DSdl Framwork

Version 2.0

4/5/2016 8:37:00 PM

Table of Contents

Namespace Index

Namespace List

Here is a list of all namespaces with brief descriptions:

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Namespace Documentation

DsdlEngine Namespace Reference

Classes

- class AudioManager
- class Button
- class CollisionShape
- class DsdlGui
- class EngineBaseNode
- class EngineMaster
- class FileIO
- class FpsLimiter
- class IMainGame
- class InputManager
- class IScene
- class Label
- class Layer
- class Music
- class Particles
- class ResourceTexture
- class SceneManager
- class SFX
- class Size
- class Sprite
- class Vec2
- class Window
- class XmlLocalStorage

Typedefs

- typedef SDL_TimerID CallBackTimer
- typedef SDL_TimerCallback CallBack

Enumerations

- enum NodeType { NodeType::BASENODE, NodeType::SPRITE, NodeType::LABEL, NodeType::BUTTON, NodeType::PARTICLE }
- enum ButtonState { ButtonState::NORMAL, ButtonState::PRESSED, ButtonState::HOVERING }
- enum ButtonType { ButtonType::LABEL_BTN, ButtonType::SPRITE_BTN }
- enum LableType { LableType::LABEL_STATIC, LableType::LABEL_DYNAMIC }
- enum SceneState { SceneState::NONE, SceneState::RUNNING, SceneState::EXIT_APP, SceneState::CHANGE_NEXT, SceneState::CHANGE_PREVIOUS }

Functions

- int init ()
- template<typename T , typename... Args> std::unique_ptr< T > make_unique (Args &&...args)

Variables

- static EngineMaster * Instance = nullptr

- static FileIO * Instance = nullptr
- static XmlLocalStorage * Instance = nullptr

Detailed Description

Author:

Derek O Brien
Derek O Brien

Class Based of https://www.youtube.com/watch?v=Epyih-LEbig&list=PLSPw4ASQYyymu3PfG9gxywSPghnSMiOAW&index=26 tutorial

Typedef Documentation

typedef SDL_TimerCallback DsdlEngine::CallBack

Definition at line 75 of file EngineDefines.h.

typedef SDL_TimerID DsdlEngine::CallBackTimer

Timer Call Back

Definition at line 74 of file EngineDefines.h.

Enumeration Type Documentation

enum DsdlEngine::ButtonState `[strong]`

Button State Enum

Enumerator

NORMAL
PRESSED
HOVERING

Definition at line 49 of file EngineDefines.h.

enum DsdlEngine::ButtonType `[strong]`

Button Type Enum

Enumerator

LABEL_BTN
SPRITE_BTN

Definition at line 58 of file EngineDefines.h.

enum DsdlEngine::LableType `[strong]`

Label Type Enum

Enumerator

LABEL_STATIC
LABEL_DYNAMIC

Definition at line 66 of file EngineDefines.h.

enum DsdlEngine::NodeType `[strong]`

Node Type Enum

Enumerator

BASENODE
SPRITE
LABEL
BUTTON
PARTICLE

Definition at line 38 of file EngineDefines.h.

enum DsdlEngine::SceneState `[strong]`

SceneState enum class. For use when controlling the which scene is active..

Enumerator

NONE
RUNNING
EXIT_APP
CHANGE_NEXT
CHANGE_PREVIOUS

Definition at line 24 of file IScene.h.

Function Documentation

int DsdlEngine::init ()

init, Initalize SDL

Definition at line 7 of file DsdlEngine.cpp.

template<typename T , typename... Args> std::unique_ptr<T> DsdlEngine::make_unique (Args &&... *args*)

Definition at line 17 of file IMainGame.cpp.

Variable Documentation

EngineMaster* DsdlEngine::Instance = nullptr `[static]`

Definition at line 6 of file EngineMaster.cpp.

FileIO* DsdlEngine::Instance = nullptr`[static]`

Definition at line 11 of file FileIO.cpp.

XmlLocalStorage* DsdlEngine::Instance = nullptr`[static]`

Definition at line 16 of file XmlLocalStorage.cpp.

Class Documentation

DsdlEngine::AudioManager Class Reference

```
#include <AudioManager.h>
```

Public Member Functions

- AudioManager ()
- ~AudioManager ()
- void init ()
- void destroy ()
- SFX loadSFX (std::string audioPath)
- Music loadMusic (std::string audioPath)

Private Attributes

- std::map< std::string, Mix_Chunk * > m_sfxAudioMap
- std::map< std::string, Mix_Music * > m_bgAudioMap
- bool m_bisInitialized

Detailed Description

AudioManager Class. The AudioManager Class is responsable for loading and playing of audio within a game. The AudioManager will also cache any loaded audio.

Definition at line 81 of file AudioManager.h.

Constructor & Destructor Documentation

DsdlEngine::AudioManager::AudioManager () `[inline]`

AudioManager Constructor. On call will init the AudioManager.

Definition at line 87 of file AudioManager.h.

DsdlEngine::AudioManager::~AudioManager () `[inline]`

AudioManager Deconstructor. On call will destroy current copy of the AudioManager and clear current cache

Definition at line 93 of file AudioManager.h.

Member Function Documentation

void DsdlEngine::AudioManager::destroy ()

Destroy current copy of AudioManager and clear cache maps.

Definition at line 24 of file AudioManager.cpp.

void DsdlEngine::AudioManager::init ()

init SDL Audio and set up Audio channels and frequency. Automaticaly called by Constructor.

Definition at line 10 of file AudioManager.cpp.

Music DsdlEngine::AudioManager::loadMusic (std::string *audioPath*)

Load Music.

Parameters:

std::string	for path to audio.

Returns:

Music to play.

Definition at line 87 of file AudioManager.cpp.

SFX DsdlEngine::AudioManager::loadSFX (std::string *audioPath*)

Load Sound effect.

Parameters:

std::string	for path to audio.

Returns:

SFX chunk to play.

Definition at line 56 of file AudioManager.cpp.

Member Data Documentation

std::map<std::string, Mix_Music*> DsdlEngine::AudioManager::m_bgAudioMap `[private]`

Private cache map for storing musics.

Definition at line 129 of file AudioManager.h.

bool DsdlEngine::AudioManager::m_bisInitialized `[private]`

Private bool for control loading.

Definition at line 134 of file AudioManager.h.

std::map<std::string, Mix_Chunk*> DsdlEngine::AudioManager::m_sfxAudioMap `[private]`

Private cache map for storing sound effects.

Definition at line 124 of file AudioManager.h.

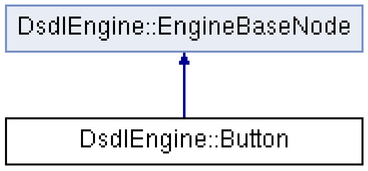
The documentation for this class was generated from the following files:

- AudioManager.h
- AudioManager.cpp

DsdlEngine::Button Class Reference

```
#include <Button.h>
```

Inheritance diagram for DsdlEngine::Button:



Public Member Functions

- Button ()
- virtual ~Button ()
- void destroy ()
- void createTextButton (Vec2 pos, Size size, std::string buttonText, std::string fontPath, SDL_Color textColor, SDL_Color bgColor)
- void createSpriteButton (Vec2 spriteSize, Vec2 position, std::string imagePath, std::string name)
- void checkInput (SDL_Event &e)
- std::string getButtonName ()

Public Attributes

- ButtonState m_eCurrentState

Private Member Functions

- void onMouseEnters ()
- void onMouseLeaves ()
- void onClicked ()

Private Attributes

- Label * m_label
- Sprite * m_spriteBtn
- std::string m_buttonName

Additional Inherited Members

Detailed Description

Button Class subclass of EngineBaseNode. The button class is for creating buttons and handling events on such buttons.

Definition at line 19 of file Button.h.

Constructor & Destructor Documentation

DsdlEngine::Button::Button ()

Button Constructor.

Definition at line 6 of file Button.cpp.

DsdlEngine::Button::~Button () [virtual]

Button Deconstructor.

Definition at line 11 of file Button.cpp.

Member Function Documentation

void DsdlEngine::Button::checkInput (SDL_Event & *e*)

checkInput. Check for input event on the current button.

Parameters:

e	as SDL_Event argument.

Definition at line 70 of file Button.cpp.

void DsdlEngine::Button::createSpriteButton (Vec2 *spriteSize*, Vec2 *position*, std::string *imagePath*, std::string *name*)

Create button as a Sprite node.

Parameters:

spriteSize	as a Vec2 argument.
position	as a Vec2 position argument
imagePath	as a std::string path to image
name	as a std::string name of button

Definition at line 33 of file Button.cpp.

void DsdlEngine::Button::createTextButton (Vec2 *pos*, Size *size*, std::string *buttonText*, std::string *fontPath*, SDL_Color *textColor*, SDL_Color *bgColor*)

Create button as Label node.

Parameters:

pos	as a Vec2 position argument.
size	as a Size content size argument.
buttonText	as a std::string argument.
fontPath	as a std::string argument.
textColor	as a SDL_Color argument.
bgColor	as a SDL_Color argument.

Definition at line 14 of file Button.cpp.

void DsdlEngine::Button::destroy () [virtual]

destroy. responsible for cleaning up after button gose out of scope.

Reimplemented from DsdlEngine::EngineBaseNode.

Definition at line 127 of file Button.cpp.

std::string DsdlEngine::Button::getButtonName () [inline]

getButtonName. Get the name of the button.

Returns:

std::string name of the button

Definition at line 70 of file Button.h.

void DsdlEngine::Button::onClicked () `[private]`

onCLicked. Set button state to CLICKED.

Definition at line 64 of file Button.cpp.

void DsdlEngine::Button::onMouseEnters () `[private]`

onMouseEnters. Set button state to HOVERING.

Definition at line 53 of file Button.cpp.

void DsdlEngine::Button::onMouseLeaves () `[private]`

onMouseLeaves. Set button state to NORMAL.

Definition at line 59 of file Button.cpp.

Member Data Documentation

std::string DsdlEngine::Button::m_buttonName `[private]`

std::string button name variable.

Definition at line 111 of file Button.h.

ButtonState DsdlEngine::Button::m_eCurrentState

ButtonState. Enum class for handeling the buttons state.

Definition at line 76 of file Button.h.

Label* DsdlEngine::Button::m_label `[private]`

Label varaible for creating label buttons.

Definition at line 101 of file Button.h.

Sprite* DsdlEngine::Button::m_spriteBtn `[private]`

Sprite variable for creating sprite button.

Definition at line 106 of file Button.h.

The documentation for this class was generated from the following files:

- Button.h
- Button.cpp

DsdlEngine::CollisionShape Class Reference

```
#include <CollisionShape.h>
```

Public Member Functions

- CollisionShape ()
- ~CollisionShape ()
- void init (b2World *world, Vec2 position, Vec2 dimensions, float density, float friction, bool fixedRotation)
- void destroy (b2World *world)
- b2Body * getBody () const
- b2Fixture * getFixture (int index) const
- const Vec2 getDimensions () const

Protected Attributes

- b2Body * m_body = nullptr
- b2Fixture * m_fixtures [1]
- Vec2 m_dimensions

Detailed Description

CollisionShape class is for creating a Box2D collision shape around the Sprite node.

Definition at line 12 of file CollisionShape.h.

Constructor & Destructor Documentation

DsdlEngine::CollisionShape::CollisionShape ()

Constructor.

Definition at line 6 of file CollisionShane.cpp.

DsdlEngine::CollisionShape::~CollisionShape ()

Deconstructor.

Definition at line 11 of file CollisionShane.cpp.

Member Function Documentation

void DsdlEngine::CollisionShape::destroy (b2World * *world*)

destroy shape in the Box2D world.

Parameters:

world	as a b2World pointer argument.

Definition at line 45 of file CollisionShane.cpp.

b2Body* DsdlEngine::CollisionShape::getBody () const `[inline]`

getBody, get the body for the shape.

Returns:

b2Body pointer.

Definition at line 50 of file CollisionShape.h.

const Vec2 DsdlEngine::CollisionShape::getDimensions () const `[inline]`

getDimensions, get the dimensions of the shape.

Returns:

Vec2 dimesions of the shape.

Definition at line 63 of file CollisionShape.h.

b2Fixture* DsdlEngine::CollisionShape::getFixture (int *index*) const `[inline]`

getFixture, get the fixture for index passed in.

Parameters:

index	as a int argument.

Returns:

b2Fixture pointer.

Definition at line 57 of file CollisionShape.h.

void DsdlEngine::CollisionShape::init (b2World * *world*, Vec2 *position*, Vec2 *dimensions*, float *density*, float *friction*, bool *fixedRotation*)

Initialize shape with arguments passed in.

Parameters:

position	as a Vec2 position argument.
dimensions	as a Vec2 size arguent.
density	as a float argument.
friction	as a float argument.
fixedRotation	as a bool argument.

Definition at line 16 of file CollisionShane.cpp.

Member Data Documentation

b2Body* DsdlEngine::CollisionShape::m_body = nullptr `[protected]`

b2Body variabel.

Definition at line 69 of file CollisionShape.h.

Vec2 DsdlEngine::CollisionShape::m_dimensions `[protected]`

Vec2 dimensions variable.

Definition at line 79 of file CollisionShape.h.

b2Fixture* DsdlEngine::CollisionShape::m_fixtures[1] `[protected]`

Array of fixtures for the shape.

Definition at line 74 of file CollisionShape.h.

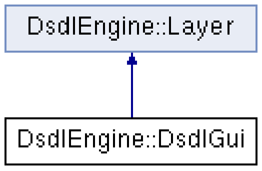
The documentation for this class was generated from the following files:

- CollisionShape.h
- CollisionShane.cpp

DsdlEngine::DsdlGui Class Reference

`#include <Gui.h>`

Inheritance diagram for DsdlEngine::DsdlGui:



Public Member Functions

- DsdlGui ()
- virtual ~DsdlGui ()
- void addButton (ButtonType type, std::string name, Vec2 pos, Vec2 size, std::string path, SDL_Color color, SDL_Color bgColor, const char *text=NULL)
- void addLabel (LableType type, Vec2 pos, std::string text, int fontSize, SDL_Color color, std::string fontFilePath)
- void addPreDefineLabel (Label *label, LableType type)
- void setGUIPos ()
- void onSDLEvent (SDL_Event &e)
- void destroy ()
- Button * getButton ()

Public Attributes

- std::vector< Button * > GUIElements

Protected Attributes

- Label * m_label
- Button * m_btn

Detailed Description

GUi Layer templtate for creating an a UI Layer. Inherits from layer

Definition at line 20 of file Gui.h.

Constructor & Destructor Documentation

DsdlEngine::DsdlGui::DsdlGui ()

Constructor

Definition at line 14 of file Gui.cpp.

DsdlEngine::DsdlGui::~DsdlGui () `[virtual]`

Deconstructor

Definition at line 19 of file Gui.cpp.

Member Function Documentation

void DsdlEngine::DsdlGui::addButton (ButtonType *type*, std::string *name*, Vec2 *pos*, Vec2 *size*, std::string *path*, SDL_Color *color*, SDL_Color *bgColor*, const char * *text* = `NULL`)

addButton, Creates and adds a button to the UI layer.

Parameters:

type,type	of button as a ButtonType
name,std::string	name of the button
pos,Vec2	position of the button
size,Vec2	size of the button
path,path	to texture to load.
color,SDL_color	of the button. for label type
bgColor,background	color of the button. for label type.
text,text	to display. for label type.

Definition at line 24 of file Gui.cpp.

void DsdlEngine::DsdlGui::addLabel (LableType *type*, Vec2 *pos*, std::string *text*, int *fontSize*, SDL_Color *color*, std::string *fontFilePath*)

addLabel, Creates and adds a Label to the UI layer.

Parameters:

type,type	of labe as a LabelType
pos,Vec2	position of the label
text,text	to display. for label type.
fontsize,as	int size of font.
color,SDL_color	of the label.
fontFilePath,file	path to the font.

Definition at line 46 of file Gui.cpp.

void DsdlEngine::DsdlGui::addPreDefineLabel (Label * *label*, LableType *type*)

addPredefinedLabel, add a pre made label to the UI layer,

Parameters:

label,the	Label to be added.
type,the	type of label.

Definition at line 57 of file Gui.cpp.

void DsdlEngine::DsdlGui::destroy () `[virtual]`

destroy, Clean up when left scope.

Reimplemented from DsdlEngine::Layer.

Definition at line 76 of file Gui.cpp.

Button* DsdlEngine::DsdlGui::getButton () `[inline]`

getButton, Get button from the UIElemets vector

Returns:

Button.

Definition at line 85 of file Gui.h.

void DsdlEngine::DsdlGui::onSDLEvent (SDL_Event & *e*)

onSDLEvent, evnet listener for GUI buttons.

Parameters:

e,evnent	to listen on.

Definition at line 68 of file Gui.cpp.

void DsdlEngine::DsdlGui::setGUIPos ()

setGUIPos, set the GUI position

Definition at line 63 of file Gui.cpp.

Member Data Documentation

std::vector<Button*> DsdlEngine::DsdlGui::GUIElements

Vector to hold GUI Elements

Definition at line 79 of file Gui.h.

Button* DsdlEngine::DsdlGui::m_btn `[protected]`

Definition at line 90 of file Gui.h.

Label* DsdlEngine::DsdlGui::m_label `[protected]`

Definition at line 89 of file Gui.h.

