such as your name, email address and company information. Other than the information which you enter, Crank does not collect any personally identifiable information during activation. Crank may collect Activation Information at any time and may use Activation Information for the purposes of verifying compliance with the terms of this Agreement.
- b. Crank may audit Your use and deployment of the Software for compliance with the terms of this Agreement and may reference Activation Information in the course of such audit. You will reimburse Crank for its reasonable out of pocket costs associated with this audit if it is determined Your use of the Software does not conform with the terms of this EULA. Crank shall treat as confidential information all of Your information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement or addendum to this Agreement.
- c. Crank may require that You provide Crank with a written report to verify Your compliance with the terms of this license; the report will be signed by an individual authorized to bind You confirming the accuracy of the report. Such a report could include, but would not be limited to the serial number of each Software product You have licensed, the MAC address or other unique identifier of each computer on which each Software copy is installed and confirmation that each developer has the Software installed only on his/her Workstation.

B5. Confidentiality.

- a. **What is Not Confidential.** The Software user interface is not confidential information and Crank encourages You to tell others about / how others Crank products. We would rather You provide negative feedback to us first so that we might have a chance to respond or fix the issue; however, we support Your right to speak about our Software even if we don't agree with what You are saying. Most Documentation is freely available on our web site and that which is available is clearly is not confidential.

END-USER LICENSE
AGREEMENT

- b. What is Confidential. "Confidential Information" means any information provided by Crank in, with or associated with the Software (1) in Software source code, (2) which is a License Key , or (3) in a document clearly marked "Confidential" (or equivalent). Confidential Information does not include any information which is publicly available, previously known to You or independently developed by You without reference to the Confidential Information. You may use Confidential Information only to exercise Your rights under this Agreement and it may not be disclosed except to those developers who have Authorized Workstations. You will protect the Confidential Information of by using the same degree of care, but no less than reasonable care, to prevent the unauthorized dissemination or publication and unauthorized use of the Confidential Information as You use to protect Your own confidential information of like nature. Your duty to protect Confidential Information disclosed to it will survive termination of this Agreement indefinitely.

B6. Intellectual Property Indemnity.

- a. Indemnity. Crank will defend You against any Infringement claims, and indemnify and hold You harmless from any Infringement damages finally awarded, in any third party action against You based on the reproduction or use of the Commercially Released Software in accordance with the terms of this Agreement, provided that You give Crank prompt notice of, as well as all authority, information, and assistance (at Crank's expense) necessary or desirable to defend, such claims. In this Section B4.2 "Infringement" means: (i) infringement of copyright by the Software; or (ii) misappropriation of trade secrets by Crank in relation to the Software; or (iii) infringement by the Software of any patent,

Crank has no liability to You if the Infringement claim is based upon: (a) the combination of Software with any product not furnished by Crank; (b) the modification of Software other than by Crank; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that You make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Crank's licensors or under an open source license who / which does not provide such indemnification to Crank's customers; (h) infringement of a patent that is required to implement any technical standard, be it formal or informal; (i) infringement of a patent that requires a license not provided by Crank as detailed in B10 (i) infringement by You that is willful. In the case of (i) You shall reimburse Crank for its legal fees and other costs related to the action upon a final judgment. In this section "technical standards" includes without limitation, standards/recommendations of ITU, IEEE, ETSI, ISO, MPEG, CSS, DVD, JPEG, DivX, Dolby, AVC/H.264, ATM Forum, Frame Relay Forum, SMPTE, ATSE, GSM, IETF, OpenGL, Posix, OpenVG, DirectFB, etc.

- b. Remedy. With respect to any finding of Infringement, or any reasonable belief of Crank that Infringement may occur, Crank will, at its sole expense and option: (1) procure for You the right to continue using the infringing Software; (2) replace the infringing Software with non-infringing software of comparable function; (3) modify the infringing Software to be non-infringing; or (4) if none of the foregoing alternatives is reasonably available to Crank, terminate Your right to the Software, but only to the extent necessary to avoid the Infringement. You will have the right to terminate all of Your rights if You determine such partial termination renders Your remaining rights ineffective. Upon such full or partial termination, Crank will refund to You, pro-rata to the extent of such termination, the license fees paid by You that are associated with the terminated rights.

- c. Entire Liability. This Section B6 states the entire liability of Crank and its licensors and Your sole and exclusive remedy with respect to any alleged infringement of intellectual property rights by the software.

