gameplay

File Formats

Contents

Legend

Legend	
[0..1]	Zero or one. Denotes an optional property.
[0..*]	Zero or many. The property may appear many times.
[1..*]	One or many. Required property and may appear many times.
required	Required property.
Italics	Non-literal
// Comment	Just like in C++
A | B | C	A or B or C. (Shows the valid enumeration values)

Types	
block	A block property that contains other properties.
string	A string of characters with no quotations.
bool	***true*** or ***false***
float	Floating point number. Decimal is optional, don't include "f".
int	Integer
uint	Unsigned integer
axis angle	Rotation axis angle (4 floats): ***x, y, z, angle_in_degrees***
color	Hex color in the format #RRGGBBAA. (***#FF6A00FF***)
xref	External reference. This can be • a file path (***res/common/game.scene***) • a reference to a named object in a property file (***res/common/game.scene#duck***)
lua xref	External reference to a lua function.
file path	A path to a file. This should be a relative path for the purpose of being cross platform.
image path	A path to an image. • A file path (***res/wood.png***) • An alias for an image that will be defined in game config file for the purpose of using different compressed textures on different platforms. (***@wood***).

Note:

- The order of the properties does not matter.
- Blocks do not require an ***id***.

File Types

Gameplay property files		blocks
.config	Game config definitions.	window, graphics, scripts, aliases, gamepads
.scene	Scene definition	scene, node, physics
.material	Material definitions	material, technique, pass
.physics	Collision objects and physics constraints	physics, collisionObject, constraint
.animation	Animation and animation clip definitions	animation, clip
.audio	Audio source definitions	audio
.particle	Particle emitter definitions	particle
.form	UI form definitions	form
.theme	UI theme definitions	theme

Gameplay files		
.gpb	Gameplay Bundle	Binary encoded scene that is created by the gameplay-encoder from a .dae or .fbx file.
.vert	Vertex shader	GLSL vertex shader source.
.frag	Fragment shader	GLSL fragment shader source.

Other file extensions	
.dae	COLLADA
.fbx	Autodesk
.mb	Maya Binary
.ma	Maya ASCII
.ttf	TrueType Font
.lua	Lua source code
.dds, .dds, .pvr	Compressed texture

Game Config

window		
{		
width = ***pixels***	[0..1]	int
height = ***pixels***	[0..1]	int
x = ***x_offset_from_top_left***	[0..1]	int
y = ***y_offset_from_top_left***	[0..1]	int
fullscreen = ***bool***	[0..1]	bool
title = ***Hello World***	[0..1]	string
}		

graphics		
{		
samples = ***multisampling_state***	[0..1]	int
defaultMaterial = ***none | res/foo.material#bar***	[0..1]	none | file
}		

Note: By default defaultMaterial uses a pink material.

scripts		
{		
initialize = ***res/game.lua#initialize***	[0..1]	lua xref
update = ***res/game.lua#update***	[0..1]	lua xref
render = ***res/game.lua#render***	[0..1]	lua xref
finalize = ***res/game.lua#finalize***	[0..1]	lua xref
keyEvent = ***res/game.lua#keyEvent***	[0..1]	lua xref
touchEvent = ***res/game.lua#touchEvent***	[0..1]	lua xref
mouseEvent = ***res/game.lua#mouseEvent***	[0..1]	lua xref
gamepadEvent = ***res/game.lua#gamepadEvent***	[0..1]	lua xref
{		

aliases		
{		
alias_name = file_path	[0..*]	file path
}		

gamepads		
{		
form = ***res/common/gamepad.form***	[0..1]	xref
}		

Scene

```
// Load a scene from a .scene property file
Scene* scene = Scene::load("res/common/game.scene");
```

scene	[0..1]	block
{		
path = *res/game.scene*	[0..1]	xref
activeCamera = *node_id*	[0..1]	string
node *node_id* {}	[0..*]	block
physics {}	[0..1]	block
}		

Node

node *node_id* : *parent_node_id*		
{		
url = *res/common/stuff.gpb#duck*	[0..1]	xref
material = *res/scene.material#wood*	[0..1]	xref
collisionObject = *res/obj.physics#box*	[0..1]	xref
audio = *res/game.audio#quack*	[0..1]	xref
translate = *x, y, z*	[0..1]	3 floats
rotate = *x, y, z, degrees*	[0..1]	axis angle
scale = *x, y, z*	[0..1]	3 floats
tags	[0..1]	block
{		
tag_name1	[0..*]	string
tag_name2		
}		
}		