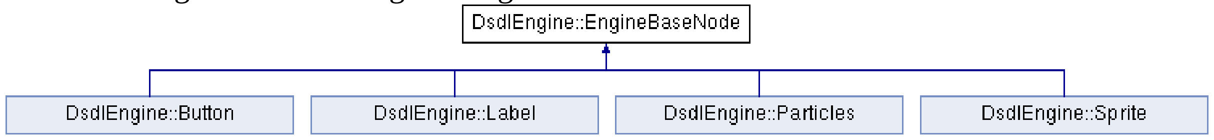
The documentation for this class was generated from the following files:

- Gui.h
- Gui.cpp

DsdlEngine::EngineBaseNode Class Reference

`#include <EngineBaseNode.h>`

Inheritance diagram for DsdlEngine::EngineBaseNode:



Public Member Functions

- EngineBaseNode ()
- virtual ~EngineBaseNode ()
- virtual void destroy ()
- virtual void cleanup ()
- bool load (SDL_Renderer *r)
- void render (SDL_Renderer *r)
- void renderCollisionShape (SDL_Renderer *r, CollisionShape *shape)
- void setPosition (const Vec2 &pos)
- void setPositionX (int x)
- void setPositionY (int y)
- const Vec2 getPosition () const
- void setSize (Size si)
- void setWidth (int w)
- void setHeight (int h)
- const Vec2 getContentSize () const
- void scaleNode (float scale)
- void scaleWidth (float scale)
- void scaleHeight (float scale)
- void setAssetPath (std::string path)
- std::string getAssetsPath ()
- NodeType getNodeType ()
- void setEngineNodeType (NodeType type)
- void setOpacity (int opacity)
- ResourceTexture * getEngineTexture ()
- void updateLabelText (std::string text)
- SDL_Rect * getBoundingBox ()
- void setBoundingBox (Vec2 pos, Vec2 size)
- void setUpdateTextureTrue (bool value)
- bool isTextureChanged ()

Protected Attributes

- std::string m_assetPath
- NodeType nodeType = NodeType::BASENODE
- ResourceTexture * m_engineTexture
- SDL_Rect * m_objectBoundingBox
- Vec2 m_position
- Vec2 m_size
- int m_numFrames
- int m_frame
- int m_opacity

- bool updateTextureInfo
- SDL_Rect m_gSpriteClips [14]
- SDL_Rect * m_currentFrame
- TTF_Font * m_font
- std::map< std::string, TTF_Font * > m_FontMap
- std::string m_labelText
- int m_textSize
- SDL_Color m_textColor
- CollisionShape * m_CollisionShape

Detailed Description

EngineBaseNode is the root for all elements in the framework

Definition at line 18 of file EngineBaseNode.h.

Constructor & Destructor Documentation

DsdlEngine::EngineBaseNode::EngineBaseNode ()

Constructor

Definition at line 10 of file EngineBaseNode.cpp.

DsdlEngine::EngineBaseNode::~EngineBaseNode () `[virtual]`

Deconstructor

Definition at line 23 of file EngineBaseNode.cpp.

Member Function Documentation

void DsdlEngine::EngineBaseNode::cleanup () `[virtual]`

virtual cleanup, for cleaning up node if it has to be reoved.

Reimplemented in DsdlEngine::Label.

Definition at line 165 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::destroy () `[virtual]`

virtual destroy, destorys node when it leaves scope.

Reimplemented in DsdlEngine::Sprite, DsdlEngine::Label, and DsdlEngine::Button.

Definition at line 154 of file EngineBaseNode.cpp.

std::string DsdlEngine::EngineBaseNode::getAssetsPath () `[inline]`

getAssetsPath, get the path of the asset to be loaded.

Returns:

std::string path to asset.

Definition at line 144 of file EngineBaseNode.h.

SDL_Rect* DsdlEngine::EngineBaseNode::getBoundingBox () `[inline]`

getBoundingBox, get the nodes SDL bounding box.

Returns:

SDL_Rect pointer.

Definition at line 180 of file EngineBaseNode.h.

const Vec2 DsdlEngine::EngineBaseNode::getContentSize () const `[inline]`

getContentSize, get size of the node.

Returns:

Vec2 size of the node

Definition at line 107 of file EngineBaseNode.h.

ResourceTexture* DsdlEngine::EngineBaseNode::getEngineTexture () `[inline]`

getEngineTexture, get the ResourceTexture for the node.

Returns:

ResourceTexture pointer.

Definition at line 168 of file EngineBaseNode.h.

NodeType DsdlEngine::EngineBaseNode::getNodeType () `[inline]`

getNodeType, get the type of a specific node.

Returns:

NodeType, the type of node.

Definition at line 150 of file EngineBaseNode.h.

const Vec2 DsdlEngine::EngineBaseNode::getPosition () const `[inline]`

getPosition, get position of the node.

Returns:

const Vec2.

Definition at line 83 of file EngineBaseNode.h.

bool DsdlEngine::EngineBaseNode::isTextureChanged () `[inline]`

isTextureChange, check if the texture was changed.

Returns:

bool.

Definition at line 200 of file EngineBaseNode.h.

bool DsdlEngine::EngineBaseNode::load (SDL_Renderer * *r*)

load, load node as SDL_Texture.

Parameters:

r	as SDL_Renderer argument.

Returns:

bool.

Definition at line 61 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::render (SDL_Renderer * *r*)

render, Render node to window.

Parameters:

r	as SDL_Renderer argument

Definition at line 30 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::renderCollisionShape (SDL_Renderer * *r*, CollisionShape * *shape*)

rendereCollisionShape, Render collision shape for node.

Parameters:

r	as a SDL_Rendere argumnet.
shape	as a CollisionShape pointer argument.

Definition at line 170 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::scaleHeight (float *scale*)`[inline]`

scaleHeight, Scale the height of the node.

Parameters:

scale	as a float argument

Definition at line 131 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::scaleNode (float *scale*)`[inline]`

scaleNode, Scale the node size by value passed in.

Parameters:

scale	as a float argument

Definition at line 119 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::scaleWidth (float *scale*)`[inline]`

scaleWidth, Scale the width of the node. scale as a float argument

Definition at line 125 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setAssetPath (std::string *path*)`[inline]`

setAssetPath, set the path to the asset to be loaded.

Parameters:

path	as a std::string argument.

Definition at line 138 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setBoundingBox (Vec2 *pos*, Vec2 *size*)

setBoundingBox, the the bounding box for the node.

Parameters:

pos	as a Vec2 argument.
size	as a Vec2 argument.

Definition at line 131 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::setEngineNodeType (NodeType *type*) `[inline]`

setEngineNodeType, set the type for a specific node.

Parameters:

type	as a NodeType argument.

Definition at line 156 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setHeight (int *h*) `[inline]`

setHeight, Set height of the node.

Parameters:

h	as a const int argument

Definition at line 101 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setOpacity (int *opacity*)

setOpacity, set the opacity value of a node, defaults to 255 if out of bounds value passed in.

Parameters:

opacity	as an int argument between 0 - 255

Definition at line 144 of file EngineBaseNode.cpp.

void DsdlEngine::EngineBaseNode::setPosition (const Vec2 & *pos*) `[inline]`

setPosition, Set position of the node.

Parameters:

pos	as a const Vec2 argument

Definition at line 65 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setPositionX (int *x*) `[inline]`

setPositionX, Set X position of the node.

Parameters:

x	as a const int argument

Definition at line 71 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setPositionY (int *y*) `[inline]`

setPositionY, Set Y position of the node.

Parameters:

Y	as a const int argument

Definition at line 77 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setSize (Size *si*) `[inline]`

setSize, Set size of the node.

Parameters:

si	as a const Size argument

Definition at line 89 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setUpdateTextureTrue (bool *value*) `[inline]`

setUpdateTextureTrue, control if node texture was changed after initial load.

Parameters:

value	as a bool argument.

Definition at line 194 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::setWidth (int *w*) `[inline]`

setWidth, Set width of the node.

Parameters:

w	as a const int argument

Definition at line 95 of file EngineBaseNode.h.

void DsdlEngine::EngineBaseNode::updateLabelText (std::string *text*)

updateLabelText, change the display text of a label.

Parameters:

text	as a std::string argument.

Definition at line 139 of file EngineBaseNode.cpp.

Member Data Documentation

std::string DsdlEngine::EngineBaseNode::m_assetPath `[protected]`

std::string asset path

Definition at line 205 of file EngineBaseNode.h.

CollisionShape* DsdlEngine::EngineBaseNode::m_CollisionShape `[protected]`

Box2D collision shape of the node

Definition at line 233 of file EngineBaseNode.h.

SDL_Rect* DsdlEngine::EngineBaseNode::m_currentFrame `[protected]`

the current frame rect

Definition at line 220 of file EngineBaseNode.h.

ResourceTexture* DsdlEngine::EngineBaseNode::m_engineTexture `[protected]`

ResourceTexture for the node

Definition at line 208 of file EngineBaseNode.h.

TTF_Font* DsdlEngine::EngineBaseNode::m_font `[protected]`

the font to use for labels

Definition at line 224 of file EngineBaseNode.h.

std::map<std::string, TTF_Font*> DsdlEngine::EngineBaseNode::m_FontMap `[protected]`

std::map for caching the font

Definition at line 225 of file EngineBaseNode.h.

int DsdlEngine::EngineBaseNode::m_frame [protected]

Definition at line 215 of file EngineBaseNode.h.

SDL_Rect DsdlEngine::EngineBaseNode::m_gSpriteClips[14] [protected]

frames rect for animation

Definition at line 219 of file EngineBaseNode.h.

std::string DsdlEngine::EngineBaseNode::m_labelText [protected]

std::string to hold label display text

Definition at line 227 of file EngineBaseNode.h.

int DsdlEngine::EngineBaseNode::m_numFrames [protected]

Definition at line 215 of file EngineBaseNode.h.

SDL_Rect* DsdlEngine::EngineBaseNode::m_objectBoundingBox [protected]

SDL_Rect bounding box for the node

Definition at line 209 of file EngineBaseNode.h.

int DsdlEngine::EngineBaseNode::m_opacity [protected]

int values for frames & opacity

Definition at line 215 of file EngineBaseNode.h.

Vec2 DsdlEngine::EngineBaseNode::m_position [protected]

Vec2 Position of the node

Definition at line 212 of file EngineBaseNode.h.

Vec2 DsdlEngine::EngineBaseNode::m_size [protected]

Vec2 Size of the node

Definition at line 213 of file EngineBaseNode.h.

SDL_Color DsdlEngine::EngineBaseNode::m_textColor [protected]

color of the label

Definition at line 229 of file EngineBaseNode.h.

int DsdlEngine::EngineBaseNode::m_textSize [protected]

int for size of font

Definition at line 228 of file EngineBaseNode.h.

NodeType DsdlEngine::EngineBaseNode::nodeType = NodeType::BASENODE `[protected]`

NodeType for containing node type

Definition at line 206 of file EngineBaseNode.h.

bool DsdlEngine::EngineBaseNode::updateTextureInfo `[protected]`

bool for texture control

Definition at line 217 of file EngineBaseNode.h.

The documentation for this class was generated from the following files:

- EngineBaseNode.h
- EngineBaseNode.cpp

DsdlEngine::EngineMaster Class Reference

```
#include <EngineMaster.h>
```

Static Public Member Functions

- static EngineMaster * getInstance ()

Protected Member Functions

- EngineMaster ()
- virtual ~EngineMaster ()

Detailed Description

EngineMaster is a stactic singleton helper class

Definition at line 14 of file EngineMaster.h.

Constructor & Destructor Documentation

DsdlEngine::EngineMaster::EngineMaster () `[inline]`, `[protected]`

Constructor

Definition at line 27 of file EngineMaster.h.

virtual DsdlEngine::EngineMaster::~EngineMaster () `[inline]`, `[protected]`, `[virtual]`

Deconstructor

Definition at line 32 of file EngineMaster.h.

Member Function Documentation

EngineMaster * DsdlEngine::EngineMaster::getInstance () `[static]`

getInstace, create EngineMaster as a Static instance.   static instance of EngineMaster

Definition at line 7 of file EngineMaster.cpp.

The documentation for this class was generated from the following files:

- EngineMaster.h
- EngineMaster.cpp

DsdlEngine::FileIO Class Reference

`#include <FileIO.h>`

Public Member Functions

- std::string getSuitableFOpen (const std::string &filenameUtf8) const
- std::string getWritablePath ()
- void setAssetsPath (std::string assetsPath)
- std::string getFileToOpen ()
- void setFileToOpen (std::string file)
- bool loadDocument (const char *filepath, char **doc_contents)
- bool writeDocument (const char *filepath, const char **doc_contents)
- XMLElement * getXMLNodeForKey (const char *pKey, XMLElement **rootNode, XMLDocument **doc)
- void setValueForKey (const char *value, const char *key)
- bool createXMLFile ()

Static Public Member Functions

- static FileIO * getInstance ()

Protected Member Functions

- FileIO ()
- virtual ~FileIO ()

Private Attributes

- std::string m_path
- std::string m_fileName

Detailed Description

FileIO class handles open and cllosing of xml files in the framework. Handles XML parsing and Saveing

Definition at line 21 of file FileIO.h.

Constructor & Destructor Documentation

DsdlEngine::FileIO::FileIO () `[inline]`, `[protected]`

Constructor

Definition at line 104 of file FileIO.h.

virtual DsdlEngine::FileIO::~FileIO () `[inline]`, `[protected]`, `[virtual]`

Deconstructor

Definition at line 109 of file FileIO.h.

Member Function Documentation

bool DsdlEngine::FileIO::createXMLFile ()

createXMLFile, Create a new Xml file.

Returns:

bool 1 on success.

Definition at line 207 of file FileIO.cpp.

std::string DsdlEngine::FileIO::getFileToOpen () `[inline]`

getFileToOpen, get the name of teh file to open.

Returns:

std::string filename.

Definition at line 54 of file FileIO.h.

FileIO * DsdlEngine::FileIO::getInstance () `[static]`

getInstance, Creates FileIO as a static singleton

Returns:

instance of FileIO

Definition at line 13 of file FileIO.cpp.

std::string DsdlEngine::FileIO::getSuitableFOpen (const std::string & *filenameUtf8*) const

getSuitableFopen, the the filename of the path to open

Parameters:

std::string	file path.

Returns:

std::string file path to open.

Definition at line 22 of file FileIO.cpp.

std::string DsdlEngine::FileIO::getWritablePath ()

getWriteablePath, get the full path to the file.

Returns:

std::string.

Definition at line 37 of file FileIO.cpp.

XMLElement * DsdlEngine::FileIO::getXMLNodeForKey (const char * *pKey*, XMLElement ** *rootNode*, XMLDocument ** *doc*)

getXMLNodeForKey, parses the file contents in the memory buffer for a xml element that matches the key.

Parameters:

pKey	key to search for in the file.
rootNode	XML node to use for search.
doc	XML doc to hold the contents.

Returns:

XMLElement the element matching the key.

Definition at line 98 of file FileIO.cpp.

bool DsdlEngine::FileIO::loadDocument (const char * *filepath*, char ** *doc_contents*)

loadDocument, load the contents of a file into memory for parsing,

Parameters:

filepath	const char path to file,
doc_contents	buffer to hold the file contents.

Returns:

bool.

Definition at line 53 of file FileIO.cpp.

void DsdlEngine::FileIO::setAssetsPath (std::string *assetsPath*) `[inline]`

setAssetsPath, set the path to the file root (only applies to windows platform)

Parameters:

std::string	path for file.

Definition at line 47 of file FileIO.h.

void DsdlEngine::FileIO::setFileToOpen (std::string *file*) `[inline]`

setFileToOpen, set the name of the file to open.

Parameters:

std::string	file name.

Definition at line 60 of file FileIO.h.

void DsdlEngine::FileIO::setValueForKey (const char * *value*, const char * *key*)

setValueForKey, Set or update the value of an XML element that matches the key.

Parameters:

vlaue	the value to be set.
key	the key to look for.

Definition at line 156 of file FileIO.cpp.

bool DsdlEngine::FileIO::writeDocument (const char * *filepath*, const char ** *doc_contents*)

writeDocument. Write the file contents from memory buffer to file and save.

Parameters:

filepath	const char path to file,
doc_contents	buffer cotaining the file contents.

Returns:

bool.

Definition at line 79 of file FileIO.cpp.

Member Data Documentation

std::string DsdlEngine::FileIO::m_fileName `[private]`

name of xml file to load

Definition at line 114 of file FileIO.h.

std::string DsdlEngine::FileIO::m_path `[private]`

path to folder which contains file

Definition at line 109 of file FileIO.h.

The documentation for this class was generated from the following files:

- FileIO.h
- FileIO.cpp

DsdlEngine::FpsLimiter Class Reference

```
#include <Timing.h>
```

Public Member Functions

- FpsLimiter ()
- ~FpsLimiter ()
- void init (float maxFPS)
- void setMaxFPS (float maxFPS)
- void begin ()
- float end ()

Private Member Functions

- void calculateFPS ()

Private Attributes

- float m_fFps
- float m_fMaxFPS
- float m_fFrameTime
- unsigned int m_iStartTicks

Detailed Description

Timming file handles setting up of caluclating and controlling frame rate of the engine.

Definition at line 12 of file Timing.h.

Constructor & Destructor Documentation

DsdlEngine::FpsLimiter::FpsLimiter ()

Constructor

Definition at line 10 of file Timing.cpp.

DsdlEngine::FpsLimiter::~FpsLimiter ()

Deconstructor

Definition at line 13 of file Timing.cpp.

Member Function Documentation

void DsdlEngine::FpsLimiter::begin ()

Start the Frame Rate Timer

Definition at line 28 of file Timing.cpp.

void DsdlEngine::FpsLimiter::calculateFPS () `[private]`

Calculate the running fps and keep it under control

Definition at line 45 of file Timing.cpp.

float DsdlEngine::FpsLimiter::end ()

End the frame rate timer

Returns:

float, the current fps vlaue.

Definition at line 33 of file Timing.cpp.

void DsdlEngine::FpsLimiter::init (float *maxFPS*)

Initializes the FPS limiter.

Parameters:

maxFPS,the	max frame rate allowed.

Definition at line 18 of file Timing.cpp.

void DsdlEngine::FpsLimiter::setMaxFPS (float *maxFPS*)

Sets the desired max FPS

Parameters:

maxFPS,the	desired Frame Rate.

Definition at line 23 of file Timing.cpp.

Member Data Documentation

float DsdlEngine::FpsLimiter::m_fFps `[private]`

Definition at line 53 of file Timing.h.

float DsdlEngine::FpsLimiter::m_fFrameTime `[private]`

float values for claculations

Definition at line 53 of file Timing.h.

float DsdlEngine::FpsLimiter::m_fMaxFPS `[private]`

Definition at line 53 of file Timing.h.

unsigned int DsdlEngine::FpsLimiter::m_iStartTicks `[private]`

starting timestamp

Definition at line 54 of file Timing.h.