B7. Limited Warranty.

- a. Crank warrants that during the warranty period the Commercially Released Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable Documentation. Crank does not warrant that Software will meet Your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 60 days starting on the day Crank

END-USER LICENSE
AGREEMENT

issues to you an invoice for the Software. You must notify Crank in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation.

- b. CRANK'S ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT CRANK'S OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO CRANK OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. CRANK MAKES NO WARRANTIES OR REPRESENTATIONS WITH RESPECT TO: (I) SERVICES; (II) SOFTWARE WHICH IS LICENSED TO YOU FOR A LIMITED TERM, FOR EVALUATION PURPOSES OR LICENSED AT NO COST; OR (III) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."
- c. THE WARRANTIES AND REPRESENTATIONS SET FORTH IN THIS SECTION B7 ARE EXCLUSIVE. EXCEPT AS EXPRESSLY PROVIDED HEREIN, THE SOFTWARE PRODUCTS AND ANY SERVICES PROVIDED UNDER THIS AGREEMENT ARE PROVIDED "AS IS" WITHOUT ANY WARRANTIES OF ANY KIND, INCLUDING IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. NOTHING STATED IN THIS AGREEMENT WILL IMPLY THAT THE OPERATION OF ANY SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE OR THAT ERRORS WILL BE CORRECTED. OTHER WRITTEN OR ORAL STATEMENTS BY CRANK, ITS REPRESENTATIVES OR OTHERS DO NOT CONSTITUTE WARRANTIES OF CRANK.

B8. LIMITATION OF LIABILITY.

- a. IN NO EVENT WILL CRANK OR ITS AFFILIATES, OR THEIR OFFICERS, EMPLOYEES, AGENTS, SUPPLIERS, DISTRIBUTORS, OR LICENSORS, (COLLECTIVELY, CRANK AND ITS REPRESENTATIVES) BE LIABLE TO YOU, YOUR CONSULTANTS, OR ANY OTHER THIRD PARTY FOR ANY INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST REVENUE, LOST OR DAMAGED DATA, OR OTHER COMMERCIAL OR ECONOMIC LOSS, ARISING OUT OF OR RELATING TO ANY BREACH OF THIS AGREEMENT, ANY USE OR INABILITY TO USE SOFTWARE PRODUCTS, OR ANY SERVICES PROVIDED OR INABILITY TO OBTAIN SERVICES, EVEN IF CRANK HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE OR CLAIM.
- b. IN NO EVENT WILL THE AGGREGATE LIABILITY OF CRANK AND ITS REPRESENTATIVES FOR ANY DAMAGES ARISING OUT OF OR RELATING TO THIS AGREEMENT, WHETHER IN CONTRACT, TORT, OR OTHERWISE, EXCEED THE TOTAL FEES YOU HAVE PAID TO CRANK FOR USE OF THE SOFTWARE UNDER THIS AGREEMENT (WHICH TOTAL FEES MAY BE ZERO). THE PROVISIONS OF SECTIONS B7(b) and (c) AND THIS SECTION B8 SHALL SURVIVE AND APPLY NOTWITHSTANDING THE FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

B9. No High Risk Applications. Unless Crank has provided You with express written consent, the Software may not be used in any application in which the failure of the Software could lead to death, personal injury, or severe physical or property damage (collectively, "High-Risk Applications"), including but not limited to the operation of nuclear facilities, mass transit systems, aircraft navigation or aircraft communication systems, air traffic control, weapon systems, and direct life support machines. Crank expressly disclaims any express or implied warranty or condition of fitness for High-Risk Applications.

B10. Third Party Licenses Required. Certain Software products noted in the License Guide provide software for implementing products or systems that may require additional patent license rights. Further, Crank only licenses to You the intellectual property interests in such Software that it owns, patent license rights (if any) that it expressly identifies for such Software in the description of Software and developer files, and any third party copyright interests in these software products. It is Your responsibility to

END-USER LICENSE AGREEMENT

determine if You require, and to obtain as necessary, any additional rights, from patent owners / consortia, before making, using or selling any infringing product or system that contains or uses such Software.