Materials

Note:

- *id* is optional for material, technique and pass.
- Materials can inherit values from another material by optionally setting a *parent_material_id*.
- Vertex and fragment shader file extensions do not matter. The convention in gameplay is to use ".vert" and ".frag".
- *scalar* is float, int or bool.
- *vector* is a comma separated list of floats.

```
// When the .material file contains one material
Material* material = model->setMaterial("res/common/box.material");

// When the .material file contains multiple materials
Material* m = model->setMaterial("res/common/stuff.material#wood");
```

material *material_id* : *parent_material_id*		
{		
uniform_name = scalar | vector	[0..*]	uniform
uniform_name = AUTO_BIND_ENUM	[0..*]	enum
sampler *uniform_name* {}	[0..*]	block
renderState {}	[0..1]	block
technique *id* {}	[0..*]	block
}		

technique *technique_id*		
{		
uniform_name = scalar | vector	[0..*]	uniform
uniform_name = AUTO_BIND_ENUM	[0..*]	enum
sampler *uniform_name* {}	[0..*]	block
renderState {}	[0..1]	block
pass *id* {}	[0..*]	block
}		

pass *pass_id*		
{		
vertexShader = *res/colored.vert*	[0..1]	file path
fragmentShader = *res/colored.frag*	[0..1]	file path
defines = *semicolon separated list*	[0..1]	string
uniform_name = scalar | vector	[0..*]	uniform
uniform_name = AUTO_BIND_ENUM	[0..*]	enum
sampler *uniform_name* {}	[0..*]	block
renderState {}	[0..1]	block
}		

sampler *uniform_name*		
{		
path = *res/wood.png | @wood*	[0..1]	image path
mipmap = *bool*	[0..1]	bool
wrapS = *REPEAT | CLAMP*	[0..1]	enum
wrapT = *REPEAT | CLAMP*	[0..1]	enum
minFilter = TEXTURE_MIN_FILTER_ENUM	[0..1]	enum
magFilter = TEXTURE_MAG_FILTER_ENUM	[0..1]	enum
}		

renderState		
{		
blend = *false*	[0..1]	bool
blendSrc = BLEND_ENUM	[0..1]	enum
blendDst = BLEND_ENUM	[0..1]	enum
cullFace = *false*	[0..1]	bool
depthTest = *false*	[0..1]	bool
depthWrite = *false*	[0..1]	bool
}		

Material Enums

AUTO_BIND_ENUM	
WORLD_MATRIX	
VIEW_MATRIX	
PROJECTION_MATRIX	
WORLD_VIEW_MATRIX	
VIEW_PROJECTION_MATRIX	
WORLD_VIEW_PROJECTION_MATRIX	
INVERSE_TRANSPOSE_WORLD_MATRIX	
INVERSE_TRANSPOSE_WORLD_VIEW_MATRIX	
CAMERA_WORLD_POSITION	
CAMERA_VIEW_POSITION	
MATRIX_PALETTE	Used for vertex skinning

TEXTURE_MIN_FILTER_ENUM	
NEAREST	Lowest quality non-mipmapped
LINEAR	Better quality non-mipmapped
NEAREST_MIPMAP_NEAREST	Fast but low quality mipmapping
LINEAR_MIPMAP_NEAREST	
NEAREST_MIPMAP_LINEAR	
LINEAR_MIPMAP_LINEAR	Best quality mipmapping

TEXTURE_MAG_FILTER_ENUM	
NEAREST	Lowest quality
LINEAR	Better quality

BLEND_ENUM	
ZERO	ONE_MINUS_DST_ALPHA
ONE	CONSTANT_ALPHA
SRC_ALPHA	ONE_MINUS_CONSTANT_ALPHA
ONE_MINUS_SRC_ALPHA	SRC_ALPHA_SATURATE
DST_ALPHA	