The documentation for this class was generated from the following files:

- Timing.h
- Timing.cpp

DsdlEngine::IMainGame Class Reference

```
#include <IMainGame.h>
```

Public Member Functions

- IMainGame ()
- virtual ~IMainGame ()
- void run ()
- void setupWindow (int w, int h, std::string windowName, std::string path, int flag)
- void setFps (float fps)
- virtual void onInit ()=0
- virtual void addScenes ()=0
- virtual void onExit ()=0
- void onSDLEvent (SDL_Event &evnt)
- void setPaused ()
- void setRunning ()
- bool checkPaused ()

Public Attributes

- InputManager m_InputManager

Protected Attributes

- std::unique_ptr< SceneManager > m_pSceneManager
- IScene * m_pCurrentRunning
- bool m_bIsRunning
- bool m_bIsPaused
- Window m_Window
- SDL_Renderer * m_pGameRenderer
- AudioManager m_audioManager

Private Member Functions

- const float getFps () const
- void mainLoop ()
- void update ()
- void draw ()
- bool init ()
- bool initSystems ()
- void exitGame ()

Private Attributes

- float m_fFps
- unsigned int windowFlag
- int m_windowWidth
- int m_windowHeight
- std::string windowtitle
- std::string mainAssetsPath

Detailed Description

IMainGame is the heart of the engine as it contians the mian game loop and ties all the engine together with the game. Users must inherit from this class to make their application entry point.

Definition at line 26 of file IMainGame.h.

Constructor & Destructor Documentation

DsdlEngine::IMainGame::IMainGame ()

Constructor

Definition at line 22 of file IMainGame.cpp.

DsdlEngine::IMainGame::~IMainGame () `[virtual]`

Deconstructor

Definition at line 26 of file IMainGame.cpp.

Member Function Documentation

virtual void DsdlEngine::IMainGame::addScenes () `[pure virtual]`

addScenes, pure virtual function for user custom logic. this is where the user can add their scenes to the game scene manager (m_pSceneManager) it is called at start of main loop.

bool DsdlEngine::IMainGame::checkPaused () `[inline]`

checkPaused, check if the game is currently paused.

Returns:

bool

Definition at line 99 of file IMainGame.h.

void DsdlEngine::IMainGame::draw () `[private]`

draw, the main draw function, called once every loop cycle. calls all nodes draw functions and display the node to window

Definition at line 229 of file IMainGame.cpp.

void DsdlEngine::IMainGame::exitGame () `[private]`

exitGame. Cleans up and exits the game.

Definition at line 246 of file IMainGame.cpp.

const float DsdlEngine::IMainGame::getFps () const `[inline]`, `[private]`

getFps, Get the running Frame Rate.

Returns:

float fps,

Definition at line 132 of file IMainGame.h.

bool DsdlEngine::IMainGame::init () `[private]`

init, Initilazie the engine subsystems.

Returns:

bool.

Definition at line 129 of file IMainGame.cpp.

bool DsdlEngine::IMainGame::initSystems () `[private]`

initSystems. Create the SDL window and Renderer.

Returns:

bool.

Definition at line 169 of file IMainGame.cpp.

void DsdlEngine::IMainGame::mainLoop () `[private]`

mainLoop The main game loop for the engine and game.

Definition at line 35 of file IMainGame.cpp.

virtual void DsdlEngine::IMainGame::onExit () `[pure virtual]`

onExit, pure virtual function for user custom logic. called when exiting the game, so user should implement any cleaup they want to do in here.

virtual void DsdlEngine::IMainGame::onInit () `[pure virtual]`

onInit, pure virtual function for user custom logic should be used to setup window and fps as it is called before window is created.

void DsdlEngine::IMainGame::onSDLEvent (SDL_Event & *evnt*)

onSDLEvent, the games main Event listner

Parameters:

envt	as an SDL_Event

Definition at line 66 of file IMainGame.cpp.

void DsdlEngine::IMainGame::run ()

run, called in main file, runs the main game loop.

Definition at line 58 of file IMainGame.cpp.

void DsdlEngine::IMainGame::setFps (float *fps*) `[inline]`

setFps, Set the desired frame rate for the game.

Parameters:

fps	as a float value

Definition at line 58 of file IMainGame.h.

void DsdlEngine::IMainGame::setPaused () `[inline]`

setPaused, Pauses the main game loop.

Definition at line 88 of file IMainGame.h.

void DsdlEngine::IMainGame::setRunning () `[inline]`

setRunning, Starts the game loop running if paused.

Definition at line 93 of file IMainGame.h.

void DsdlEngine::IMainGame::setupWindow (int *w*, int *h*, std::string *windowName*, std::string *path*, int *flag*)

setupWindow, sets up the window defaults for Windows Platform.

Parameters:

w	as int width of the window.
h	as int height of the window.
windowName	as a std::string name of the window.
path	as std::string path to the windows root assets folder.
flag	as int SDL window creation flag

Definition at line 110 of file IMainGame.cpp.

void DsdlEngine::IMainGame::update () `[private]`

update, the main update function, called once every loop cycle updates any node that needs updating

Definition at line 181 of file IMainGame.cpp.

Member Data Documentation

AudioManager DsdlEngine::IMainGame::m_audioManager `[protected]`

the main AudioManager

Definition at line 113 of file IMainGame.h.

bool DsdlEngine::IMainGame::m_bIsPaused `[protected]`

bool variables for control

Definition at line 108 of file IMainGame.h.

bool DsdlEngine::IMainGame::m_bIsRunning `[protected]`

Definition at line 108 of file IMainGame.h.

float DsdlEngine::IMainGame::m_fFps `[private]`

engines fps

Definition at line 118 of file IMainGame.h.

InputManager DsdlEngine::IMainGame::m_InputManager

Main games inputmanage object

Definition at line 101 of file IMainGame.h.

IScene* DsdlEngine::IMainGame::m_pCurrentRunning `[protected]`

current running scene

Definition at line 107 of file IMainGame.h.

SDL_Renderer* DsdlEngine::IMainGame::m_pGameRenderer `[protected]`

the engine renderer

Definition at line 111 of file IMainGame.h.

std::unique_ptr<SceneManager> DsdlEngine::IMainGame::m_pSceneManager `[protected]`

Main Scene Manager for the Engine

Definition at line 104 of file IMainGame.h.

Window DsdlEngine::IMainGame::m_Window `[protected]`

the main window variable

Definition at line 110 of file IMainGame.h.

int DsdlEngine::IMainGame::m_windowHeight `[private]`

window height variable

Definition at line 123 of file IMainGame.h.

int DsdlEngine::IMainGame::m_windowWidth `[private]`

window width variabel

Definition at line 122 of file IMainGame.h.

std::string DsdlEngine::IMainGame::mainAssetsPath `[private]`

asset path to windows assets folder

Definition at line 125 of file IMainGame.h.

unsigned int DsdlEngine::IMainGame::windowFlag `[private]`

windowFlag variable

Definition at line 121 of file IMainGame.h.

std::string DsdlEngine::IMainGame::windowtitle `[private]`

window title variable

Definition at line 124 of file IMainGame.h.

The documentation for this class was generated from the following files:

- IMainGame.h
- IMainGame.cpp

DsdlEngine::InputManager Class Reference

```
#include <InputManager.h>
```

Public Member Functions

- InputManager ()
- ~InputManager ()
- void update ()
- void pressKey (unsigned int keyID)
- void releaseKey (unsigned int keyID)
- void setMouseCoords (float x, float y)
- bool isKeyDown (unsigned int keyID)
- bool isKeyPressed (unsigned int keyID)
- bool isKeyReleased (unsigned int KeyID)
- bool isTouch (unsigned int keyID)
- bool isSwipe (SDL_Event &evnt)
- bool isSwipeUp ()
- bool isSwipeDown ()
- bool isSwipeLeft (float x, float y)
- bool isSwipeRight (float x, float y)

Private Member Functions

- bool wasKeyDown (unsigned int keyID)

Private Attributes

- std::unordered_map< unsigned int, bool > _keyMap
- std::unordered_map< unsigned int, bool > _previousKeyMap
- bool swipeup
- bool swipedown
- bool swipeleft
- bool swiperight
- bool fingerDown
- bool fingerUp

Detailed Description

InputManager Class handles all input in the game

Definition at line 15 of file InputManager.h.

Constructor & Destructor Documentation

DsdlEngine::InputManager::InputManager ()

Constructor

Definition at line 6 of file InputManager.cpp.

DsdlEngine::InputManager::~InputManager ()

Deconstructor

Definition at line 13 of file InputManager.cpp.

Member Function Documentation

bool DsdlEngine::InputManager::isKeyDown (unsigned int *keyID*)

isKeyDown, check if key is down.

Parameters:

keyID,ID	of key to be checked.

Returns:

bool.

Definition at line 39 of file InputManager.cpp.

bool DsdlEngine::InputManager::isKeyPressed (unsigned int *keyID*)

isKeyPressed, check if key was just pressed.

Parameters:

keyID,ID	of key to be checked.

Returns:

bool.

Definition at line 69 of file InputManager.cpp.

bool DsdlEngine::InputManager::isKeyReleased (unsigned int *KeyID*)

isKeyReleased, check if key was just realesed.

Parameters:

keyID,ID	of key to be checked.

Returns:

bool.

Definition at line 79 of file InputManager.cpp.

bool DsdlEngine::InputManager::isSwipe (SDL_Event & *evnt*)

isSwipe, check if it was a swipe event.

Parameters:

evnt,event	to be checked.

Returns:

bool.

Definition at line 100 of file InputManager.cpp.

bool DsdlEngine::InputManager::isSwipeDown ()

isSwipeDown, check if event was a swipe down.

Returns:

bool.

Definition at line 154 of file InputManager.cpp.

bool DsdlEngine::InputManager::isSwipeLeft (float *x*, float *y*)

isSwipeLeft, check if event was a swipe left.

Returns:

bool.

Definition at line 159 of file InputManager.cpp.

bool DsdlEngine::InputManager::isSwipeRight (float *x*, float *y*)

isSwipeRight, check if event was a swipe right.

Returns:

bool.

Definition at line 164 of file InputManager.cpp.

bool DsdlEngine::InputManager::isSwipeUp ()

isSwipeUP, check if event was a swipe up.

Returns:

bool.

Definition at line 149 of file InputManager.cpp.

bool DsdlEngine::InputManager::isTouch (unsigned int *keyID*)

isTouch, check if it was touch event.

Parameters:

keyID,ID	of key to be checked.

Returns:

bool.

Definition at line 90 of file InputManager.cpp.

void DsdlEngine::InputManager::pressKey (unsigned int *keyID*)

pressKey, add key pressed to the key map.

Parameters:

keyID,the	id of the key that was pressed.

Definition at line 25 of file InputManager.cpp.

void DsdlEngine::InputManager::releaseKey (unsigned int *keyID*)

releaseKey, remove key pressed from the key map and add to previous map.

Parameters:

keyID,the	id of the key that was pressed.

Definition at line 30 of file InputManager.cpp.

void DsdlEngine::InputManager::setMouseCoords (float *x*, float *y*)

setMouseCoords, set the coordinates for the mouse.

Parameters:

x,float	value for mouse x location.
y,float	value for mouse y location.

Definition at line 34 of file InputManager.cpp.

void DsdlEngine::InputManager::update ()

update, loops through the key map.

Definition at line 18 of file InputManager.cpp.

bool DsdlEngine::InputManager::wasKeyDown (unsigned int *keyID*) `[private]`

wasKeyDown, Check if key was down.

Parameters:

keyID,ID	of key to be checked.

Returns:

bool.

Definition at line 54 of file InputManager.cpp.

Member Data Documentation

std::unordered_map<unsigned int, bool> DsdlEngine::InputManager::_keyMap `[private]`

map to hold current keys

Definition at line 119 of file InputManager.h.

std::unordered_map<unsigned int, bool> DsdlEngine::InputManager::_previousKeyMap `[private]`

map to hold previous keys

Definition at line 120 of file InputManager.h.

bool DsdlEngine::InputManager::fingerDown `[private]`

Definition at line 123 of file InputManager.h.

bool DsdlEngine::InputManager::fingerUp `[private]`

bools to control touch checking

Definition at line 123 of file InputManager.h.

bool DsdlEngine::InputManager::swipedown `[private]`

Definition at line 122 of file InputManager.h.

bool DsdlEngine::InputManager::swipeleft `[private]`

Definition at line 122 of file InputManager.h.

bool DsdlEngine::InputManager::swiperight `[private]`

bools to control swipe checking

Definition at line 122 of file InputManager.h.

bool DsdlEngine::InputManager::swipeup `[private]`

Definition at line 122 of file InputManager.h.

The documentation for this class was generated from the following files:

- InputManager.h
- InputManager.cpp

DsdlEngine::IScene Class Reference

```
#include <IScene.h>
```

Public Member Functions

- IScene ()
- virtual ~IScene ()
- virtual int getNextSceneIndex () const =0
- virtual int getPreviousSceneIndex () const =0
- virtual void onEntryScene ()=0
- virtual void onExitScene ()=0
- virtual void updateScene ()=0
- virtual void destroyScene ()=0
- int getSceneIndex () const
- SceneState getSceneState () const
- void setSceneRunning ()
- void setParentGame (IMainGame *game)
- virtual void onInput ()
- void addLayerToScene (Layer *layer)
- void loadScene (SDL_Renderer *r)
- void drawScene (SDL_Renderer *r)

Public Attributes

- std::vector< Layer * > sceneLayers

Protected Attributes

- SceneState m_eCurrentState = SceneState::NONE
- IMainGame * m_game = nullptr
- int m_iSceneIndex = SCENE_INDEX_NO_SCENE
- InputManager m_inputManager

Friends

- class SceneManager
  Friend Classes.
- class InputManager

Detailed Description

IScene is a inteface class to inherith from when creating a scene in the game.

Definition at line 35 of file IScene.h.

Constructor & Destructor Documentation

DsdlEngine::IScene::IScene () `[inline]`

Constructor

Definition at line 40 of file IScene.h.

virtual DsdlEngine::IScene::~IScene () `[inline]`, `[virtual]`

Deconstructor

Definition at line 47 of file IScene.h.

Member Function Documentation

void DsdlEngine::IScene::addLayerToScene (Layer * *layer*) `[inline]`

Add a Layer to the current Scene.

Parameters:

layer,*Layer*	to add to the scene.

Definition at line 123 of file IScene.h.

virtual void DsdlEngine::IScene::destroyScene () `[pure virtual]`

Pure virtual function. Destroy and cleanup when scene leaves scope.

void DsdlEngine::IScene::drawScene (SDL_Renderer * *r*) `[inline]`

Draw the current scenes layers to the window.

Parameters:

r,*SDL_Renderer*	to use when rendering.

Definition at line 141 of file IScene.h.

virtual int DsdlEngine::IScene::getNextSceneIndex () const `[pure virtual]`

Pure virtual function returns next scene.

Returns:

const int.

virtual int DsdlEngine::IScene::getPreviousSceneIndex () const `[pure virtual]`

Pure virtual function returns previous scene.

Returns:

const int.

int DsdlEngine::IScene::getSceneIndex () const `[inline]`

Gets the current scene's index.

Returns:

int.

Definition at line 93 of file IScene.h.

SceneState DsdlEngine::IScene::getSceneState () const `[inline]`

Get the current scenes state.

Returns:

SceneState int.

Definition at line 99 of file IScene.h.

void DsdlEngine::IScene::loadScene (SDL_Renderer * *r*) [inline]

Load the scene and its layers.

Parameters:

r,SDL_Renderer	to use when loading

Definition at line 131 of file IScene.h.

virtual void DsdlEngine::IScene::onEntryScene () [pure virtual]

Pure virtual function. Called when scene is loaded into focus.

virtual void DsdlEngine::IScene::onExitScene () [pure virtual]

Pure virtual function. Called when scene leaves focus.

void DsdlEngine::IScene::onInput () [virtual]

Virtual function for scene specific input.

Definition at line 12 of file Scene.cpp.

void DsdlEngine::IScene::setParentGame (IMainGame * *game*) [inline]

Set the game that the scene belongs to. game. the IMainGame the scene belongs to.

Definition at line 110 of file IScene.h.

void DsdlEngine::IScene::setSceneRunning () [inline]

Set a scene running by setting the state.

Definition at line 104 of file IScene.h.

virtual void DsdlEngine::IScene::updateScene () [pure virtual]

Pure virtual function. Called when scene is in focus and updates all elemets in the scene.

Friends And Related Function Documentation

friend class InputManager [friend]

Definition at line 150 of file IScene.h.

friend class SceneManager [friend]

Friend Classes.

Definition at line 149 of file IScene.h.

Member Data Documentation

SceneState DsdlEngine::IScene::m_eCurrentState = SceneState::NONE `[protected]`

Scenes current state variabel

Definition at line 152 of file IScene.h.

IMainGame* DsdlEngine::IScene::m_game = nullptr `[protected]`

parent game.

Definition at line 154 of file IScene.h.

InputManager DsdlEngine::IScene::m_inputManager `[protected]`

scnenes input manager

Definition at line 156 of file IScene.h.

int DsdlEngine::IScene::m_iSceneIndex = SCENE_INDEX_NO_SCENE `[protected]`

scene index int

Definition at line 155 of file IScene.h.

std::vector<Layer*> DsdlEngine::IScene::sceneLayers

Vector to hold scenes game Layer

Definition at line 117 of file IScene.h.

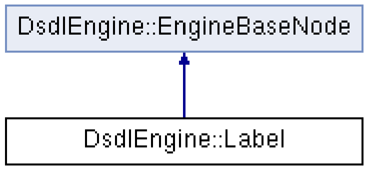
The documentation for this class was generated from the following files:

- IScene.h
- Scene.cpp

DsdlEngine::Label Class Reference

```
#include <Label.h>
```

Inheritance diagram for DsdlEngine::Label:



Public Member Functions

- Label ()
- virtual ~Label ()
- void create (Vec2 pos, std::string text, int fontSize, SDL_Color color, std::string fontFilePath)
- void setType (LableType type)
- const int getType ()
- void destroy ()
- void cleanup ()

Protected Attributes

- LableType m_labelType

Detailed Description

Label class is the base for all labels in the engine it inherits for EngineBaseNode.

Definition at line 12 of file Label.h.

Constructor & Destructor Documentation

DsdlEngine::Label::Label ()

Definition at line 11 of file Label.cpp.

DsdlEngine::Label::~Label () `[virtual]`

Deconstructor

Definition at line 16 of file Label.cpp.

Member Function Documentation

void DsdlEngine::Label::cleanup () `[virtual]`

Cleanup the lable texture.

Reimplemented from DsdlEngine::EngineBaseNode.

Definition at line 39 of file Label.cpp.

void DsdlEngine::Label::create (Vec2 *pos*, std::string *text*, int *fontSize*, SDL_Color *color*, std::string *fontFilePath*)

Create a basic Label pos, Vec2 Position of the label. text, std::String label display text, fontSize, int the font size to use color, SDL_Color of the label. fontFilePath, std::string path to the font to use.