B11. Term and Termination

- a. Term of this Agreement. This term of this Agreement will commence on the date of Your acceptance of this Agreement, as indicated above, and will continue indefinitely thereafter until it expires or is terminated in whole or in part under Sections B11(b)[Time Limited Licenses] or B11(c)[Termination].
- b. Time Limited Licenses. The term of any Software evaluation, beta, trial or other time limited Software license will expire on the earlier of: (1) the end of the applicable time-limited trial period, or (2) thirty (30) days after notice from Crank.
- c. Termination. You may terminate this Agreement at any time. It will be deemed to terminate immediately if You fail to comply with any material term herein or if You fail to pay within thirty (30) days of receipt of invoice (or such longer period as may be expressly permitted by Crank in writing) any license fees invoiced by Crank for Software licensed pursuant to this Agreement.
- d. Implication of Termination. The provisions of this Agreement that are expressed or by their sense and context are intended to survive the termination of this Agreement will survive, including Sections B2(c) [Ownership], B4 [Activation, Audit and Reporting], , B5 [Confidentiality], B6 [IP Indemnity], B7 [Limited Warranty], B8 [Limitation of Liability], B9 [No High Risk Applications], B10 [Limited Patent License] this Section B10 and Section B12 [General]. When this Agreement terminates or expires for any Software product(s) Your associated license rights end and You agree to immediately destroy all whole or partial copies of that Software that are in Your possession or control. Termination is without prejudice to any right or remedy that may have accrued, or be accruing to either party prior to termination.

B12. General

- a. Entire Agreement. This Agreement, comprising Parts A, B and C, along with the License Guide and any other terms expressly referenced by this Agreement (including third party terms referenced in the License Guide), constitutes the entire agreement between the parties pertaining to its subject matter and supersedes any prior or contemporaneous agreement, representation, statement, negotiation or undertaking dealing with the same subject matter. No amendment, modification or waiver of any part of this Agreement will be binding unless in a written document that expressly refers to this Agreement and that is signed by both parties. Except as otherwise expressly contemplated in this Agreement, the terms and conditions of this Agreement will prevail over any inconsistent or additional terms or conditions of either party's purchase orders or invoices.
- b. Assignment. Except as specifically allowed in this Section B12(b) or with Crank's written consent, you may not assign this Agreement or your License rights to third parties. Upon written notice to Crank and subject to the export restrictions in Section B2(h)[Use Restrictions], you assign this Agreement in conjunction with a change of ownership, merger, acquisition, sale or transfer of all or substantially all of your business. Any such assignee must provide Crank with prior written acknowledgement of their acceptance of the terms of this Agreement and you must transfer Your License Keys to the assignee and destroy all whole or partial copies of the Software and License Keys that are in Your possession or control. Any other attempted assignment or delegation in violation of the foregoing will be void and of no effect. This License will inure to the benefit of and be binding upon the parties and their respective successors and permitted assigns.
- c. Payment Terms. You will pay amounts invoiced, in the currency specified on the applicable invoice, within 30 days from the date of such invoice, unless otherwise agreed upon in advance by Crank in writing. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower.

END-USER LICENSE
AGREEMENT

- d. **Governing Law.** This Agreement will be governed by and construed in accordance with the laws in force in the Province of Ontario, Canada without regard to the conflicts of laws provisions thereof. The parties hereby irrevocably waive: (1) the provisions of the United Nations Convention on Contracts for the International Sale of Goods, and (2) any right to a trial by jury regarding the resolution of any dispute between the parties hereto arising out of or in connection with this Agreement.
- e. **Arbitration.** The parties will attempt to settle any disputes in connection with this Agreement in good faith. If the parties are unable to settle a dispute, it will be resolved by arbitration and finally settled by a sole arbitrator under the provisions of the Arbitration Act (Ontario) and the National Arbitration Rules of the ADR Institute of Canada, Inc. The arbitration shall take place in Ottawa, Canada and arbitration proceedings shall be held in the English Language. The arbitrator shall have all powers conferred on him or her by the Arbitration Act (Ontario), including the power to set or dispense with any process, to award costs and to award injunctive relief. The decision of the arbitrator will be final and binding on the parties. The prevailing party will be entitled to recover its costs and expenses from the arbitration, including but not limited to reasonable attorney's fees. All information relating to any dispute in connection with this License will be considered Confidential Information for the purpose of Section B5 [Confidentiality].