Physics

physics		
{		
gravity = *x, y, z*	[0..1]	3 floats
constraint {}	[0..1]	block
}		

Default gravity: *0.0, -9.8, 0.0*

Collision Objects

Physics Rigid Body

collisionObject *id*		
{		
type = **RIGID_BODY**	required	enum
shape = *BOX | SPHERE | MESH | CAPSULE | HEIGHTFIELD*	required	enum
mass = *kilograms*	[0..1]	float
friction = *coefficient*	[0..1]	float
restitution = *coefficient*	[0..1]	float
linearDamping = *coefficient*	[0..1]	float
angularDamping = *coefficient*	[0..1]	float
kinematic = *bool*	[0..1]	bool
anisotropicFriction = *x, y, z*	[0..1]	3 floats
gravity = *x, y, z*	[0..1]	3 floats
// BOX properties		
extents = *x, y, z*	[0..1]	3 floats
center = *x, y, z*	[0..1]	3 floats
centerAbsolute = *x, y, z*	[0..1]	3 floats
// SPHERE properties		
radius = *float*	[0..1]	float
center = *x, y, z*	[0..1]	3 floats
centerAbsolute = *x, y, z*	[0..1]	3 floats
// CAPSULE properties		
radius = *float*	[0..1]	float
height = *float*	[0..1]	float
center = *x, y, z*	[0..1]	3 floats
centerAbsolute = *x, y, z*	[0..1]	3 floats
// HEIGHTFIELD properties		
image = *res/common/image.png*	[0..1]	xref
}		

Physics Ghost Object

collisionObject *id*		
{		
type = **GHOST_OBJECT**	required	enum
shape = ***BOX | SPHERE | MESH | CAPSULE | HEIGHTFIELD***	required	enum
// BOX properties		
extents = ***x, y, z***	[0..1]	3 floats
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// SPHERE properties		
radius = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// CAPSULE properties		
radius = ***float***	[0..1]	float
height = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// HEIGHTFIELD properties		
image = ***res/common/image.png***	[0..1]	xref
}		

Physics Character

collisionObject *id*		
{		
type = **CHARACTER**	required	enum
shape = ***BOX | SPHERE | MESH | CAPSULE***	required	enum
mass = ***kilograms***	[0..1]	float
// BOX properties		
extents = ***x, y, z***	[0..1]	3 floats
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// SPHERE properties		
radius = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// CAPSULE properties		
radius = ***float***	[0..1]	float
height = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
}		

Physics Vehicle

collisionObject *id*		
{		
type = **VEHICLE**	required	enum
shape = ***BOX* | *SPHERE* | *MESH* | *CAPSULE***	required	enum
mass = ***kilograms***	[0..1]	float
friction = ***coefficient***	[0..1]	float
restitution = ***coefficient***	[0..1]	float
linearDamping = ***coefficient***	[0..1]	float
angularDamping = ***coefficient***	[0..1]	float
kinematic = ***bool***	[0..1]	bool
anisotropicFriction = ***x, y, z***	[0..1]	3 floats
gravity = ***x, y, z***	[0..1]	3 floats
// BOX properties		
extents = ***x, y, z***	[0..1]	3 floats
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// SPHERE properties		
radius = ***radius***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// CAPSULE properties		
radius = ***float***	[0..1]	float
height = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// VEHICLE only		
steeringGain = ***float***	[0..1]	float
brakingForce = ***float***	[0..1]	float
drivingForce = ***float***	[0..1]	float
steerdownSpeed = ***float***	[0..1]	float
steerdownGain = ***float***	[0..1]	float
brakedownStart = ***float***	[0..1]	float
brakedownFull = ***float***	[0..1]	float
drivedownStart = ***float***	[0..1]	float
drivedownFull = ***float***	[0..1]	float
boostSpeed = ***float***	[0..1]	float
boostGain = ***float***	[0..1]	float
downforce = ***float***	[0..1]	float
}		

Physics Vehicle Wheel

collisionObject *id*		
{		
type = **VEHICLE_WHEEL**	required	enum
shape = ***BOX* | *SPHERE* | *MESH* | *CAPSULE***	required	enum
mass = ***kilograms***	[0..1]	float
friction = ***coefficient***	[0..1]	float
restitution = ***coefficient***	[0..1]	float