Definition at line 21 of file Label.cpp.

void DsdlEngine::Label::destroy ()`[virtual]`

Destroy the label.

Reimplemented from DsdlEngine::EngineBaseNode.

Definition at line 34 of file Label.cpp.

const int DsdlEngine::Label::getType ()`[inline]`

Get the type of label it is.

Returns:

int label type,

Definition at line 44 of file Label.h.

void DsdlEngine::Label::setType (LableType *type*)`[inline]`

Set the type of label STATIC or DYNAMIC type, type of label.

Definition at line 38 of file Label.h.

Member Data Documentation

LableType DsdlEngine::Label::m_labelType`[protected]`

LabelType variable

Definition at line 57 of file Label.h.

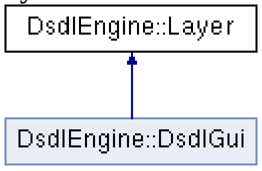
The documentation for this class was generated from the following files:

- Label.h
- Label.cpp

DsdlEngine::Layer Class Reference

`#include <Layer.h>`

Inheritance diagram for DsdlEngine::Layer:



Public Member Functions

- Layer ()
- virtual ~Layer ()
- virtual void destroy ()
- void loadNodes (SDL_Renderer *r)
- void drawNodes (SDL_Renderer *r)
- void addNodeToLayer (EngineBaseNode *node)
- void removeNodeFromLayer (EngineBaseNode *node)

Public Attributes

- std::vector< EngineBaseNode * > layerNodes

Friends

- class Gui

Detailed Description

Definition at line 15 of file Layer.h.

Constructor & Destructor Documentation

DsdlEngine::Layer::Layer ()

Constructor

Definition at line 13 of file Layer.cpp.

DsdlEngine::Layer::~Layer () `[virtual]`

Deconstructor

Definition at line 19 of file Layer.cpp.

Member Function Documentation

void DsdlEngine::Layer::addNodeToLayer (EngineBaseNode * *node*)

Add A node to the layer.

Parameters:

node,EngineBaseNode	to add to the Layer

Definition at line 38 of file Layer.cpp.

void DsdlEngine::Layer::destroy () [virtual]

Destroy the layer and all its contents

Reimplemented in DsdlEngine::DsdlGui.

Definition at line 24 of file Layer.cpp.

void DsdlEngine::Layer::drawNodes (SDL_Renderer * *r*)

Draw all nodes in the layer

Parameters:

r,SDL_Renderer	to be used when rendering

Definition at line 57 of file Layer.cpp.

void DsdlEngine::Layer::loadNodes (SDL_Renderer * *r*)

Load all nodes in the layer

Parameters:

r,SDL_Renderer	to be used when loading

Definition at line 50 of file Layer.cpp.

void DsdlEngine::Layer::removeNodeFromLayer (EngineBaseNode * *node*)

Remove a node from the layer.

Parameters:

node,EngineBaseNode	to be removed from the layer

Definition at line 44 of file Layer.cpp.

Friends And Related Function Documentation

friend class Gui [friend]

Definition at line 18 of file Layer.h.

Member Data Documentation

std::vector<EngineBaseNode*> DsdlEngine::Layer::layerNodes

vector to hold layer nodes

Definition at line 59 of file Layer.h.

The documentation for this class was generated from the following files:

- Layer.h
- Layer.cpp

DsdlEngine::Music Class Reference

`#include <AudioManager.h>`

Public Member Functions

- void play (int loop=-1)
- void audioPauseBG ()
- void audioResumeBG ()
- void audioStopBG ()

Private Attributes

- Mix_Music * m_Music

Friends

- class AudioManager

Detailed Description

Music class for interfacing with SDL Mix_Music.

Definition at line 39 of file AudioManager.h.

Member Function Documentation

void DsdlEngine::Music::audioPauseBG () `[inline]`

Pause Music currently playeing.

Definition at line 55 of file AudioManager.h.

void DsdlEngine::Music::audioResumeBG () `[inline]`

Resume Music that is currently paused.

Definition at line 60 of file AudioManager.h.

void DsdlEngine::Music::audioStopBG () `[inline]`

Stop Music that is currently playing.

Definition at line 65 of file AudioManager.h.

void DsdlEngine::Music::play (int *loop* = `-1`) `[inline]`

Play Music.

Parameters:

loops	== -1 : loop forever, 0 : loop once, 1+ : loop that many times

Definition at line 50 of file AudioManager.h.

Friends And Related Function Documentation

friend class AudioManager `[friend]`

Friend Class Audio Manager.

Definition at line 44 of file AudioManager.h.

Member Data Documentation

Mix_Music* DsdlEngine::Music::m_Music `[private]`

Private Mix_Music Variable

Definition at line 65 of file AudioManager.h.

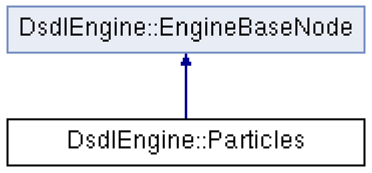
The documentation for this class was generated from the following file:

- AudioManager.h

DsdlEngine::Particles Class Reference

```
#include <Particles.h>
```

Inheritance diagram for DsdlEngine::Particles:



Public Member Functions

- Particles (int x, int y)
- ~Particles ()
- bool isDead (Particles *p)

Static Public Member Functions

- static float torad (float angle)

Private Attributes

- int life
- float mPosX
- float mPosY
- float xvel
- float yvel
- float angle
- float size
- Uint32 endtime

Additional Inherited Members

Detailed Description

Definition at line 10 of file Particles.h.

Constructor & Destructor Documentation

DsdlEngine::Particles::Particles (int *x*, int *y*)

Definition at line 6 of file Particles.cpp.

DsdlEngine::Particles::~Particles ()

Definition at line 16 of file Particles.cpp.

Member Function Documentation

bool DsdlEngine::Particles::isDead (Particles * *p*)

Definition at line 19 of file Particles.cpp.

static float DsdlEngine::Particles::torad (float *angle*) `[inline]`, `[static]`

Definition at line 18 of file Particles.h.

Member Data Documentation

float DsdlEngine::Particles::angle `[private]`

Definition at line 26 of file Particles.h.

Uint32 DsdlEngine::Particles::endtime `[private]`

Definition at line 27 of file Particles.h.

int DsdlEngine::Particles::life `[private]`

Definition at line 25 of file Particles.h.

float DsdlEngine::Particles::mPosX `[private]`

Definition at line 26 of file Particles.h.

float DsdlEngine::Particles::mPosY `[private]`

Definition at line 26 of file Particles.h.

float DsdlEngine::Particles::size `[private]`

Definition at line 26 of file Particles.h.

float DsdlEngine::Particles::xvel `[private]`

Definition at line 26 of file Particles.h.

float DsdlEngine::Particles::yvel `[private]`

Definition at line 26 of file Particles.h.

The documentation for this class was generated from the following files:

- Particles.h
- Particles.cpp

DsdlEngine::ResourceTexture Class Reference

```
#include <ResourceTexture.h>
```

Public Member Functions

- ResourceTexture ()
- ~ResourceTexture ()
- bool loadTexture (std::string texturePath, SDL_Renderer *r)
- bool loadTTF (std::string text, SDL_Color color, TTF_Font *myFont, SDL_Renderer *r)
- void render (Vec2 p, Vec2 s, SDL_Renderer *r, SDL_Rect *clip=NULL)
- void setBlendMode (SDL_BlendMode blending)
- void setAlpha (Uint8 alpha)
- void destroy ()

Private Attributes

- SDL_Texture * m_Texture
- std::map< std::string, SDL_Texture * > m_TextureMap
- int m_iWidth
- int m_iHeight

Detailed Description

ResourceTexture is responsible for loading and rendering all textures in the game. it is the base class for all textures.

Definition at line 15 of file ResourceTexture.h.

Constructor & Destructor Documentation

DsdlEngine::ResourceTexture::ResourceTexture ()

Constructor

Definition at line 24 of file ResourceTexture.cpp.

DsdlEngine::ResourceTexture::~ResourceTexture ()

Deconstructor

Definition at line 31 of file ResourceTexture.cpp.

Member Function Documentation

void DsdlEngine::ResourceTexture::destroy ()

Destroy the texture

Definition at line 129 of file ResourceTexture.cpp.

bool DsdlEngine::ResourceTexture::loadTexture (std::string *texturePath*, SDL_Renderer * *r*)

LoadTexture loads in sprite texture from the giving asset path.

Parameters:

texturePath,std::string	to the asset.
r,the	Renderer to use in loading

Returns:

bool

Definition at line 37 of file ResourceTexture.cpp.

bool DsdlEngine::ResourceTexture::loadTTF (std::string *text*, SDL_Color *color*, TTF_Font * *myFont*, SDL_Renderer * *r*)

LoadTTF loads in a texture created from a TTF font file.

Parameters:

text,text	to display onthe texture
color,the	SDL_COlor to use for the texture.
myFont,the	TTF_Font to use.
r,the	SDL_Renderer to use.

Returns:

bool.

Definition at line 80 of file ResourceTexture.cpp.

void DsdlEngine::ResourceTexture::render (Vec2 *p*, Vec2 *s*, SDL_Renderer * *r*, SDL_Rect * *clip* = `NULL`)

Render a texture to the window

Parameters:

p,Vec2	postion to render too.
s,Vec2	size of texture to render.
r,SDL_Renderer	to use.
clip,the	Sprite texture clip frame to use.

Definition at line 107 of file ResourceTexture.cpp.

void DsdlEngine::ResourceTexture::setAlpha (Uint8 *alpha*)

Set the Alpha for a texture

Parameters:

alpha,UNit8	value of Alpha to use.

Definition at line 150 of file ResourceTexture.cpp.

void DsdlEngine::ResourceTexture::setBlendMode (SDL_BlendMode *blending*)

Set the blend mode of the texture.

Parameters:

blending.	Blendmode to use.

Definition at line 141 of file ResourceTexture.cpp.

Member Data Documentation

int DsdlEngine::ResourceTexture::m_iHeight `[private]`

widht and height of the SDL_Texture

Definition at line 75 of file ResourceTexture.h.

int DsdlEngine::ResourceTexture::m_iWidth `[private]`

Definition at line 75 of file ResourceTexture.h.

SDL_Texture* DsdlEngine::ResourceTexture::m_Texture `[private]`

The SDL_Texture to use when loading and rendering

Definition at line 72 of file ResourceTexture.h.

std::map<std::string, SDL_Texture*> DsdlEngine::ResourceTexture::m_TextureMap `[private]`

std::Map to cache the textures

Definition at line 73 of file ResourceTexture.h.

The documentation for this class was generated from the following files:

- ResourceTexture.h
- ResourceTexture.cpp

DsdlEngine::SceneManager Class Reference

`#include <SceneManager.h>`

Public Member Functions

- SceneManager (IMainGame *game)
- ~SceneManager ()
- IScene * moveNext ()
- IScene * movePrevious ()
- void setScene (int nextScene)
- void addScene (IScene *newScene)
- void destroy ()
- IScene * getCurrentScene ()

Protected Attributes

- IMainGame * m_pGame
- std::vector< IScene * > m_pScenes
- int m_iCurrentSceneIndex

Detailed Description

Scene Manager for handling all in game scenes, holds vector of all scenes,

Definition at line 19 of file SceneManager.h.

Constructor & Destructor Documentation

DsdlEngine::SceneManager::SceneManager (IMainGame * *game*)

Constructor.

Parameters:

game,the	IMainGame the manager belongs to

Definition at line 10 of file SceneManager.cpp.

DsdlEngine::SceneManager::~SceneManager () `[inline]`

Deconstructor.

Definition at line 31 of file SceneManager.h.

Member Function Documentation

void DsdlEngine::SceneManager::addScene (IScene * *newScene*)

Add a Scene to the Scene Manager.

Parameters:

newScene,the	IScene to add to the Manager.

Definition at line 39 of file SceneManager.cpp.

void DsdlEngine::SceneManager::destroy ()

Destroy the SceneManager and all of its Scenes

Definition at line 46 of file SceneManager.cpp.

IScene * DsdlEngine::SceneManager::getCurrentScene ()

Get the Current Scene been managed

Returns:

IScene, the current scene.

Definition at line 56 of file SceneManager.cpp.

IScene * DsdlEngine::SceneManager::moveNext ()

Move to Next scene in vector

Returns:

IScene, the scene to move to.

Definition at line 16 of file SceneManager.cpp.

IScene * DsdlEngine::SceneManager::movePrevious ()

Move to Previous scene in vector

Returns:

IScene, the scene to move to.

Definition at line 25 of file SceneManager.cpp.

void DsdlEngine::SceneManager::setScene (int *nextScene*)

Sets the current Scene

Parameters:

nextScene,the	current scene.

Definition at line 34 of file SceneManager.cpp.

Member Data Documentation

int DsdlEngine::SceneManager::m_iCurrentSceneIndex `[protected]`

index for the current Scene

Definition at line 74 of file SceneManager.h.

IMainGame* DsdlEngine::SceneManager::m_pGame `[protected]`

Main Game which scenemanager belongs too

Definition at line 70 of file SceneManager.h.

std::vector<IScene*> DsdlEngine::SceneManager::m_pScenes `[protected]`

Vector to hold the game scenes

Definition at line 72 of file SceneManager.h.

The documentation for this class was generated from the following files:

- SceneManager.h
- SceneManager.cpp

DsdlEngine::SFX Class Reference

`#include <AudioManager.h>`

Public Member Functions

- void play (int loop=0)

Private Attributes

- Mix_Chunk * m_Chunk

Friends

- class AudioManager

Detailed Description

Sound effect class for interfacing with SDL Mix_Chunk.

Definition at line 16 of file AudioManager.h.

Member Function Documentation

void DsdlEngine::SFX::play (int *loop* = 0)

Paly sound effect.

Parameters:

int	loops == -1 : loop forever, 0 : loop once, 1+ : loop that many times

Definition at line 46 of file AudioManager.cpp.

Friends And Related Function Documentation

friend class AudioManager `[friend]`

Friend Class Audio Manager.

Definition at line 22 of file AudioManager.h.

Member Data Documentation

Mix_Chunk* DsdlEngine::SFX::m_Chunk `[private]`

Private Mix_Chunk Variable

Definition at line 33 of file AudioManager.h.

The documentation for this class was generated from the following files:

- AudioManager.h
- AudioManager.cpp

DsdlEngine::Size Class Reference

```
#include <EngineMath.h>
```

Public Member Functions

- Size ()
- Size (float w, float h)
- Size (const Size &s)
- ~Size ()

Public Attributes

- float w_
- float h_

Detailed Description

Size is a class for creating a 2 point size variable

Definition at line 55 of file EngineMath.h.

Constructor & Destructor Documentation

DsdlEngine::Size::Size ()

Constructor Defaults values to 0 , 0

Definition at line 36 of file EngineMath.cpp.

DsdlEngine::Size::Size (float *w*, float *h*)

Constructor Set values on creation.

Parameters:

w	as a float argument
h	as a float argument

Definition at line 38 of file EngineMath.cpp.

DsdlEngine::Size::Size (const Size & *s*)

Constructor. Create a Size object with another Size

Parameters:

s	as a Size argument

Definition at line 40 of file EngineMath.cpp.

DsdlEngine::Size::~Size ()

Deconstructor

Definition at line 45 of file EngineMath.cpp.

Member Data Documentation

float DsdlEngine::Size::h_

float value for height

Definition at line 87 of file EngineMath.h.

float DsdlEngine::Size::w_

float value for width

Definition at line 86 of file EngineMath.h.

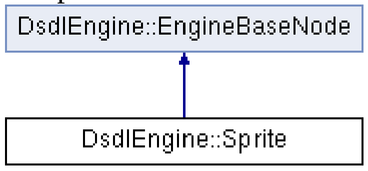
The documentation for this class was generated from the following files:

- EngineMath.h
- EngineMath.cpp

DsdlEngine::Sprite Class Reference

`#include <Sprite.h>`

Inheritance diagram for DsdlEngine::Sprite:



Public Member Functions

- Sprite ()
- virtual ~Sprite ()
- void create (Vec2 spriteSize, Vec2 position, std::string path)
- void create (Vec2 spriteSize, Vec2 position, std::string path, int numFrames)
- void createWithPhysics (b2World *world, Vec2 spriteSize, Vec2 position, std::string path, int numFrames, float den, float fri, bool FixedRotation)
- void updateTexure (Vec2 spriteSize, Vec2 position, std::string path, int numFrames)
- b2Body * getCollisionBody ()
- void destroy ()

Additional Inherited Members

Detailed Description

Sprite file for creating in game sprites. Inherits from EngineBaseNode

Definition at line 13 of file Sprite.h.

Constructor & Destructor Documentation

DsdlEngine::Sprite::Sprite ()

Constructor

Definition at line 11 of file Sprite.cpp.

DsdlEngine::Sprite::~Sprite () `[virtual]`

Deconstructor

Definition at line 21 of file Sprite.cpp.

Member Function Documentation

void DsdlEngine::Sprite::create (Vec2 *spriteSize*, Vec2 *position*, std::string *path*)

Create basic sprite with one frame.

Parameters:

spriteSize,Vec2	Size of the sprite.
position,Vec2	position of sprite.
path,std::string	path to sprite asset.

Definition at line 27 of file Sprite.cpp.

void DsdlEngine::Sprite::create (Vec2 *spriteSize*, Vec2 *position*, std::string *path*, int *numFrames*)

Create basic sprite with more then one frame.

Parameters:

spriteSize,Vec2	Size of the sprite.
position,Vec2	position of sprite.
path,std::string	path to sprite asset.
numFrames,int	number of frames.

Definition at line 43 of file Sprite.cpp.

void DsdlEngine::Sprite::createWithPhysics (b2World * *world*, Vec2 *spriteSize*, Vec2 *position*, std::string *path*, int *numFrames*, float *den*, float *fri*, bool *FixedRotation*)

Create basic sprite with one frame.

Parameters:

world,b2World	to add the sprite body to.
spriteSize,Vec2	Size of the sprite.
position,Vec2	position of sprite.
path,std::string	path to sprite asset.
numFrames,int	number of frames.
den,float	value for body dentisty.
fri,float	value for body friction.
fixedRotation,bool	value.

Definition at line 59 of file Sprite.cpp.

void DsdlEngine::Sprite::destroy () `[virtual]`

Destroy the sprite.

Reimplemented from DsdlEngine::EngineBaseNode.

Definition at line 79 of file Sprite.cpp.

b2Body* DsdlEngine::Sprite::getCollisionBody () `[inline]`

Get the collision body of the sprite.