Part C: STANDARD SUPPORT TERMS

C0. Background. This Part C: Standard Support Terms ("Part C"), together with the other terms and conditions of this Storyboard EULA, provides the terms and conditions upon which Crank will provide you with the maintenance and support services described below ("Standard Support") for the Software. All defined terms in other parts of the Agreement will have the same meanings in this Part C.

C1. Subscription. Your Standard Support subscription applies to the Software, subject to your payment when due of all applicable Standard Support subscription fees specified for the first subscription year on your Software Invoice and subsequently on corresponding subscription renewal invoices. Standard Support subscriptions: (i) are specific to Authorized Work Stations and Software products; and (ii) may not be renewed once expired;

C2. Services. During your Standard Support subscription Crank will provide you with the following services for the current version of the Commercially Released Software by delivering by phone, email or the web assistance with: (i) installation and configuration issues; (ii) understanding the functionality and behaviour of specific parts; (iii) isolating problems you encounter by verifying whether or not they are errors; (iv) providing you with patches or work-arounds for known errors; and (v) submitting problem reports for confirmed errors that do not have current solutions. Crank has the right to publish information (including but not limited to work-arounds and fixes) relating to any issues you report for the benefit of the Storyboard development community; we will not publish any details that would identify you or your customers.

C3. Updates. During your Standard Support subscription Crank will provide you with access to Updates for use under the terms and conditions of this EULA if the Update is made available without a new end user license agreement, or under any new end user license agreement terms and conditions that are provided with the Update. An "Update" means a new Software release version designated by a change to the minor version number (i.e. n.1 to n.2) and such other versions as Crank at its discretion makes available under Your Standard Support subscription.

- a. **General.** For the purpose of this License, Updates: (a) may only be used if they are first made available before you purchased the applicable License Certificate or during your corresponding Standard Support subscription, (b) may not be shared with any other persons, unless they are entitled to use them under their own Storyboard license, (c) do not include any major Software releases (e.g., Storyboard 2 to Storyboard 3) unless they are designated as an Update by Crank, (d) do not include Crank products that you have not licensed commercially (i.e. the availability of Beta Software will not entitle you to free commercially released Software versions if additional license fees apply), and (e) do not include any new components, technologies or features that require Crank to pay additional third party fees.

END-USER LICENSE
AGREEMENT

b. Development. Updates may only be used on Authorized Workstations for which corresponding Standard Support fees have been paid.

C4. Standard Support Subscription Term. Each subscription is valid for Standard Support services for one Authorized Workstation for one year. Your subscription will end on the anniversary of the first day of the month following the date of your original Standard Support Invoice, unless you first renew your subscription by delivering a purchase order to Crank for the applicable Standard Support fee(s) for the next subscription year at least thirty (30) days prior to the expiry date. All subscription fees are due in advance and are non-refundable. Crank has the right to withhold Standard Support if you have not paid your subscription fees. Crank may cease to provide Standard Support for the Software upon twelve (12) months prior notice.

Part D - PRIORITY SUPPORT SERVICES TERMS

D0. Background. This Part D: Priority Support Terms ("Part D"), together with the other terms and conditions of this Storyboard EULA, provides the terms and conditions upon which Crank will provide you with the priority support services described below ("Priority Support") for the Software. All defined terms in other parts of the Agreement will have the same meanings in this Part D.

D1. Priority Support Services. Crank offers You the ability to define your own priority support plan, within reason. You may purchase blocks of "Priority Support Hours" in advance and have Crank deploy resource hours against these as required for your project. These Hours can be used in any manner which You and Crank have agreed and will be deployed in accordance with these terms.

D2. Priority Support Hours. Priority Support Hours are available in fixed blocks of time over a fixed time period. If the order for the Priority Support Hours does not specify a schedule for using these hours Crank will make them available on a pro-rated weekly basis and will use reasonable efforts to accommodate Your schedule and variance over the time period. Unless otherwise specified a block of Hours must be used within one (1) year of purchasing same. We will let you know if your Priority Support Hours are at risk of expiring.

D3. Delivery. Crank will deliver Priority Support services in a professional manner. This includes not knowingly infringing any third party intellectual property rights. However, this is the extent of the representations and warranties that we can offer to you under this Agreement. The services we provide will otherwise be delivered on an "AS IS" basis. There are no other representations, warranties or conditions, express or implied from us and the provisions of sections B8 [Limitation of Liability] apply to the provision of these services.