linearDamping = ***coefficient***	[0..1]	float
angularDamping = ***coefficient***	[0..1]	float
kinematic = ***bool***	[0..1]	bool
anisotropicFriction = ***x, y, z***	[0..1]	3 floats
gravity = ***x, y, z***	[0..1]	3 floats
// BOX properties		
extents = ***x, y, z***	[0..1]	3 floats
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// SPHERE properties		
radius = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// CAPSULE properties		
radius = ***float***	[0..1]	float
height = ***float***	[0..1]	float
center = ***x, y, z***	[0..1]	3 floats
centerAbsolute = ***x, y, z***	[0..1]	3 floats
// VEHICLE_WHEEL only		
steerable = ***false***	[0..1]	bool
wheelDirection = ***x, y, z***	[0..1]	3 floats
wheelAxle = ***x, y, z***	[0..1]	3 floats
strutConnectionOffset = ***x, y, z***	[0..1]	3 floats
strutRestLength = ***float***	[0..1]	float
strutStiffness = ***coefficient***	[0..1]	float
strutDampingCompression = ***coefficient***	[0..1]	float
strutDampingRelaxation = ***coefficient***	[0..1]	float
frictionBreakout = ***float***	[0..1]	float
wheelRadius = ***radius***	[0..1]	float
rollInfluence = ***float***	[0..1]	float
}		

Constraints

Fixed Constraint

constraint ***id***		
{		
type = **FIXED**	required	enum
breakingImpulse = ***float***	[0..1]	float
}		

Generic Constraint

constraint ***id***		
{		
type = **GENERIC**	required	enum
translationOffsetA = ***x, y, z***	[0..1]	3 floats
translationOffsetB = ***x, y, z***	[0..1]	3 floats
rotationOffsetA = ***x, y, z, degrees***	[0..1]	axis angle
rotationOffsetB = ***x, y, z, degrees***	[0..1]	axis angle
angularLowerLimit = ***x, y, z***	[0..1]	3 floats

angularUpperLimit = ***x, y, z***	[0..1]	3 floats
linearLowerLimit = ***x, y, z***	[0..1]	3 floats
linearUpperLimit = ***x, y, z***	[0..1]	3 floats
breakingImpulse = ***float***	[0..1]	float
}		

Hinge Constraint

constraint ***id***		
{		
type = **HINGE**	required	enum
translationOffsetA = ***x, y, z***	[0..1]	3 floats
translationOffsetB = ***x, y, z***	[0..1]	3 floats
rotationOffsetA = ***x, y, z, degrees***	[0..1]	axis angle
rotationOffsetB = ***x, y, z, degrees***	[0..1]	axis angle
limits = ***lowerLimit, upperLimit, bounciness***	[0..1]	2 or 3 floats
breakingImpulse = ***float***	[0..1]	float
}		

Socket Constraint

constraint ***id***		
{		
type = **SOCKET**	required	enum
translationOffsetA = ***x, y, z***	[0..1]	3 floats
translationOffsetB = ***x, y, z***	[0..1]	3 floats
breakingImpulse = ***float***	[0..1]	float
}		

Sprint Constraint

constraint ***id***		
{		
type = **SPRING**	required	enum
translationOffsetA = ***x, y, z***	[0..1]	3 floats
translationOffsetB = ***x, y, z***	[0..1]	3 floats
rotationOffsetA = ***x, y, z, degrees***	[0..1]	axis angle
rotationOffsetB = ***x, y, z, degrees***	[0..1]	axis angle
angularLowerLimit = ***x, y, z***	[0..1]	3 floats
angularUpperLimit = ***x, y, z***	[0..1]	3 floats
linearLowerLimit = ***x, y, z***	[0..1]	3 floats
linearUpperLimit = ***x, y, z***	[0..1]	3 floats
angularDampingX = ***damping***	[0..1]	float
angularDampingY = ***damping***	[0..1]	float
angularDampingZ = ***damping***	[0..1]	float
angularStrengthX = ***strength***	[0..1]	float
angularStrengthY = ***strength***	[0..1]	float
angularStrengthZ = ***strength***	[0..1]	float
linearDampingX = ***damping***	[0..1]	float
linearDampingY = ***damping***	[0..1]	float
linearDampingZ = ***damping***	[0..1]	float
linearStrengthX = ***strength***	[0..1]	float
linearStrengthY = ***strength***	[0..1]	float
linearStrengthZ = ***strength***	[0..1]	float
breakingImpulse = ***float***	[0..1]	float
}		