Returns:

b2Body, the sprites collision body

Definition at line 68 of file Sprite.h.

void DsdlEngine::Sprite::updateTexure (Vec2 *spriteSize*, Vec2 *position*, std::string *path*, int *numFrames*)

Change the sprite texture of a sprite that is all ready loaded

Parameters:

spriteSize,Vec2	Size of the sprite.

position,Vec2	position of sprite.
path,std::string	path to sprite asset.
numFrames,int	number of frames.

Definition at line 84 of file Sprite.cpp.

The documentation for this class was generated from the following files:

- Sprite.h
- Sprite.cpp

DsdlEngine::Vec2 Class Reference

```
#include <EngineMath.h>
```

Public Member Functions

- Vec2 ()
- Vec2 (float x, float y)
- Vec2 (const Vec2 &v)
- ~Vec2 ()

Public Attributes

- float x_
- float y_

Static Public Attributes

- static const Vec2 ZERO

Detailed Description

Vec2 is a class for creating a 2 point position variable

Definition at line 11 of file EngineMath.h.

Constructor & Destructor Documentation

DsdlEngine::Vec2::Vec2 ()

Constructor Defaults values to 0 , 0

Definition at line 11 of file EngineMath.cpp.

DsdlEngine::Vec2::Vec2 (float *x*, float *y*)

Constructor Set values on creation.

Parameters:

x	as a float argument
y	as a float argument

Definition at line 15 of file EngineMath.cpp.

DsdlEngine::Vec2::Vec2 (const Vec2 & *v*)

Constructor. Create a Vec2 object with another Vec2

Parameters:

v	as a Vec2 argument

Definition at line 18 of file EngineMath.cpp.

DsdlEngine::Vec2::~Vec2 ()

Deconstructor

Definition at line 24 of file EngineMath.cpp.

Member Data Documentation

float DsdlEngine::Vec2::x_

float value for x position

Definition at line 46 of file EngineMath.h.

float DsdlEngine::Vec2::y_

float value for y position

Definition at line 47 of file EngineMath.h.

const Vec2 DsdlEngine::Vec2::ZERO `[static]`

const postition set to origin

Definition at line 44 of file EngineMath.h.

The documentation for this class was generated from the following files:

- EngineMath.h
- EngineMath.cpp

DsdlEngine::Window Class Reference

`#include <Window.h>`

Public Member Functions

- Window ()
- virtual ~Window ()
- int createWindow (std::string windowNname, int screenWidth, int screenHeight, unsigned int flag)
- void swapBuffer ()
- int getScreenWidth ()
- int getScreenHeight ()
- SDL_Renderer * getRenderer ()
- void destroy ()

Private Attributes

- SDL_Window * m_pSdlWindow
- SDL_Renderer * m_pSdlRenderer
- SDL_Surface * m_pScreenSurface
- int m_screenHeight
- int m_screenWidth

Detailed Description

Window class is the engines link to the SDL_Window. The window class is responsible for creating the window and renderer

Definition at line 16 of file Window.h.

Constructor & Destructor Documentation

DsdlEngine::Window::Window ()

Constructor

Definition at line 10 of file Window.cpp.

DsdlEngine::Window::~Window () `[virtual]`

Deconstructor

Definition at line 14 of file Window.cpp.

Member Function Documentation

int DsdlEngine::Window::createWindow (std::string *windowNname*, int *screenWidth*, int *screenHeight*, unsigned int *flag*)

Create the SDL_Window with the arguments passed in.

Parameters:

windowName,the	name of the window.
screenWidth,the	width of the window.
screenHeight,the	height of the window.
flag,SDL_Window	flag to use upon creation

Returns:

int, 0 upon success,

Definition at line 19 of file Window.cpp.

void DsdlEngine::Window::destroy ()

Destroy the window

Definition at line 74 of file Window.cpp.

SDL_Renderer* DsdlEngine::Window::getRenderer () `[inline]`

Get the SDL_Renderer for the window.

Returns:

SDL_Renderer.

Definition at line 60 of file Window.h.

int DsdlEngine::Window::getScreenHeight () `[inline]`

Get the height of the window.

Returns:

int, height of the window,

Definition at line 54 of file Window.h.

int DsdlEngine::Window::getScreenWidth () `[inline]`

Get the width of the window.

Returns:

int, width of the window.

Definition at line 48 of file Window.h.

void DsdlEngine::Window::swapBuffer ()

Swap the OpenGl window buffer.

Definition at line 69 of file Window.cpp.

Member Data Documentation

SDL_Surface* DsdlEngine::Window::m_pScreenSurface `[private]`

the SDL_Surface variable

Definition at line 70 of file Window.h.

SDL_Renderer* DsdlEngine::Window::m_pSdlRenderer `[private]`

the SDL_Renderer variable

Definition at line 69 of file Window.h.

SDL_Window* DsdlEngine::Window::m_pSdlWindow `[private]`

the SDL_Window variable

Definition at line 68 of file Window.h.

int DsdlEngine::Window::m_screenHeight `[private]`

Definition at line 72 of file Window.h.

int DsdlEngine::Window::m_screenWidth `[private]`

int variables for screenHeight and screenWidth

Definition at line 72 of file Window.h.

The documentation for this class was generated from the following files:

- Window.h
- Window.cpp

DsdlEngine::XmlLocalStorage Class Reference

`#include <XmlLocalStorage.h>`

Public Member Functions

- void setIntegerForKey (int value, const char *key)
- void setBoolForKey (bool value, const char *key)
- void setDoubleForKey (double value, const char *key)
- void setFloatForKey (float value, const char *key)
- void setStringForKey (std::string value, const char *key)
- int getIntegerForKey (const char *key)
- bool getBoolForKey (const char *key)
- double getDoubleForKey (const char *key)
- float getFloatForKey (const char *key)
- std::string getStringForKey (const char *key)
- void deleteValueForKey (const char *key)

Static Public Member Functions

- static XmlLocalStorage * getInstance ()

Protected Member Functions

- XmlLocalStorage ()
- virtual ~XmlLocalStorage ()

Detailed Description

XmlLocalStorage is a class for reading and setting values in XML It is able to read and set all the base value types.

Definition at line 17 of file XmlLocalStorage.h.

Constructor & Destructor Documentation

DsdlEngine::XmlLocalStorage::XmlLocalStorage () `[inline]`, `[protected]`

Constructor

Definition at line 108 of file XmlLocalStorage.h.

virtual DsdlEngine::XmlLocalStorage::~XmlLocalStorage () `[inline]`, `[protected]`, `[virtual]`

Deconstructor

Definition at line 113 of file XmlLocalStorage.h.

Member Function Documentation

void DsdlEngine::XmlLocalStorage::deleteValueForKey (const char * *key*)

Delete value for key. key, key to delete value for

Definition at line 211 of file XmlLocalStorage.cpp.

bool DsdlEngine::XmlLocalStorage::getBoolForKey (const char * *key*)

Get bool value by key, if the key doesn't exist, will return false.

Parameters:

key	The key to get value.

Returns:

bool value by `key` .

Definition at line 55 of file XmlLocalStorage.cpp.

double DsdlEngine::XmlLocalStorage::getDoubleForKey (const char * *key*)

Get double value by key, if the key doesn't exist, will return 0.

Parameters:

key	The key to get value.

Returns:

double value by `key` .

Definition at line 84 of file XmlLocalStorage.cpp.

float DsdlEngine::XmlLocalStorage::getFloatForKey (const char * *key*)

Get float value by key, if the key doesn't exist, will return 0.

Parameters:

key	The key to get value.

Returns:

float value by `key` .

Definition at line 112 of file XmlLocalStorage.cpp.

XmlLocalStorage * DsdlEngine::XmlLocalStorage::getInstance () `[static]`

getInstance gets a static instance for the XmlLocalStorage class.

Returns:

XmlLocalStorage instance

Definition at line 17 of file XmlLocalStorage.cpp.

int DsdlEngine::XmlLocalStorage::getIntegerForKey (const char * *key*)

Get int value by key, if the key doesn't exist, will return 0.

Parameters:

key	The key to get value.

Returns:

int value by `key` .

Definition at line 27 of file XmlLocalStorage.cpp.

std::string DsdlEngine::XmlLocalStorage::getStringForKey (const char * *key*)

Get string value by key, if the key doesn't exist, will return null.

Parameters:

key	The key to get value.

Returns:

string value by `key` .

Definition at line 118 of file XmlLocalStorage.cpp.

void DsdlEngine::XmlLocalStorage::setBoolForKey (bool *value*, const char * *key*)

Set a bool value for a key in xml.

Parameters:

key	The key to set.
value,bool	value to be saved

Definition at line 166 of file XmlLocalStorage.cpp.

void DsdlEngine::XmlLocalStorage::setDoubleForKey (double *value*, const char * *key*)

Set a double value for a key in xml.

Parameters:

key	The key to set.
value,double	value to be saved

Definition at line 176 of file XmlLocalStorage.cpp.

void DsdlEngine::XmlLocalStorage::setFloatForKey (float *value*, const char * *key*)

Set a float value for a key in xml.

Parameters:

key	The key to set.
value,float	value to be saved

Definition at line 198 of file XmlLocalStorage.cpp.

void DsdlEngine::XmlLocalStorage::setIntegerForKey (int *value*, const char * *key*)

Set a int value for a key in xml.

Parameters:

key	The key to set.
value,int	value to be saved

Definition at line 144 of file XmlLocalStorage.cpp.

void DsdlEngine::XmlLocalStorage::setStringForKey (std::string *value*, const char * *key*)

Set a string value for a key in xml.

Parameters:

key	The key to set.
value,string	value to be saved

Definition at line 203 of file XmlLocalStorage.cpp.

The documentation for this class was generated from the following files:

- XmlLocalStorage.h
- XmlLocalStorage.cpp

File Documentation

AudioManager.cpp File Reference

```
#include "AudioManager.h"
#include "FileIO.h"
```

Namespaces

- DsdlEngine

AudioManager.cpp

```
#include "AudioManager.h"
#include "FileIO.h"

namespace DsdlEngine {

    //init audio manager
    void AudioManager::init() {
        //init audio
        if (Mix_Init(MIX_INIT_MP3 | MIX_INIT_OGG) == -1) {
            SDL_Log("Mix_Init error: %s " ,Mix_GetError());
        }
        //open audio defaults
        if (Mix_OpenAudio(MIX_DEFAULT_FREQUENCY, MIX_DEFAULT_FORMAT, 2, 2048) == -1) {
            SDL_Log("Mix_OpenAudio error: %s " ,Mix_GetError());
        }

        m_bisInitialized = true;
    }

    //Clean Up Aduio
    void AudioManager::destroy() {
        if (m_bisInitialized)
            m_bisInitialized = false;

        //Loop Through maps and free audio
        for (auto& it : m_sfxAudioMap) {
            Mix_FreeChunk(it.second);
        }

        for (auto& it : m_bgAudioMap) {
            Mix_FreeMusic(it.second);
        }
        //Clear maps
        m_bgAudioMap.clear();
        m_sfxAudioMap.clear();

        //Close and Quit Audio
        Mix_CloseAudio();
        Mix_Quit();
    }

    //Play sound effect
    void SFX::play(int loop) {
        if (Mix_PlayChannel(-1, m_Chunk, loop) == -1) {
            if (Mix_PlayChannel(0, m_Chunk, loop) == -1) {
            SDL_Log("Mix_PlayChannel error: %s", Mix_GetError());
            }
        }
    }

    //Load Sound effect file
    //@parma : string audioPath ( path to file)
    SFX AudioManager::loadSFX(std::string audioPath) {

        std::string temp;
        temp = FileIO::getInstance()->getWritablePath() + audioPath;


        //Load SFX music (Mix_Chunk)
        SFX sfx;
        Mix_Chunk* sfxChunk = nullptr;

        //Check if allready cached
        auto it = m_sfxAudioMap.find(temp);

        //Not cached so load and cahe it
        if (it == m_sfxAudioMap.end()) {
            if ((sfxChunk = Mix_LoadWAV(temp.c_str())) == NULL) {
                SDL_Log("Mix_LoadWAV: Failed to load Audio %s", Mix_GetError());
            }
            sfx.m_Chunk = sfxChunk;
            m_sfxAudioMap[temp] = sfxChunk;
        }
        //it is cached
        else {
            sfx.m_Chunk = it->second;
        }
        return sfx;
    }


    //Load Music file
    //@parma : string audioPath ( path to file)
    Music AudioManager::loadMusic(std::string audioPath) {

        std::string temp;
        temp = FileIO::getInstance()->getWritablePath() + audioPath;


        //Check if allready cached
        auto it = m_bgAudioMap.find(temp);

        Music music;
        Mix_Music* mix = nullptr;
        //Not cached so load and cahe it
        if (it == m_bgAudioMap.end()) {
            if ((mix = Mix_LoadMUS(temp.c_str())) == NULL) {
                SDL_Log("Mix_LoadMUS: Failed to load Audio %s", Mix_GetError());
            }
            music.m_Music = mix;
            m_bgAudioMap[temp] = mix;
        }
        //it is cached
        else {
            music.m_Music = it->second;
        }
        return music;
    }
}

```

AudioManager.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::SFX
- class DsdlEngine::Music
- class DsdlEngine::AudioManager

Namespaces

- DsdlEngine

AudioManager.h

```
#ifndef _AUDIOMANAGER_
#define _AUDIOMANAGER_

#include "EngineDefines.h"
/*
Author: Derek O Brien
File : AudioManager.h
Description: Engine Audio Manger handles loading, playing and stoping of audio files
*/
//Name Space Wrapper
namespace DsdlEngine {

    class SFX {
    public:

        friend class AudioManager;

        void play(int loop = 0);
    private:
        Mix_Chunk* m_Chunk;
    };

    class Music {
    public:
        friend class AudioManager;

        void play(int loop = -1) { Mix_PlayMusic(m_Music, loop); };

        void audioPauseBG() { Mix_PauseMusic(); };

        void audioResumeBG() { Mix_ResumeMusic(); };

        void audioStopBG() { Mix_HaltMusic(); };

    private:

        Mix_Music* m_Music;
    };


    class AudioManager {
    public:
        AudioManager() { init(); };

        ~AudioManager() { destroy(); };

        void init();

        void destroy();
        SFX loadSFX(std::string audioPath);

        Music loadMusic(std::string audioPath);

    private:
        std::map<std::string, Mix_Chunk*> m_sfxAudioMap;

        std::map<std::string, Mix_Music*> m_bgAudioMap;

        bool m_bisInitialized;
    };
}

#endif
```

Button.cpp File Reference

```
#include "Button.h"
#include "IScene.h"
```

Namespaces

- DsdlEngine

Button.cpp

```cpp
#include "Button.h"
#include "IScene.h"

namespace DsdlEngine {

    Button::Button() {
        setEngineNodeType(NodeType::LABEL);
        m_eCurrentState = ButtonState::NORMAL;
    }

    Button::~Button() { destroy(); }

    //create button as label
    void Button::createTextButton(Vec2 pos, Size btnsize, std::string buttonText, std::string fontPath, SDL_Color color, SDL_Color bgColor) {

        m_size.y_ = btnsize.h_;
        m_size.x_ = buttonText.length() * btnsize.h_;

        m_position.x_ = pos.x_;
        m_position.y_ = pos.y_;


        m_labelText = buttonText;
        m_textSize = btnsize.h_;
        m_textColor = color;
        setAssetPath(fontPath);

        m_label = new Label();
        m_label->create(pos, m_labelText, m_textSize, m_textColor, fontPath);
    }

    //create button as sprite
    void Button::createSpriteButton(Vec2 spriteSize, Vec2 position, std::string imagePath, std::string name) {
        m_size.x_ = spriteSize.x_;
        m_size.y_ = spriteSize.y_;

        setAssetPath(imagePath);

        m_buttonName = name;

        m_position.x_ = position.x_;
        m_position.y_ = position.y_;
        setPosition(position);

        setEngineNodeType(NodeType::SPRITE);

        m_numFrames = 1;
        m_spriteBtn = new Sprite();
        m_spriteBtn->create(m_size, m_position, imagePath, 1);
    }

    //Set State to Hovering
    void Button::onMouseEnters() {
        m_eCurrentState = ButtonState::HOVERING;
        SDL_Log("Mouse over button");
    }

    //Set State Back to Normal
    void Button::onMouseLeaves() {
        m_eCurrentState = ButtonState::NORMAL;
    }

    //Set State to Pressd, Preform Action
    void Button::onClicked() {
        m_eCurrentState = ButtonState::PRESSED;
        SDL_Log("Pressed button");
    }

    //Check for mouse input on a button
    void Button::checkInput(SDL_Event& e) {

        //check if mouse over button
        if (e.type == SDL_MOUSEBUTTONDOWN || e.type == SDL_MOUSEBUTTONUP) {
            int x, y;
            SDL_GetMouseState(&x, &y);

            //Check if mouse inside button area
            bool inside = true;

            //Check if mouse inside button
            if (x < m_position.x_) {
                inside = false;
            }
            else if (x > m_position.x_ + m_size.y_) {
                inside = false;
            }
            else if (y < m_position.y_) {
                inside = false;
            }
            else if (y > m_position.y_ + m_size.x_) {
                inside = false;
            }

            //If mouse outside buttton
            if (!inside) {
                onMouseLeaves();
            }
            else {//If mouse is inside button check mouse input type
                switch (e.type) {
                case SDL_MOUSEMOTION:
                    onMouseEnters();
                    break;
                case SDL_MOUSEBUTTONUP:
                    onMouseEnters();
                    break;
                case SDL_MOUSEBUTTONDOWN:
                    onClicked();
                    break;
                    //Touch down
                case SDL_FINGERDOWN:
                    onClicked();
                    break;
                case SDL_FINGERMOTION:
                    onMouseEnters();
                    break;
                case SDL_FINGERUP:
                    onMouseEnters();
                    break;
                default:
                    break;
                }
            }
        }

    }

    void Button::destroy() {

    }
}
```

Button.h File Reference

```
#include "EngineBaseNode.h"
#include "Label.h"
#include "Sprite.h"
```