Chapter 28. Crank Software Third Party License Guide

Introduction

Crank Software incorporates certain third party software in our software suite. The license terms associated with this software require that we give copyright and license information, and this Third Party License Terms List (“TPLTL”) provides those details.

Storyboard Designer

These third party software components are used within the Storyboard Designer product.

Lightweight Java Game Library

The Lightweight Java Game Library <http://lwjgl.org/> is used to provide the 3D model rendering within Storyboard Designer.

```
/*
 * Copyright (c) 2002-2007 Lightweight Java Game Library Project
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are
 * met:
 *
 * * Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of 'Light Weight Java Game Library' nor the names of
 *   its contributors may be used to endorse or promote products derived
 *   from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
 * TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

*/

Storyboard Engine

These third party software components are used within the Storyboard Engine product.

Lua

The Lua scripting engine (<http://www.lua.org>) is used by the Storyboard Engine runtime when the lua plugin (libgre-plugin-lua.so) is loaded.

License for Lua 5.0 and later versions

Copyright © 1994-2008 Lua.org, PUC-Rio. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

SOIL

The SOIL library (<http://www.lonesock.net/soil.html>) is used to load images for the Storyboard Engine runtime. It is used on the following rendering platforms:

Win32 GDI
All OpenGL ES 2.0
All OpenVG
Linux fbdev
Fujitsu Jade

Jonathan Dummer
2007-07-26-10.36

Simple OpenGL Image Library

Public Domain
using Sean Barret's stb_image as a base

Thanks to:

- * Sean Barret - for the awesome stb_image
- * Dan Venkitachalam - for finding some non-compliant DDS files, and patching some explicit casts
- * everybody at gamedev.net

Option Parsing

The `getopt` and `getsubopt` argument parsing functions are used by the Storyboard Engine runtime and plugins.

Copyright (c) 1990, 1993

The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

XML Parsing

The `ezxml` library (ezxml.sourceforge.net [<http://ezxml.sourceforge.net>]) provides the XML parsing support for the Storyboard Engine runtime on all platforms.

Copyright 2004-2006 Aaron Voisine aaron@voisine.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including

without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Imagination OpenGL libraries

The `libEGL.dll` and `libEGLv2.dll` libraries from Imagination Technologies are used in the win32 OpenGL based Storyboard Engine runtimes.

THESE LIBRARIES ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. CRANK AND ITS LICENSORS HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THESE LIBRARIES, INCLUDING ALL WARRANTIES, IMPLIED OR EXPRESS, OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL CRANK OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES WHATSOEVER, (INCLUDING, WITHOUT LIMITATION, DAMAGES RESULTING FROM LOSS OF USE, DATA OR PROFITS), WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORT RELATED ACTION, ARISING OUT OF, OR IN CONNECTION WITH, OR IN CONTEMPLATION OF THE USE OR PERFORMANCE OF THE LIBRARIES, EVEN IF CRANK HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

FreeType library

The FreeType Project's www.freetype.org (<http://www.freetype.org>) library is used by the Storyboard Engine for text rendering on all Linux, QNX, Windows, Mac OS platforms.

Portions of this software are copyright © <2011> The FreeType Project (www.freetype.org). All rights reserved.

Legal Terms

=====

0. Definitions

Throughout this license, the terms 'package', 'FreeType Project', and 'FreeType archive' refer to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and

Werner Lemberg) as the 'FreeType Project', be they named as alpha, beta or final release.

'You' refers to the licensee, or person using the project, where 'using' is a generic term including compiling the project's source code as well as linking it to form a 'program' or 'executable'. This program is referred to as 'a program using the FreeType engine'.

This license applies to all files distributed in the original FreeType Project, including all source code, binaries and documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this.

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below.

1. No Warranty

THE FREETYPE PROJECT IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

2. Redistribution

This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions:

- o Redistribution of source code must retain this license file ('FTL.TXT') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered, original files must be preserved in all copies of source files.
- o Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory.

These conditions apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us.

3. Advertising

Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission.

We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: 'FreeType Project', 'FreeType Engine', 'FreeType library', or 'FreeType Distribution'.

As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

Scanline edge-flag algorithm for antialiasing

Scanline edge-flag algorithm for antialiasing
Copyright (c) 2005-2007 Kiia Kallio kkallio@uijah.fi

<http://mlab.uijah.fi/~kkallio/antialiasing/>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR

BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Research Paper Reference: <http://mlab.uiah.fi/~kkallio/antialiasing/> [<http://mlab.uiah.fi/~kkallio/antialiasing/>].

General IFF format

Copyright (c) 2012 Sander van der Burg

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Source GitHub Repository: <https://github.com/svanderburg/libiff> [<https://github.com/svanderburg/libiff>].

GNU LESSER GENERAL PUBLIC LICENSE

POSIX Threads Library for Windows, WinCE, and Windows Compact 7

Version 3, 29 June 2007

Copyright 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Storyboard Engine Platform Specific Dependencies

The individual render managers implementation may have a dependency on third party libraries that may not be included in the standard operating system platform libraries.

The API of these libraries are used by the Storyboard Engine runtime, but the binary dependency is via a shared object.

All Simple Direct Media Layer (SDL) renderers

SDL

Used for rendering <http://www.libsdl.org>

All Simple Direct Media Layer (SDL), OpenGL ES 2.0, and Fujitsu Jade renderers.

zlib

Compression library, required by FreeType library <http://www.zlib.net/>