Animation

animation *animation_id*		
{		
property = ANIMATION_PROPERTY	required	enum
keyCount = *number_of_key_frames*	[0..1]	int
keyTimes = *uint uint uint uint ...*	[0..1]	uint(s)
keyValues = *float float float float ...*	[0..1]	float(s)
curve = INTERPOLATION_TYPE	required	enum
}		

```
// Load the animation clips for sample03-character
Animation* animation = node->getAnimation("animations");
animation->createClips("res/common/boy.animation");
```

animation *animation_id*		
{		
frameCount = *frame_count*	required	int
clip *clip_id* {}	[0..*]	block
}		

Animation Clip

clip *clip_id*		
{		
begin = *frame_index*	[0..1]	int
end = *frame_index*	[0..1]	int
repeatCount = *float | REPEAT_INDEFINITE*	[0..1]	float | enum
speed = *1.0*	[0..1]	float
}		

Animation Enums

ANIMATION_PROPERTY	
ANIMATE_SCALE	ANIMATE_TRANSLATE
ANIMATE_SCALE_X	ANIMATE_TRANSLATE_X
ANIMATE_SCALE_Y	ANIMATE_TRANSLATE_Y
ANIMATE_SCALE_Z	ANIMATE_TRANSLATE_Z
ANIMATE_ROTATE	ANIMATE_ROTATE_TRANSLATE
ANIMATE_UNIFORM	ANIMATE_SCALE_ROTATE_TRANSLATE

INTERPOLATION_TYPE		
BEZIER	QUARTIC_IN	CIRCULAR_IN
BSPLINE	QUARTIC_OUT	CIRCULAR_OUT
FLAT	QUARTIC_IN_OUT	CIRCULAR_IN_OUT
HERMITE	QUARTIC_OUT_IN	CIRCULAR_OUT_IN
LINEAR	QUINTIC_IN	ELASTIC_IN
SMOOTH	QUINTIC_OUT	ELASTIC_OUT
STEP	QUINTIC_IN_OUT	ELASTIC_IN_OUT
QUADRATIC_IN	QUINTIC_OUT_IN	ELASTIC_OUT_IN
QUADRATIC_OUT	SINE_IN	OVERSHOOT_IN
QUADRATIC_IN_OUT	SINE_OUT	OVERSHOOT_OUT
QUADRATIC_OUT_IN	SINE_IN_OUT	OVERSHOOT_IN_OUT
CUBIC_IN	SINE_OUT_IN	OVERSHOOT_OUT_IN
CUBIC_OUT	EXPONENTIAL_IN	BOUNCE_IN

CUBIC_IN_OUT	EXPONENTIAL_OUT	BOUNCE_OUT
CUBIC_OUT_IN	EXPONENTIAL_IN_OUT	BOUNCE_IN_OUT
	EXPONENTIAL_OUT_IN	BOUNCE_OUT_IN

Audio

```
// Load a sound from file
AudioSource* source = AudioSource::create("res/game.audio#explode");
```

Audio Source

audio *audio_id*		
{		
path = *res/common/engine_loop.ogg*	[0..1]	file path
looped = *false*	[0..1]	bool
gain = *volume_amplification (1.0)*	[0..1]	float
pitch = *pitch_value [0.5-2.0]*	[0..1]	float
velocity = *x, y, z*	[0..1]	3 floats
}		

Particles

```
// Load a particle emitter from a property file
ParticleEmitter* p = ParticleEmitter::create("res/fire.particle");
```

particle *particle_id*		
{		
sprite	[0..1]	block
{		
path = ***res/common/smoke.png***	[0..1]	file path
width = ***int***	[0..1]	int
height = ***int***	[0..1]	int
frameCount = ***count***	[0..1]	int
frameDuration = ***seconds***	[0..1]	float
frameRandomOffset = ***offset***	[0..1]	int
looped = ***false***	[0..1]	bool
animated = ***false***	[0..1]	bool
blending = ***OPAQUE | TRANSPARENT | ADDITIVE | MULTIPLIED***	[0..1]	enum
}		
particleCountMax = ***100***	[0..1]	uint
emissionRate = ***particles_per_second***	[0..1]	uint
orbitPosition = ***false***	[0..1]	bool
orbitVelocity = ***false***	[0..1]	bool
orbitAcceleration = ***false***	[0..1]	bool
ellipsoid = ***false***	[0..1]	bool
sizeStartMin = ***1.0***	[0..1]	float
sizeStartMax = ***1.0***	[0..1]	float
sizeEndMin = ***1.0***	[0..1]	float
sizeEndMax = ***1.0***	[0..1]	float
energyMin = ***milliseconds***	[0..1]	float
energyMax = ***milliseconds***	[0..1]	float
rotationPerParticleSpeedMin = ***radians per second***	[0..1]	float
rotationPerParticleSpeedMax = ***radians per second***	[0..1]	float
colorStart = ***red, blue, green, alpha***	[0..1]	4 floats
colorEnd = ***red, blue, green, alpha***	[0..1]	4 floats
position = ***x, y, z***	[0..1]	3 floats
positionVar = ***x, y, z***	[0..1]	3 floats
velocity = ***x, y, z***	[0..1]	3 floats
velocityVar = ***x, y, z***	[0..1]	3 floats
acceleration = ***x, y, z***	[0..1]	3 floats
accelerationVar = ***x, y, z***	[0..1]	3 floats
}		