Classes

- class DsdlEngine::Button

Namespaces

- DsdlEngine

Button.h

```
#ifndef _BUTTON_
#define _BUTTON_

#include "EngineBaseNode.h"

#include "Label.h"
#include "Sprite.h"

namespace DsdlEngine {

    class Button : public EngineBaseNode {

    public:
        Button();

        virtual ~Button();

        void destroy();

        void createTextButton(Vec2 pos, Size size, std::string buttonText, std::string fontPath, SDL_Color textColor, SDL_Color bgColor);

        void createSpriteButton(Vec2 spriteSize, Vec2 position, std::string imagePath, std::string name);

        void checkInput(SDL_Event& e);


        std::string getButtonName() { return m_buttonName; }

        ButtonState m_eCurrentState;
    private:

        void onMouseEnters();

        void onMouseLeaves();

        void onClicked();


        Label* m_label;

        Sprite* m_spriteBtn;

        std::string m_buttonName;

    };
}

```

```
#endif //!_BUTTON_
```

CollisionShane.cpp File Reference

`#include "CollisionShape.h"`

Namespaces

- DsdlEngine

CollisionShane.cpp

```
#include "CollisionShape.h"


namespace DsdlEngine {

    CollisionShape::CollisionShape(){
        //Empty

    }

    CollisionShape::~CollisionShape(){
        //Empty
    }


    void CollisionShape::init(b2World* world,
        Vec2 position,
        Vec2 dimensions,
        float density,
        float friction,
        bool fixedRotation) {

        m_dimensions = dimensions;

        // Make the body
        b2BodyDef bodyDef;
        bodyDef.type = b2_dynamicBody;
        bodyDef.position.Set(position.x_, position.y_);
        bodyDef.fixedRotation = fixedRotation;
        m_body = world->CreateBody(&bodyDef);


        // Create the box
        b2PolygonShape boxShape;
        boxShape.SetAsBox(dimensions.x_ / 2.0f, (dimensions.y_ - dimensions.x_) / 2.0f);

        b2FixtureDef fixtureDef;
        fixtureDef.shape = &boxShape;
        fixtureDef.density = density;
        fixtureDef.friction = friction;
        m_fixtures[0] = m_body->CreateFixture(&fixtureDef);
    }


    void CollisionShape::destroy(b2World* world) {
        if (m_body) {
            world->DestroyBody(m_body);
            m_body = nullptr;
        }
    }
}
```

CollisionShape.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::CollisionShape

Namespaces

- DsdlEngine

CollisionShape.h

```
#ifndef __COLLISIONSHAPE__
#define __COLLISIONSHAPE__

#include "EngineDefines.h"
namespace DsdlEngine {
    class CollisionShape
    {
    public:
        CollisionShape();

        ~CollisionShape();

        void init(b2World* world,
            Vec2 position,
            Vec2 dimensions,
            float density,
            float friction,
            bool fixedRotation);

        void destroy(b2World* world);

        b2Body* getBody() const { return m_body; }

        b2Fixture* getFixture(int index) const { return m_fixtures[index]; }

        const Vec2 getDimensions() const { return m_dimensions; }

    protected:
        b2Body* m_body = nullptr;

        b2Fixture* m_fixtures[1];

        Vec2 m_dimensions;
    };

}
#endif // !__COLLISIONSHAPE__
```

DsdlEngine.cpp File Reference

```
#include "EngineDefines.h"
#include "DsdlEngine.h"
```

Namespaces

- DsdlEngine

Functions

- int DsdlEngine::init ()

DsdlEngine.cpp

```

#include "EngineDefines.h"
#include "DsdlEngine.h"

namespace DsdlEngine{

    int init(){
        //Initialize SDL
        SDL_Init(SDL_INIT_AUDIO | SDL_INIT_EVENTS | SDL_INIT_TIMER | SDL_INIT_VIDEO);

        SDL_Log("Log Print SDL Finised Init!\n");

        //Initialize TTF
        TTF_Init();

        SDL_GL_SetAttribute(SDL_GL_ACCELERATED_VISUAL, 1);

        return 0;
    }
}
```

DsdlEngine.h File Reference

```
#include "AudioManager.h"
#include "Button.h"
#include "CollisionShape.h"
#include "EngineBaseNode.h"
#include "EngineDefines.h"
#include "EngineMaster.h"
#include "EngineMath.h"
#include "EngineError.h"
#include "FileIO.h"
#include "Gui.h"
#include "IMainGame.h"
#include "InputManager.h"
#include "IScene.h"
#include "Label.h"
#include "Layer.h"
#include "ResourceTexture.h"
#include "SceneManager.h"
#include "Sprite.h"
#include "Timing.h"
#include "XmlLocalStorage.h"
#include "Window.h"
```

Namespaces

- DsdlEngine

Functions

- int DsdlEngine::init ()

DsdlEngine.h

```
#ifndef _DSDLENGINE_

#include "AudioManager.h"
#include "Button.h"
#include "CollisionShape.h"
#include "EngineBaseNode.h"
#include "EngineDefines.h"
#include "EngineMaster.h"
#include "EngineMath.h"
#include "EngineError.h"
#include "FileIO.h"
#include "Gui.h"
#include "IMainGame.h"
#include "InputManager.h"
#include "IScene.h"
#include "Label.h"
#include "Layer.h"
#include "ResourceTexture.h"
#include "SceneManager.h"
#include "Sprite.h"

#include "Timing.h"
#include "XmlLocalStorage.h"

#include "Window.h"
```

```
namespace DsdlEngine {
    extern int init();
}

#endif // !_DSDLENGINE_

```

EngineBaseNode.cpp File Reference

```
#include "EngineBaseNode.h"
#include "FileIO.h"
```

Namespaces

- DsdlEngine

EngineBaseNode.cpp

```

#include "EngineBaseNode.h"
#include "FileIO.h"
namespace DsdlEngine {

    //Constructor
    EngineBaseNode::EngineBaseNode() {

        m_engineTexture = NULL;
        setEngineNodeType(NodeType::BASENODE);
        m_frame = 0;
        m_numFrames = 1;
        m_opacity = 255;
        m_objectBoundingBox = new SDL_Rect();

        updateTextureInfo = false;
    }

    //Deconstructor
    EngineBaseNode::~EngineBaseNode() {
        destroy();
    }



    //Render Node by type
    void EngineBaseNode::render(SDL_Renderer* r) {
        if (nodeType == NodeType::SPRITE) {
            m_currentFrame = &m_gSpriteClips[m_frame / m_numFrames];
            m_engineTexture->setAlpha(m_opacity);

            //Draw Bounding Box
            //SDL_SetRenderDrawColor(r, 0, 255, 255, 255);
            //SDL_RenderDrawRect(r, m_objectBoundingBox);

            //render texture
            m_engineTexture->render(m_position, m_size, r, m_currentFrame);
            ++m_frame;

            if (m_frame / m_numFrames >= m_numFrames) {
                m_frame = 0;
            }
        }
        else if (nodeType == NodeType::LABEL) {
            m_engineTexture->render(m_position, m_size, r);
        }
        else if (nodeType == NodeType::BUTTON) {
            m_engineTexture->render(m_position, m_size, r);
        }
        else if (nodeType == NodeType::PARTICLE) {

        }

    }


    //Load Node as engine texture
    bool EngineBaseNode::load(SDL_Renderer * r) {

        m_engineTexture = new ResourceTexture();
        m_objectBoundingBox = new SDL_Rect();

        if (nodeType == NodeType::SPRITE || nodeType == NodeType::BUTTON) {
            if (!m_engineTexture->loadTexture(m_assetPath, r))
                SDL_Log("Faild to load sprite");

            else {

                int temp = 0;

                for (int i = 0; i < m_numFrames; i++) {
                    m_gSpriteClips[i].x = temp;
                    m_gSpriteClips[i].y = 0;
                    m_gSpriteClips[i].w = m_size.x_;
                    m_gSpriteClips[i].h = m_size.y_;

                    temp += m_size.x_;
                }
                    m_objectBoundingBox->x = m_position.x_;
                    m_objectBoundingBox->y = m_position.y_;
                    m_objectBoundingBox->w = m_size.x_;
                    m_objectBoundingBox->h = m_size.y_;

            }
            return true;
        }
        else if (nodeType == NodeType::LABEL) {


            if (!TTF_WasInit()) {
                TTF_Init();
            }

            std::string temp;

            temp = FileIO::getInstance()->getWritablePath() + m_assetPath;

            //Using getWritablePath
            FileIO::getInstance()->getWritablePath() + m_assetPath;

            //Check if font in chache
            auto it = m_FontMap.find(temp);

            // if not load and create texture
            if (it == m_FontMap.end()) {

                //open font
                m_font = TTF_OpenFont(temp.c_str(), m_textSize);
                if (m_font == NULL) {
                    SDL_Log("TTF_OpenFont Error : %s",TTF_GetError());
                }

                m_engineTexture->loadTTF(m_labelText, m_textColor, m_font, r);

                m_FontMap[temp] = m_font;
            }
            else {//create texture
                m_font = it->second;
                m_engineTexture->loadTTF(m_labelText, m_textColor, m_font, r);
            }

            return true;
        }
        else
            return false;
    }

    void EngineBaseNode::setBoundingBox(Vec2 pos, Vec2 size) {
        m_objectBoundingBox = new SDL_Rect();
        m_objectBoundingBox->x = pos.x_;
        m_objectBoundingBox->y = pos.y_;
        m_objectBoundingBox->w = size.x_;
        m_objectBoundingBox->h = size.y_;
    }

    void EngineBaseNode::updateLabelText(std::string text) {
        m_labelText = text;
    }


    void EngineBaseNode::setOpacity(int op) {
        if (op > 255 || op < 0) {
            SDL_Log("Invalid opacity value passed in.");
            m_opacity = 255;
        }
        else {
            m_opacity = op;
        }
    }

    void EngineBaseNode::destroy() {
        m_engineTexture->destroy();
        m_objectBoundingBox = nullptr;
        m_currentFrame = nullptr;
        m_frame = 0;
        m_numFrames = 0;
        m_opacity = 0;
        m_CollisionShape = nullptr;
        m_size = Vec2(0, 0);
    }

    void EngineBaseNode::cleanup() {
        m_engineTexture->destroy();
    }


    void EngineBaseNode::renderCollisionShape(SDL_Renderer* r, CollisionShape* shape) {

        SDL_Rect collisionbox;
        collisionbox.x = shape->getBody()->GetPosition().x;
        collisionbox.y = shape->getBody()->GetPosition().y;
        collisionbox.w = shape->getDimensions().x_;
        collisionbox.h = shape->getDimensions().y_;

        SDL_SetRenderDrawColor(r, 0, 255, 0, 255);
        SDL_RenderDrawRect(r, &collisionbox);
    }

}
```

EngineBaseNode.h File Reference

```
#include "EngineDefines.h"
#include "ResourceTexture.h"
#include "CollisionShape.h"
```

Classes

- class DsdlEngine::EngineBaseNode

Namespaces

- DsdlEngine

EngineBaseNode.h

```
#ifndef _ENGINEBASENODE_
#define _ENGINEBASENODE_

#include "EngineDefines.h"
#include "ResourceTexture.h"
#include "CollisionShape.h"


namespace DsdlEngine {
    class EngineBaseNode {
    public:
        EngineBaseNode();

        virtual ~EngineBaseNode();

        virtual void destroy();

        virtual void cleanup();

        bool load(SDL_Renderer* r);

        void render(SDL_Renderer* r);


        void renderCollisionShape(SDL_Renderer* r, CollisionShape* shape);

        void setPosition(const Vec2& pos) { m_position.x_ = pos.x_, m_position.y_ = pos.y_; };

        void setPositionX(int x) { m_position.x_ = x; }

        void setPositionY(int y) { m_position.y_ = y; }

        const Vec2 getPosition() const { return m_position; }

        void setSize(Size si) { m_size.x_ = si.w_; m_size.y_ = si.h_; }

        void setWidth(int w) { m_size.x_ = w; };

        void setHeight(int h) { m_size.y_ = h; };

        const Vec2 getContentSize() const { return m_size; }

        //Set Anchor Point
        //TODO

        //Rotate Node
        //TODO

        void scaleNode(float scale) { m_size.x_ = m_size.x_ * scale; m_size.y_ = m_size.y_ * scale; }

        void scaleWidth(float scale) { m_size.x_ = m_size.x_ * scale; }

        void scaleHeight(float scale) { m_size.y_ = m_size.y_ * scale; }


        void setAssetPath(std::string path) { m_assetPath = path; }

        std::string getAssetsPath() { return m_assetPath; }

        NodeType getNodeType() { return nodeType; }

        void setEngineNodeType(NodeType type) { nodeType = type; }

        void setOpacity(int opacity);

        ResourceTexture* getEngineTexture() { return m_engineTexture; }

        void updateLabelText(std::string text);

        SDL_Rect* getBoundingBox() { return m_objectBoundingBox; }

        void setBoundingBox(Vec2 pos, Vec2 size);


        void setUpdateTextureTrue(bool value) { updateTextureInfo = value; }

        bool isTextureChanged() { return updateTextureInfo; }

    protected:

        //EngineBaseNode* m_node;                                  /**< EngineBaseNode node*/
        std::string m_assetPath;
        NodeType nodeType = NodeType::BASENODE;
        ResourceTexture* m_engineTexture;
        SDL_Rect* m_objectBoundingBox;
        //nodes position Vec2
        Vec2 m_position;
        Vec2 m_size;
        int m_numFrames, m_frame, m_opacity;
        bool updateTextureInfo;
        SDL_Rect m_gSpriteClips[14];
        SDL_Rect* m_currentFrame;
        // For labels
        TTF_Font* m_font;
        std::map<std::string, TTF_Font*> m_FontMap;
        std::string m_labelText;
        int m_textSize;
        SDL_Color m_textColor;
        //Node Collision shape
        CollisionShape* m_CollisionShape;
    };
}

#endif // !_ENGINEBASENODE_
```

EngineDefines.h File Reference

```
#include <SDL.h>
#include "../dependencies/SDL2/SDL_image/SDL_image.h"
#include "../dependencies/SDL2/SDL_ttf/SDL_ttf.h"
#include "../dependencies/SDL2/SDL_mixer/SDL_mixer.h"
#include <string>
#include <iostream>
#include <memory>
#include <vector>
#include <map>
#include <unordered_map>
#include <functional>
#include "EngineError.h"
#include "EngineMath.h"
#include "EngineMaster.h"
#include <Box2D\Box2D.h>
```

Namespaces

- DsdlEngine

Macros

- #define USING_NS_DSDL using namespace DsdlEngine
- #define NS_DSDL_START namespace DsdlEngine{
- #define NS_DSDL_END }
- #define DEFAULT_ROOT_NAME "DefaultRoot"
- #define XML_FILE "Default.xml"
- #define TOTAL_PARTICLES 30
- #define METRESTOPIXELS 30

  Box2D scaling defines.
- #define PIXELSTOMETRES 1/30.0f
- #define RADTODEG (-180/3.1415926536f)
- #define DEGTORAD -0.0174532925199432957f
- #define GRAVITYSCALE 9.0f

Typedefs

- typedef SDL_TimerID DsdlEngine::CallBackTimer
- typedef SDL_TimerCallback DsdlEngine::CallBack

Enumerations

- enum DsdlEngine::NodeType { DsdlEngine::NodeType::BASENODE, DsdlEngine::NodeType::SPRITE, DsdlEngine::NodeType::LABEL, DsdlEngine::NodeType::BUTTON, DsdlEngine::NodeType::PARTICLE }
- enum DsdlEngine::ButtonState { DsdlEngine::ButtonState::NORMAL, DsdlEngine::ButtonState::PRESSED, DsdlEngine::ButtonState::HOVERING }
- enum DsdlEngine::ButtonType { DsdlEngine::ButtonType::LABEL_BTN, DsdlEngine::ButtonType::SPRITE_BTN }
- enum DsdlEngine::LableType { DsdlEngine::LableType::LABEL_STATIC, DsdlEngine::LableType::LABEL_DYNAMIC }

Macro Definition Documentation

#define DEFAULT_ROOT_NAME   "DefaultRoot"

Definition at line 77 of file EngineDefines.h.

#define DEGTORAD   -0.0174532925199432957f

Definition at line 88 of file EngineDefines.h.

#define GRAVITYSCALE   9.0f

Definition at line 90 of file EngineDefines.h.

#define METRESTOPIXELS   30

Box2D scaling defines.

Definition at line 85 of file EngineDefines.h.

#define NS_DSDL_END   }

Definition at line 32 of file EngineDefines.h.

#define NS_DSDL_START   namespace DsdlEngine{

Definition at line 31 of file EngineDefines.h.

#define PIXELSTOMETRES   1/30.0f

Definition at line 86 of file EngineDefines.h.

#define RADTODEG   (-180/3.1415926536f)

Definition at line 87 of file EngineDefines.h.

#define TOTAL_PARTICLES   30

Definition at line 81 of file EngineDefines.h.

#define USING_NS_DSDL   using namespace DsdlEngine

Definition at line 30 of file EngineDefines.h.

#define XML_FILE   "Default.xml"

Definition at line 78 of file EngineDefines.h.

EngineDefines.h

```
 1
 2 #ifndef _ENGINEDEFINES_
 3 #define _ENGINEDEFINES_
 4
 5
 6 #include <SDL.h>
 7 #include "../dependencies/SDL2/SDL_image/SDL_image.h"
 8 #include "../dependencies/SDL2/SDL_ttf/SDL_ttf.h"
 9 #include "../dependencies/SDL2/SDL_mixer/SDL_mixer.h"
10
11 #include <string>
12 #include <iostream>
13 #include <memory>
14 #include <vector>
15 #include <map>
16 #include <unordered_map>
17
18 #include <functional> //for std::function (CALLBACK FUNCTION)
19
20
21 #include "EngineError.h"
22 #include "EngineMath.h"
23 #include "EngineMaster.h"
24
25
26 #include <Box2D\Box2D.h>
27
28
29 //Set Macro Defines for Namespace
30 #define USING_NS_DSDL   using namespace DsdlEngine
31 #define NS_DSDL_START   namespace DsdlEngine{
32 #define NS_DSDL_END     }
33
34 namespace DsdlEngine {
38     enum class NodeType {
39         BASENODE,
40         SPRITE,
41         LABEL,
42         BUTTON,
43         PARTICLE
44     };
45
49     enum class ButtonState {
50         NORMAL,
51         PRESSED,
52         HOVERING
53     };
54
58     enum class ButtonType {
59         LABEL_BTN,
60         SPRITE_BTN
61     };
62
66     enum class LableType {
67         LABEL_STATIC,
68         LABEL_DYNAMIC
69     };
70
74     typedef SDL_TimerID CallBackTimer;
75     typedef SDL_TimerCallback CallBack;

#define DEFAULT_ROOT_NAME "DefaultRoot"
#define XML_FILE "Default.xml"


#define TOTAL_PARTICLES 30


#define METRESTOPIXELS 30
#define PIXELSTOMETRES 1/30.0f
#define RADTODEG  (-180/3.1415926536f)
#define DEGTORAD -0.0174532925199432957f

#define GRAVITYSCALE 9.0f

}

#endif //!_ENGINEDEFINES_
```

EngineError.cpp File Reference

`#include "EngineError.h"`

Namespaces

- DsdlEngine

EngineError.cpp

```
#include "EngineError.h"

namespace DsdlEngine{


}
```

EngineError.h File Reference

```
#include "EngineDefines.h"
```

Namespaces

- DsdlEngine

Macros

- #define DEBUG_DSDL 1
- #define DEBUG_MSG(x) (std::cout << (x) <<std::endl)

Macro Definition Documentation

#define DEBUG_DSDL 1

Definition at line 9 of file EngineError.h.