```
/* zlib.h -- interface of the 'zlib' general purpose compression library
   version 1.2.8, April 28th, 2013
```

Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly	Mark Adler
jloup@gzip.org	madler@alumni.caltech.edu

*/

Fonts

Storyboard Designer includes a number of Open Source fonts that may be used in commercially deployable products. The licenses for the specific fonts are provided here.

Bitstream Vera

The Vera*.ttf font files are covered by these license terms

Fonts are (c) Bitstream (see below). DejaVu changes are in public domain. Glyphs imported from Arev fonts are (c) Tavmjong Bah (see below)

Bitstream Vera Fonts Copyright -----

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

Bitstream Deja Vu

The DejaVu*.ttf font files are covered by these license terms

Bitstream Vera Fonts Copyright

The fonts have a generous copyright, allowing derivative works (as long as "Bitstream" or "Vera" are not in the names), and full redistribution (so long as they are not *sold* by themselves). They can be bundled, redistributed and sold with any software.

The fonts are distributed under the following copyright:

Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF

MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Liberation

Digitized data copyright (c) 2010 Google Corporation
with Reserved Font Arimo, Tinos and Cousine.
Copyright (c) 2012 Red Hat, Inc.
with Reserved Font Name Liberation.

This Font Software is licensed under the SIL Open Font License,
Version 1.1.

This license is copied below, and is also available with a FAQ at:
<http://scripts.sil.org/OFL>

SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007

PREAMBLE The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others.

The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives.

DEFINITIONS

"Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such.
This may include source files, build scripts and documentation.

"Reserved Font Name" refers to any names specified as such after the copyright statement(s).

"Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s).

"Modified Version" refers to any derivative made by adding to, deleting, or substituting ? in part or in whole ? any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment.

"Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software.

PERMISSION & CONDITIONS

Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified copies of the Font Software, subject to the following conditions:

- 1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself.
- 2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user.
- 3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users.
- 4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission.
- 5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be distributed under any other license. The requirement for fonts to remain under this license does not apply to any document created using the Font Software.

TERMINATION

This license becomes null and void if any of the above conditions are not met.

DISCLAIMER

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING

FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Roboto

Copyright (C) 2008 The Android Open Source Project

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This directory contains the fonts for the platform. They are licensed under the Apache 2 license.

Lato

Copyright (c) 2010, #Åukasz Dziedzic (dziedzic@typoland.com),
with Reserved Font Name Lato.

This Font Software is licensed under the SIL Open Font License, Version 1.1.
This license is copied below, and is also available with a FAQ at:
<http://scripts.sil.org/OFL>

SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007

PREAMBLE

The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others.

The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives.

DEFINITIONS

"Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such. This may include source files, build scripts and documentation.

"Reserved Font Name" refers to any names specified as such after the copyright statement(s).

"Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s).

"Modified Version" refers to any derivative made by adding to, deleting, or substituting -- in part or in whole -- any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment.

"Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software.

PERMISSION & CONDITIONS

Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified copies of the Font Software, subject to the following conditions:

- 1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself.
- 2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user.
- 3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users.
- 4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission.
- 5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be distributed under any other license. The requirement for fonts to remain under this license does not apply to any document created using the Font Software.

TERMINATION

This license becomes null and void if any of the above conditions are not met.

DISCLAIMER

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.