UI Forms

```
// Load a form from a property file
Form* form = Form::create("res/editor.form");
```

Form

form ***form_id***		
{		
theme = ***res/editor.theme***	required	xref
layout = LAYOUT_ENUM	required	enum
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
alignment = ALIGNMENT_ENUM	[0..1]	enum
size = ***width, height***	[0..1]	2 floats
autoWidth = ***false***	[0..1]	bool
autoHeight = ***false***	[0..1]	bool
width = ***width***	[0..1]	float
height = ***height***	[0..1]	float
consumeInputEvents = ***true***	[0..1]	bool
container ***container_id*** {}	[0..*]	block
label ***label_id*** {}	[0..*]	block
textBox ***textBox_id*** {}	[0..*]	block
button ***button_id*** {}	[0..*]	block
checkBox ***checkBox_id*** {}	[0..*]	block
radioButton ***radioButton_id*** {}	[0..*]	block
slider ***slider_id*** {}	[0..*]	block
}		

Container

container ***container_id***		
{		
layout = LAYOUT_ENUM	required	enum
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
alignment = ALIGNMENT_ENUM	[0..1]	enum
size = ***width, height***	[0..1]	2 floats
autoWidth = ***false***	[0..1]	bool
autoHeight = ***false***	[0..1]	bool
width = ***width***	[0..1]	float
height = ***height***	[0..1]	float
consumeInputEvents = ***true***	[0..1]	bool
container ***container_id*** {}	[0..*]	block
label ***label_id*** {}	[0..*]	block
textBox ***textBox_id*** {}	[0..*]	block
button ***button_id*** {}	[0..*]	block
checkBox ***checkBox_id*** {}	[0..*]	block
radioButton ***radioButton_id*** {}	[0..*]	block
slider ***slider_id*** {}	[0..*]	block
}		

Label

label ***label_id***		
{		
text = ***Hello World***	[0..1]	string
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
alignment = ALIGNMENT_ENUM	[0..1]	enum
size = ***width, height***	[0..1]	2 floats
autoWidth = ***false***	[0..1]	bool
autoHeight = ***false***	[0..1]	bool
consumeInputEvents = ***true***	[0..1]	bool
}		

TextBox

textBox ***textBox_id***		
{		
... same as label		
}		

Button

button ***button_id***		
{		
... same as label		
}		

CheckBox

checkBox ***checkBox_id***		
{		
checked = ***false***	[0..1]	bool
text = ***Hello World***	[0..1]	string
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
alignment = ALIGNMENT_ENUM	[0..1]	enum
size = ***width, height***	[0..1]	2 floats
autoWidth = ***false***	[0..1]	bool
autoHeight = ***false***	[0..1]	bool
consumeInputEvents = ***true***	[0..1]	bool
}		

RadioButton

radioButton ***radioButton_id***		
{		
group = ***group_id***	[0..1]	string
checked = ***false***	[0..1]	bool
text = ***Hello World***	[0..1]	string
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
alignment = ALIGNMENT_ENUM	[0..1]	enum
size = ***width, height***	[0..1]	2 floats
autoWidth = ***false***	[0..1]	bool
autoHeight = ***false***	[0..1]	bool
imageSize = ***width, height***	[0..1]	2 floats
consumeInputEvents = ***true***	[0..1]	bool
}		