#define DEBUG_MSG(x) (std::cout << (x) <<std::endl)

Definition at line 12 of file EngineError.h.

EngineError.h

```
 1
 2 #ifndef _ENGINEERROR_
 3 #define _ENGINEERROR_
 4
 5 #include "EngineDefines.h"
 6
 7 namespace DsdlEngine{
 8
 9     #define DEBUG_DSDL 1
10     #if defined DEBUG_DSDL
11         #if (DEBUG_DSDL == 1)
12             #define DEBUG_MSG(x) (std::cout << (x) <<std::endl)
13         #else
14             #define DEBUG_MSG(x)
15         #endif
16     #else
17     #define DEBUG_MSG(x)
18     #endif
19
20 }
21
22
23 #endif
```

EngineMaster.cpp File Reference

`#include "EngineMaster.h"`

Namespaces

- DsdlEngine

Variables

- static EngineMaster * DsdlEngine::Instance = nullptr

EngineMaster.cpp

```
#include "EngineMaster.h"

namespace DsdlEngine {

    //Create As Singleton
    static EngineMaster* Instance = nullptr;
    EngineMaster* EngineMaster::getInstance() {
        if (!Instance) {
            Instance = new (std::nothrow) EngineMaster();
        }
        return Instance;
    }

}
```

EngineMaster.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::EngineMaster

Namespaces

- DsdlEngine

EngineMaster.h

```
#ifndef _ENGINEMASTER_
#define _ENGINEMASTER_

#include "EngineDefines.h"

namespace DsdlEngine{
    class EngineMaster{
    public:

        static EngineMaster* getInstance();

    protected:
        EngineMaster(){};

        virtual ~EngineMaster(){};

    private:

    };
}


#endif // !_ENGINEMASTER_
```

EngineMath.cpp File Reference

```
#include "EngineMath.h"
#include "XmlLocalStorage.h"
```

Namespaces

- DsdlEngine

EngineMath.cpp

```
#include "EngineMath.h"
#include "XmlLocalStorage.h"
namespace DsdlEngine{


    //Defaults to position of (0 , 0)
    Vec2::Vec2() : x_(0), y_(0){
    }

    //Set position to values passed in (x , y)
    Vec2::Vec2(float x, float y) : x_(x), y_(y){
    }

    Vec2::Vec2(const Vec2& v){
        this->x_ = v.x_;
        this->y_ = v.y_;
    }


    Vec2::~Vec2(){}

    /*
        SDL Window Coordintes
        origin is top left corner
    */
    const Vec2 Vec2::ZERO(0, 0);
}

namespace DsdlEngine{


    Size::Size() : w_(0), h_(0){}

    Size::Size(float w, float h) : w_(w), h_(h){}

    Size::Size(const Size& s){
        this->h_ = s.h_;
        this->w_ = s.w_;
    }

    Size::~Size(){}

}
```

EngineMath.h File Reference

Classes

- class DsdlEngine::Vec2
- class DsdlEngine::Size

Namespaces

- DsdlEngine

EngineMath.h

```
#ifndef _ENGINEMATH_
#define _ENGINEMATH_

namespace DsdlEngine{
    class Vec2{

    public:

        Vec2();

        Vec2(float x, float y);

        Vec2(const Vec2& v);

        ~Vec2();

        static const Vec2 ZERO;

        float x_;
        float y_;
    };

    class Size{
    public:

        Size();

        Size(float w, float h);


        Size(const Size& s);

        ~Size();


        float w_;
        float h_;
    };


}
#endif
```

FileIO.cpp File Reference

```
#include "FileIO.h"
#include <fstream>
```

Namespaces

- DsdlEngine

Variables

- static FileIO * DsdlEngine::Instance = nullptr

FileIO.cpp

```
#include "FileIO.h"
#include <fstream>

namespace DsdlEngine{

    //Create As Singleton
    static FileIO* Instance = nullptr;

    FileIO* FileIO::getInstance(){
        if (!Instance){
            Instance = new (std::nothrow) FileIO();
        }
        return Instance;
    }



    std::string FileIO::getSuitableFOpen(const std::string& filenameUtf8) const{
        return filenameUtf8;
    }


    /*
        Get Path to file

        if defs here for different platfoms as windows needs to find assets in root folder which
i have created
        but android needs to find assets in the jni/assets folder. android is allready set up
to go look in this folder
        there for all that was needed was the name and in the windows platfom i add on path to
the assets folder so it just has to look for name of file

        this will need to be done to each asset type loding function eg. audio, fonts, images
    */

    std::string FileIO::getWritablePath(){
    //For Windows
#ifdef __WIN32__
        m_path;
#endif

        //For Android
#ifdef __ANDROID__
        m_path = "";
#endif
        return m_path;
    }



    //Loads complete file into memory
    bool FileIO::loadDocument(const char* filepath, char** doc_contents) {

        //Open file
        SDL_RWops *file = SDL_RWFromFile(filepath, "rb");
        if (file != nullptr) {

            //Get length of file
            size_t file_length = SDL_RWseek(file, 0, SEEK_END);
            (*doc_contents) = new char[file_length + 1];
            SDL_RWseek(file, 0, SEEK_SET);

            //Read File into buffer
            int n_blocks = SDL_RWread(file, (*doc_contents), 1, file_length);

            //Close file
            SDL_RWclose(file);

            //add null terminator to end of file
            (*doc_contents)[file_length] = '\0';
            return true;
        }
        return false;
    }


    //Write contents of buffer to file and save.
    bool FileIO::writeDocument(const char* filepath, const char** doc_contents) {

        SDL_RWops *file = SDL_RWFromFile(filepath, "w");
        if (file != nullptr) {
            //Length of data to write
            size_t len = SDL_strlen(*doc_contents);

            //Write the data
            SDL_RWwrite(file, *doc_contents, 1, len);

            //close file
            SDL_RWclose(file);

            return true;
        }
        return false;
    }

    //Parse Xml for Element for Key and return Element node if found
    XMLElement* FileIO::getXMLNodeForKey(const char*pKey, XMLElement** rootNode,
XMLDocument** doc) {

        XMLElement* curNode = nullptr;

        char* contents = NULL;

        std::string path = getWritablePath() + "Default.xml";

        //Check the key
        if (!pKey) {
            return nullptr;
        }

        //Load Xml document into contents char
        if (FileIO::getInstance()->loadDocument(path.c_str(), &contents) != true) {
            SDL_Log("can not read xml file using SDL_rwops");
        }

        //SDL_Log(contents);

        XMLDocument* xmlDoc = new XMLDocument;
        *doc = xmlDoc;

        if (xmlDoc->Parse(contents) == XML_SUCCESS) {
            //SDL_Log("Doc Parsed");
            // get root node
            *rootNode = xmlDoc->RootElement();

            if (nullptr == *rootNode) {
                SDL_Log("read root node error ");
            }

            // find the node
            curNode = (*rootNode)->FirstChildElement();
            while (curNode != nullptr)
            {
                const char* nodeName = curNode->Value();
                if (!strcmp(nodeName, pKey)) {
                    break;
                }
                curNode = curNode->NextSiblingElement();
            }
        }
        else {
            SDL_Log("Could not load doc: ");
        }

        delete[] contents;

        return curNode;
    }


    /*
        Set Value for key in xml file
        @parma key = name of node to be written to file
        @parma vale = vale of node to be saved
    */
    void FileIO::setValueForKey(const char* value, const char* key) {

        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;
        XMLPrinter printer;
        std::string path;


        if (!key || !value) {
            return;
        }

        path = getWritablePath() + "Default.xml";

        //Check if node exists allready
        node = getXMLNodeForKey(key, &rootNode, &doc);

        //if node allready exists change value
        if (node) {
            if (node->FirstChild()) {
                node->FirstChild()->SetValue(value);
            }
            else {
                XMLText* content = doc->NewText(value);
                node->LinkEndChild(content);
            }
        }//Create new node and set value
        else {
            if (rootNode) {
                XMLElement* temp = doc->NewElement(key);
                rootNode->LinkEndChild(temp);
                XMLText* content = doc->NewText(value);
                temp->LinkEndChild(content);
            }
        }

        // attach printer to the document you want to convert in to a std::string
        doc->Accept(&printer);

        // Create a std::string and copy your document data in to the string
        const char* buffer = printer.CStr();

        //Write back to file and save file
        if (FileIO::getInstance()->writeDocument(path.c_str(), &buffer)) {
            SDL_Log("Key : %s :: Value : %s :: saved", key, value);
        }
        delete doc;
    }

    //Create XML File
    bool FileIO::createXMLFile() {
        bool bRet = false;

        XMLPrinter printer;

        XMLDocument *doc = new XMLDocument();
        if (nullptr == doc) {
            return false;
        }

        XMLDeclaration *pDeclaration = doc->NewDeclaration(nullptr);
        if (nullptr == pDeclaration) {
            return false;
        }

        doc->LinkEndChild(pDeclaration);
        XMLElement *pRootEle = doc->NewElement(DEFAULT_ROOT_NAME);
        if (nullptr == pRootEle) {
            return false;
        }

        doc->LinkEndChild(pRootEle);

        std::string path;

        path = getWritablePath() + "Default.xml";


        bRet = XML_SUCCESS ==
doc->SaveFile(FileIO::getInstance()->getSuitableFOpen(path).c_str());

        if (doc) delete doc;

        return bRet;
    }

}
```

FileIO.h File Reference

```
#include "EngineDefines.h"
#include <sys/stat.h>
#include "../dependencies/tinyxml/tinyxml2.h"
```

Classes

- class DsdlEngine::FileIO

Namespaces

- DsdlEngine

FileIO.h

```
#ifndef _FILEIO_
#define _FILEIO_

#include "EngineDefines.h"
#include <sys/stat.h>
#include "../dependencies/tinyxml/tinyxml2.h"
namespace DsdlEngine{

    using namespace tinyxml2;
    using namespace std;

    class FileIO{

    public:
        static FileIO* getInstance();

        std::string getSuitableFOpen(const std::string& filenameUtf8) const;

        std::string getWritablePath();

        void setAssetsPath(  std::string assetsPath)  { m_path = assetsPath; }


        std::string getFileToOpen() { return m_fileName; }

        void setFileToOpen(std::string file) { m_fileName = file; }

        bool loadDocument(const char* filepath, char** doc_contents);

        bool writeDocument(const char* filepath, const char** doc_contents);

        XMLElement* getXMLNodeForKey(const char*pKey, XMLElement** rootNode, XMLDocument**
doc);

        void setValueForKey(const char* value, const char* key);

        bool createXMLFile();

    protected:
        FileIO(){};

        virtual ~FileIO(){};

    private:

        std::string m_path;
        std::string m_fileName;
    };
}
```

```

#endif // !_FILEIO_
```

Gui.cpp File Reference

```
#include "Gui.h"
#include "Button.h"
#include "Window.h"
#include "Label.h"
```

Namespaces

- DsdlEngine

Gui.cpp

```
#include "Gui.h"
#include "Button.h"
#include "Window.h"
#include "Label.h"

/*
    File: Gui
    Author: Derek O Brien
    Description: GUi Layer templtate for creating an a UI Layer. Inherits from layer
*/
namespace DsdlEngine{

    //Constructor
    DsdlGui::DsdlGui() {
        //Empty
    }

    //Deconstructor
    DsdlGui::~DsdlGui() {
        destroy();
    }

    //Add Button to GUI
    void DsdlGui::addButton(ButtonType type, std::string name, Vec2 pos, Vec2 size, std::string path, SDL_Color color, SDL_Color bgColor, const char* text) {

        m_btn = new Button();
        //Create as Text Button
        if (type == ButtonType::LABEL_BTN) {
            m_btn->createTextButton(pos, Size(size.x_, size.y_), text, path, color, bgColor);
            m_btn->setPosition(pos);
        }

        //Create as Sprite Button
        if (type == ButtonType::SPRITE_BTN) {
            m_btn->createSpriteButton(size , pos, path, name);
            m_btn->setEngineNodeType(NodeType::SPRITE);
            m_btn->setPosition(pos);
        }

        //Add button to gui elements array
        GUIElements.push_back(m_btn);
        layerNodes.push_back(m_btn);
    }

    //Add Label to Gui Layer
    void DsdlGui::addLabel(LableType type, Vec2 pos, std::string text, int fontSize, SDL_Color color, std::string fontFilePath){
        m_label = new Label();

        m_label->setType(type);
        m_label->create(pos, text, fontSize, color, fontFilePath);

        layerNodes.push_back(m_label);
    }


    //Add predefined label to the gui layer
    void DsdlGui::addPreDefineLabel(Label* label, LableType type) {
        label->setType(type);
        layerNodes.push_back(label);
    }

    //Set Gui Layer Position
    void DsdlGui::setGUIPos() {

    }

    //Event Manager for input on gui Buttons
    void DsdlGui::onSDLEvent(SDL_Event& e) {
        //Loop and Check each button for input
        for (size_t i = 0; i < GUIElements.size(); i++){
            GUIElements.at(i)->checkInput(e);
        }
    }

    //Destroy
    void DsdlGui::destroy() {

        if (layerNodes.size() > 0) {

            for (size_t i = 0; i < layerNodes.size(); i++) {
                layerNodes.erase(std::remove(layerNodes.begin(), layerNodes.end(), layerNodes[i]), layerNodes.end());

                layerNodes[i]->destroy();

            }
        //  layerNodes.clear();
        }
        //  GUIElements.clear();
    }
}


```

Gui.h File Reference

```
#include "EngineDefines.h"
#include "Layer.h"
#include "IScene.h"
```

Classes

- class DsdlEngine::DsdlGui

Namespaces

- DsdlEngine

Gui.h

```
#ifndef _GUI_
#define _GUI_

#include "EngineDefines.h"
#include "Layer.h"
#include "IScene.h"

namespace DsdlEngine{

    //Forward Decalare Classes
    class Button;
    class Label;

    class DsdlGui : public Layer{
    public:
        DsdlGui();

        virtual ~DsdlGui();

        void addButton(ButtonType type, std::string name, Vec2 pos, Vec2 size, std::string path,
SDL_Color color, SDL_Color bgColor, const char* text = NULL);

        void addLabel(LableType type, Vec2 pos, std::string text, int fontSize, SDL_Color color,
std::string fontFilePath);

        void addPreDefineLabel(Label* label, LableType type);

        void setGUIPos();

        void onSDLEvent(SDL_Event& e);

        void destroy();

        std::vector<Button*> GUIElements;
        Button* getButton() { return m_btn; }

    protected:
        //Variables
        Label* m_label;
        Button* m_btn;
    };
}

#endif
```

IMainGame.cpp File Reference

```
#include "IMainGame.h"
#include "SceneManager.h"
#include "IScene.h"
```

Namespaces

- DsdlEngine

Functions

- template<typename T , typename... Args> std::unique_ptr< T > DsdlEngine::make_unique (Args &&...args)