Slider

slider ***slider_id***		
{		
style = ***style_id***	[0..1]	string
position = ***x, y***	[0..1]	2 floats
size = ***width, height***	[0..1]	2 floats
min = ***float***	[0..1]	float
max = ***float***	[0..1]	float
value = ***default_value***	[0..1]	float
step = ***discrete_steps***	[0..1]	float
text = ***Hello World***	[0..1]	string
consumeInputEvents = ***true***	[0..1]	bool
}		

Form Enums

LAYOUT_ENUM	
LAYOUT_FLOW	Controls are placed next to one another horizontally until the right-most edge of the container is reached, at which point a new row is started.
LAYOUT_VERTICAL	Controls are placed next to one another vertically until the bottom-most edge of the container is reached.
LAYOUT_ABSOLUTE	Controls are not modified at all by this layout. They must be positioned and sized manually.

ALIGNMENT_ENUM	
ALIGN_LEFT	Left
ALIGN_HCENTER	Horizontal center
ALIGN_RIGHT	Right
ALIGN_TOP	Top
ALIGN_VCENTER	Vertical center
ALIGN_BOTTOM	Bottom
ALIGN_TOP_LEFT	ALIGN_TOP | ALIGN_LEFT
ALIGN_VCENTER_LEFT	ALIGN_VCENTER | ALIGN_LEFT
ALIGN_BOTTOM_LEFT	ALIGN_BOTTOM | ALIGN_LEFT
ALIGN_TOP_HCENTER	ALIGN_TOP | ALIGN_HCENTER
ALIGN_VCENTER_HCENTER	ALIGN_VCENTER | ALIGN_HCENTER
ALIGN_BOTTOM_HCENTER	ALIGN_BOTTOM | ALIGN_HCENTER
ALIGN_TOP_RIGHT	ALIGN_TOP | ALIGN_RIGHT
ALIGN_VCENTER_RIGHT	ALIGN_VCENTER | ALIGN_RIGHT
ALIGN_BOTTOM_RIGHT	ALIGN_BOTTOM | ALIGN_RIGHT

Theme

theme ***theme_id***		
{		
texture = ***image_path***	required	image path
cursor ***cursor_id*** {}	[0..*]	block
imageList ***imageList_id*** {}	[0..*]	block
skin ***skin_id*** {}	[0..*]	block
style ***style_id*** {}	[0..*]	block
}		

cursor ***cursor_id***		
{		
region = ***x, y, width, height***	[0..1]	4 floats
color = ***#RRGGBBAA***	[0..1]	color
}		

imageList ***imageList_id***		
{		
color = ***#RRGGBBAA***	[0..1]	color
image ***image_id*** {}	[0..*]	block
}		

skin ***skin_id***		
{		
region = ***x, y, width, height***	[0..1]	4 floats
color = ***#RRGGBBAA***	[0..1]	color
border	[0..1]	block
{		
top = ***top***	[0..1]	int
bottom = ***bottom***	[0..1]	int
left = ***left***	[0..1]	int
right = ***right***	[0..1]	int
}		
}		

style ***style_id***		
{		
margin	[0..1]	block
{		
top = ***top***	[0..1]	int
bottom = ***bottom***	[0..1]	int
left = ***left***	[0..1]	int
right = ***right***	[0..1]	int
}		
padding	[0..1]	block
{		
top = ***top***	[0..1]	int
bottom = ***bottom***	[0..1]	int
left = ***left***	[0..1]	int
right = ***right***	[0..1]	int
}		
stateNormal {}	[0..1]	block
stateFocus {}	[0..1]	block
stateActive {}	[0..1]	block
stateDisabled {}	[0..1]	block
}		

stateNormal		
{		
skin = ***skin_id***	[0..1]	string
imageList = ***imageList_id***	[0..1]	string
cursor = ***cursor_id***	[0..1]	string

font = ***res/common/arial40.gpb***	[0..1]	file path
fontSize = ***fontSize***	[0..1]	int
textColor = ***#RRGGBBAA***	[0..1]	color
textAlignment = ALIGNMENT_ENUM	[0..1]	enum
rightToLeft = ***false***	[0..1]	bool
opacity = ***float***	[0..1]	float
}		

stateFocus		
{		
... same as stateNormal		
}		

stateActive		
{		
... same as stateNormal		
}		

stateDisabled		
{		
... same as stateNormal		
}		

License

The project is open sourced under the Apache 2.0 license.

Disclaimer

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Published in Canada