IMainGame.cpp

```
#include "IMainGame.h"


#include "SceneManager.h"
#include "IScene.h"


namespace DsdlEngine {

    /*
    *Added template version of make_unique as Ndk did not support it in its version of STL
    *Error was make_unique not part of std::
    *After research this was the easiest solution to solve error
    *Ndk-build now builds apk as of 26/01/2016
    */
    template<typename T, typename ...Args>
    std::unique_ptr<T> make_unique(Args&& ...args) {
        return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
    }

    //Constructor
    IMainGame::IMainGame() {
        m_pSceneManager = DsdlEngine::make_unique<SceneManager>(this);
    }

    IMainGame::~IMainGame() {
        //Empty
    }


    /*
        Main Game Loop

    */
    void IMainGame::mainLoop() {
        if (!init()) return;
        FpsLimiter fpsLimit;
        fpsLimit.setMaxFPS(m_fFps);

        setRunning();
        while (m_bIsRunning) {
            fpsLimit.begin();

            m_pCurrentRunning->onInput();

            update();
            draw();

            while (m_bIsPaused == true) {
                m_pCurrentRunning->onInput();
            }

            m_fFps = fpsLimit.end();
        }
    }

    //Call Main Update loop
    void IMainGame::run() {
        mainLoop();
    }


    /*
        Main Inputmanager control
    */
    void IMainGame::onSDLEvent(SDL_Event& evnt) {
        m_InputManager.update();
        //Will keep looping until there are no more events to process
        if (evnt.key.repeat == 0)
        {


            switch (evnt.type) {
            case SDL_QUIT:
                exitGame();
                break;
            case SDL_MOUSEMOTION:
                m_InputManager.setMouseCoords((float)evnt.motion.x, (float)evnt.motion.y);
                break;
            case SDL_KEYDOWN:
                m_InputManager.pressKey(evnt.key.keysym.sym);
                break;
            case SDL_KEYUP:
                m_InputManager.releaseKey(evnt.key.keysym.sym);
                break;
            case SDL_MOUSEBUTTONDOWN:
                m_InputManager.pressKey(evnt.button.button);
                break;
            case SDL_MOUSEBUTTONUP:
                m_InputManager.releaseKey(evnt.button.button);
                break;
            case SDL_FINGERDOWN:
                m_InputManager.isSwipe(evnt);
                break;
            case SDL_FINGERMOTION:
                m_InputManager.isSwipe(evnt);
                break;
            case SDL_FINGERUP:
                m_InputManager.releaseKey(evnt.tfinger.fingerId);
                break;
            default:
                break;
            }
        }
    }

    /*
        Get users window information
    */
    void IMainGame::setupWindow(int w, int h, std::string windowName, std::string path, int
flag) {
        m_windowWidth = w;
        m_windowHeight = h;
        windowtitle = windowName;
        windowFlag = flag;

        mainAssetsPath = path;
        //Set windowss asset path
#ifdef __WIN32__
        FileIO::getInstance()->setAssetsPath(mainAssetsPath);
```

```
#endif // !__WIN32__

    }


    /*

        Init all Engine elements
    */
    bool IMainGame::init() {
        //Init Engine
        DsdlEngine::init();
        //Init audio Manager
        m_audioManager.init();

        //call game's on init method
        onInit();

        //If window creation fails exit
        if (!initSystems()) {
            SDL_Log("InitSystems Failed : Window not created");
            return false;
        }

        //Add all Scenes
        addScenes();

        //Load up First Scene
        m_pCurrentRunning = m_pSceneManager->getCurrentScene();
        m_pCurrentRunning->setSceneRunning();
        m_pCurrentRunning->onEntryScene();

        //Load all scene Children nodes for first scene on init of game
        for (size_t i = 0; i < m_pCurrentRunning->sceneLayers.size(); i++) {
            m_pCurrentRunning->loadScene(m_pGameRenderer);
        }

        //for running scene render each node that is in the child vector
        for (size_t i = 0; i < m_pCurrentRunning->sceneLayers.size(); i++) {
            m_pCurrentRunning->drawScene(m_pGameRenderer);
        }

        return true;
    }

    /*
        InitSystem
        Create window and get window render
    */
    bool IMainGame::initSystems() {
        m_Window.createWindow(windowtitle, m_windowWidth, m_windowHeight, windowFlag);
        m_pGameRenderer = m_Window.getRenderer();
        return true;
    }


    /*
        Call current scenes update
        Handel switching between scenes
    */

    void IMainGame::update() {
        if (m_pCurrentRunning) {
            switch (m_pCurrentRunning->getSceneState()) {
            case SceneState::RUNNING:
                m_pCurrentRunning->updateScene();
                break;
            case SceneState::CHANGE_NEXT:
                m_pCurrentRunning->onExitScene();
                m_pCurrentRunning = m_pSceneManager->moveNext();
                if (m_pCurrentRunning) {
                    m_pCurrentRunning->setSceneRunning();
                    m_pCurrentRunning->onEntryScene();
                    //Load all scene Children nodes for next scene
                    for (size_t i = 0; i < m_pCurrentRunning->sceneLayers.size(); i++) {
                        m_pCurrentRunning->loadScene(m_pGameRenderer);
                    }
                }
                break;
            case SceneState::CHANGE_PREVIOUS:
                m_pCurrentRunning->onExitScene();
                m_pCurrentRunning = m_pSceneManager->movePrevious();
                if (m_pCurrentRunning) {
                    m_pCurrentRunning->setSceneRunning();
                    m_pCurrentRunning->onEntryScene();
                    //Load all scene Children nodes for previous scene
                    for (size_t i = 0; i < m_pCurrentRunning->sceneLayers.size(); i++) {
                        m_pCurrentRunning->loadScene(m_pGameRenderer);
                    }
                }
                break;
            case SceneState::EXIT_APP:
                exitGame();
                break;
            default:
                break;
            }
        }
        else {
            exitGame();
        }
    }


    /*
        Render all Scene nodes to screen

    */

    void IMainGame::draw() {
        if (m_pCurrentRunning && m_pCurrentRunning->getSceneState() == SceneState::RUNNING) {
            SDL_RenderClear(m_pGameRenderer);

            //for running scene render each node that is in the child vector
            for (size_t i = 0; i < m_pCurrentRunning->sceneLayers.size(); i++) {
                m_pCurrentRunning->drawScene(m_pGameRenderer);
            }
            SDL_RenderPresent(m_pGameRenderer);
        }
    }



    /*
        Exit Game
    */
    void IMainGame::exitGame() {
        m_pCurrentRunning->onExitScene();
        if (m_pSceneManager) {
            m_pSceneManager->destroy();
            m_pSceneManager.reset();
        }
        m_bIsRunning = false;
    }
}
```

IMainGame.h File Reference

```
#include "EngineDefines.h"
#include "DsdlEngine.h"
#include "Window.h"
#include "InputManager.h"
#include "Timing.h"
#include "AudioManager.h"
#include "ResourceTexture.h"
#include "Layer.h"
```

Classes

- class DsdlEngine::IMainGame

Namespaces

- DsdlEngine

IMainGame.h

```
#ifndef _MAINGAME_
#define _MAINGAME_

#include "EngineDefines.h"
#include "DsdlEngine.h"
#include "Window.h"
#include "InputManager.h"
#include "Timing.h"
#include "AudioManager.h"
#include "ResourceTexture.h"
#include "Layer.h"
namespace DsdlEngine{

    class SceneManager;
    class IScene;

    class IMainGame{

    public:
        IMainGame();

        virtual ~IMainGame();

        void run();

        void setupWindow(int w, int h, std::string windowName, std::string path, int flag);

        void setFps(float fps){ m_fFps = fps; }

        virtual void onInit() = 0;

        virtual void addScenes() = 0;

        virtual void onExit() = 0;

        void onSDLEvent(SDL_Event& evnt);

        void setPaused() { m_bIsPaused = true; }

        void setRunning() { m_bIsPaused = false; m_bIsRunning = true; }

        bool checkPaused() { return m_bIsPaused; }

        InputManager m_InputManager;
    protected:
        //Scene Manager
        std::unique_ptr<SceneManager> m_pSceneManager;
        //Current Scene
        IScene* m_pCurrentRunning;
        bool m_bIsRunning, m_bIsPaused;
        Window m_Window;
        SDL_Renderer* m_pGameRenderer;
        AudioManager m_audioManager;
    private:

        //Game frame rate
        float m_fFps;
        //Game Windows details
        unsigned int windowFlag;
        int m_windowWidth;
        int m_windowHeight;
        std::string windowtitle;
        std::string mainAssetsPath;
        const float getFps() const { return m_fFps; }

        void mainLoop();

        void update();

        void draw();

        bool init();

        bool initSystems();

        void exitGame();

    };
}

#endif //!_MAINGAME_
```

InputManager.cpp File Reference

`#include "InputManager.h"`

Namespaces

- DsdlEngine

InputManager.cpp

```
#include "InputManager.h"

namespace DsdlEngine{


    InputManager::InputManager(){
        swipedown = false;
        swipeup = false;
        swipeleft = false;
        swiperight = false;
    }

    InputManager::~InputManager(){
        //Empty
    }


    void InputManager::update() {
        // Loop through _keyMap using a for each loop, and copy it over to _previousKeyMap
        for (auto& it : _keyMap) {
            _previousKeyMap[it.first] = it.second;
        }
    }

    void InputManager::pressKey(unsigned int keyID) {
        // if keyID doesn't already exist in _keyMap, it will get added
        _keyMap[keyID] = true;
    }

    void InputManager::releaseKey(unsigned int keyID) {
        _keyMap[keyID] = false;
    }

    void InputManager::setMouseCoords(float x, float y) {

    }


    bool InputManager::isKeyDown(unsigned int keyID) {
        // We dont want to use the associative array approach here
        // because we don't want to create a key if it doesnt exist.
        // So we do it manually
        auto it = _keyMap.find(keyID);
        if (it != _keyMap.end()) {
            // Found the key
            return it->second;
        }
        else {
            // Didn't find the key
            return false;
        }
    }

    bool InputManager::wasKeyDown(unsigned int keyID) {
        // We dont want to use the associative array approach here
        // because we don't want to create a key if it doesnt exist.
        // So we do it manually
        auto it = _previousKeyMap.find(keyID);
        if (it != _previousKeyMap.end()) {
            // Found the key
            return it->second;
        }
        else {
            // Didn't find the key
            return false;
        }
    }

    bool InputManager::isKeyPressed(unsigned int keyID) {
        // Check if it is pressed this frame, and wasn't pressed last frame
        if (isKeyDown(keyID) == true && wasKeyDown(keyID) == false) {
            pressKey(keyID);
            return true;
        }
        return false;
    }


    bool InputManager::isKeyReleased(unsigned int keyID) {
        // Check if it is pressed this frame, and wasn't pressed last frame
        if (isKeyDown(keyID) == false && wasKeyDown(keyID) == true) {
            releaseKey(keyID);
            return true;
        }
        return false;

    }


    bool InputManager::isTouch(unsigned int keyID) {

        if (keyID == SDL_FINGERDOWN) {
            return true;
        }
        return false;
    }



    bool InputManager::isSwipe(SDL_Event& evnt) {

        float startX, startY, endX, endY;

        if (evnt.type = SDL_FINGERMOTION) {

            startX = ((float)evnt.tfinger.x);
            startY = ((float)evnt.tfinger.y);
            fingerDown = true;

            if (evnt.type = SDL_FINGERUP) {
                endX = ((float)evnt.tfinger.x);
                endY = ((float)evnt.tfinger.y);
                fingerUp = true;
            }
        }


        if (fingerDown && fingerUp) {

            if (startX < endX) {
                //swipe down
                SDL_Log("SWIPE DOWN---Start : %f ----- End : %f", startX, endX);
                swipedown = true;
            }
            else if (startX > endX) {
                //swipe up
                SDL_Log("SWIPE UP---Start : %f ----- End : %f", startX, endX);
                swipeup = true;
            }
            else if (startY > endY) {
                //swipe left
                swipeleft = true;
                SDL_Log("SWIPE LEFT---Start : %f ----- End : %f", startY, endY);
            }
            else if (startY < endY) {
                //swipe right
                swiperight = true;
                SDL_Log("SWIPE RIGHT---Start : %f ----- End : %f", startY, endY);
            }
            else if (startX == endX) {
                SDL_Log("Only tough happened not swipe");
            }
        }
        return true;
    }



    bool InputManager::isSwipeUp() {

        return swipeup;
    }

    bool InputManager::isSwipeDown() {

        return swipedown;
    }

    bool InputManager::isSwipeLeft(float x, float y) {

        return swipeleft;
    }

    bool InputManager::isSwipeRight(float x, float y) {

        return swiperight;
    }

}
```

InputManager.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::InputManager

Namespaces

- DsdlEngine

InputManager.h

```
#ifndef _INPUTMANAGER_
#define _INPUTMANAGER_

#include "EngineDefines.h"
namespace DsdlEngine{
    class InputManager{
    public:
        InputManager();

        ~InputManager();

        void update();

        void pressKey(unsigned int keyID);

        void releaseKey(unsigned int keyID);

        void setMouseCoords(float x, float y);

        bool isKeyDown(unsigned int keyID);

        bool isKeyPressed(unsigned int keyID);

        bool isKeyReleased(unsigned int KeyID);

        bool isTouch(unsigned int keyID);

        bool isSwipe(SDL_Event& evnt);

        bool isSwipeUp();

        bool isSwipeDown();

        bool isSwipeLeft(float x, float y);

        bool isSwipeRight(float x, float y);

    private:

        bool wasKeyDown(unsigned int keyID);

        std::unordered_map<unsigned int, bool> _keyMap;
        std::unordered_map<unsigned int, bool> _previousKeyMap;
        bool swipeup, swipedown, swipeleft, swiperight;
        bool fingerDown, fingerUp;
    };
}
#endif
```

IScene.h File Reference

```
#include "EngineDefines.h"
#include "EngineBaseNode.h"
#include "Sprite.h"
#include "InputManager.h"
#include "Layer.h"
```

Classes

- class DsdlEngine::IScene

Namespaces

- DsdlEngine

Macros

- #define SCENE_INDEX_NO_SCENE -1

Enumerations

- enum DsdlEngine::SceneState { DsdlEngine::SceneState::NONE, DsdlEngine::SceneState::RUNNING, DsdlEngine::SceneState::EXIT_APP, DsdlEngine::SceneState::CHANGE_NEXT, DsdlEngine::SceneState::CHANGE_PREVIOUS }

Macro Definition Documentation

#define SCENE_INDEX_NO_SCENE -1

Definition at line 4 of file IScene.h.

IScene.h

```
 1 #ifndef _ISCENE_
 2 #define _ISCENE_
 3 
 4 #define SCENE_INDEX_NO_SCENE -1
 5 #include "EngineDefines.h"
 6 #include "EngineBaseNode.h"
 7 #include "Sprite.h"
 8 #include "InputManager.h"
 9 #include "Layer.h"
10 
15 namespace DsdlEngine {
16 
17     //forward declartion of class
18     class IMainGame;
19 
24     enum class SceneState {
25         NONE,
26         RUNNING,
27         EXIT_APP,
28         CHANGE_NEXT,
29         CHANGE_PREVIOUS
30     };
```

```

    class IScene {
    public:
        IScene() {
            //Empty
        };

        virtual ~IScene() {
            //Empty
        };

        virtual int getNextSceneIndex() const = 0;

        virtual int getPreviousSceneIndex() const = 0;

        // Called when a screen enters and exits focus

        virtual void onEntryScene() = 0;

        virtual void onExitScene() = 0;

        virtual void updateScene() = 0;

        virtual void destroyScene() = 0;

        int getSceneIndex() const { return m_iSceneIndex; }

        SceneState getSceneState() const { return m_eCurrentState; }

        void setSceneRunning() { m_eCurrentState = SceneState::RUNNING; }

        void setParentGame(IMainGame* game) { m_game = game; }

        virtual void onInput();

        std::vector<Layer*> sceneLayers;
        void addLayerToScene(Layer* layer) {
            sceneLayers.push_back(layer);
        }

        void loadScene(SDL_Renderer* r) {
            for (size_t i = 0; i < sceneLayers.size(); i++) {
                sceneLayers.at(i)->loadNodes(r);
            }
        }

        void drawScene(SDL_Renderer* r) {
            for (size_t i = 0; i < sceneLayers.size(); i++) {
                sceneLayers.at(i)->drawNodes(r);
            }
        }

    protected:
        friend class SceneManager;
        friend class InputManager;

        SceneState m_eCurrentState = SceneState::NONE;
        IMainGame* m_game = nullptr;
        int m_iSceneIndex = SCENE_INDEX_NO_SCENE;
        InputManager m_inputManager;
    };
}


#endif //!_ISCENE_
```

Label.cpp File Reference

```
#include "Label.h"
```

Namespaces

- DsdlEngine

Label.cpp

```
#include "Label.h"
/*
    Base Label Class
    author: Derek O Brien
    Description: label class for creating labels. inherits for EngineBaseNode
*/

namespace DsdlEngine{

    //Constructor
    Label::Label(){
        setEngineNodeType(NodeType::LABEL);
    }

    //Deconstructor
    Label::~Label() {
        destroy();
    }

    //Create Label
    void Label::create(Vec2 pos, std::string text, int txtsize, SDL_Color color, std::string fontFilePath){

        m_labelText = text;
        m_textSize = txtsize;
        m_textColor = color;

        setAssetPath(fontFilePath);

        m_position.x_ = pos.x_;
        m_position.y_ = pos.y_;
    }

    //Destroy Label
    void Label::destroy(){
        EngineBaseNode::destroy();
    }

    //Cleanup Label
    void Label::cleanup() {
        EngineBaseNode::cleanup();
    }
}
```

Label.h File Reference

`#include "EngineBaseNode.h"`

Classes

- class DsdlEngine::Label

Namespaces

- DsdlEngine

Label.h

```
#ifndef _LABEL_
#define _LABEL_

#include "EngineBaseNode.h"
namespace DsdlEngine{
    class Label : public EngineBaseNode{
    public:
        /***
        *   COnstructor
        */
        Label();

        virtual ~Label();

        void create(Vec2 pos, std::string text, int fontSize, SDL_Color color, std::string
fontFilePath);

        void setType(LableType type) { m_labelType = type; };

        const int getType() { return (int)m_labelType; }

        void destroy();

        void cleanup();

    protected:
        LableType m_labelType;
    };
}
#endif  //!_LABEL_
```

Layer.cpp File Reference

```
#include "Layer.h"
#include "Sprite.h"
```

Namespaces

- DsdlEngine

Layer.cpp

```
#include "Layer.h"
#include "Sprite.h"

/*
    Base Layer Class
    author: Derek O Brien
    Description: Layer base class for all layers in game.
*/

namespace DsdlEngine {

    //Constructor
    Layer::Layer() {
        //Emptty
        layerNodes.reserve(20);
    }

    //Deconstructor
    Layer ::~Layer() {
        destroy();
    }

    //Destroy layer nodes and cleanup
    void Layer::destroy() {
        for (size_t i = 0; i < layerNodes.size(); i++) {
            layerNodes.erase(std::remove(layerNodes.begin(), layerNodes.end(),
layerNodes[i]), layerNodes.end());

            layerNodes[i]->destroy();

            //layerNodes.shrink_to_fit();
            //delete(layerNodes[i]);
        }
        layerNodes.clear();
    }


    //Add Engine node to layer for loading and rendering
    void Layer::addNodeToLayer(EngineBaseNode* node) {
        layerNodes.push_back(node);
    }


    //Remove Node from sceen Vector
    void Layer::removeNodeFromLayer(EngineBaseNode* node) {
        layerNodes.erase(std::remove(layerNodes.begin(), layerNodes.end(), node),
layerNodes.end());
        node->destroy();
    }

    //Load all nodes added to layer
    void Layer::loadNodes(SDL_Renderer* r) {
        for (size_t i = 0; i < layerNodes.size(); i++) {
            layerNodes.at(i)->load(r);
        }
    }

    //Render all nodes added to layer
    void Layer::drawNodes(SDL_Renderer* r) {
        for (size_t i = 0; i < layerNodes.size(); i++) {

            if (layerNodes.at(i)->getNodeType() == NodeType::SPRITE) {

                //Reload texture if the texture has been changed
                if (layerNodes.at(i)->isTextureChanged() == true) {
                    layerNodes.at(i)->cleanup();
                    layerNodes.at(i)->load(r);
                    layerNodes.at(i)->setUpdateTextureTrue(false);
                }
                layerNodes.at(i)->render(r);
            }

            /*
            Reload labels for update each tick for changes to take effect
            old label destroyed first to realease its memory so no extra memory been taking up
            */
            if (layerNodes.at(i)->getNodeType() == NodeType::LABEL) {
                layerNodes.at(i)->cleanup();
                layerNodes.at(i)->load(r);
                layerNodes.at(i)->render(r);
            }
        }
    }

}
```

Layer.h File Reference

```
#include "EngineBaseNode.h"
#include "ResourceTexture.h"
```

Classes

- class DsdlEngine::Layer

Namespaces

- DsdlEngine

Layer.h

```
#ifndef _LAYER_
#define _LAYER_

#include "EngineBaseNode.h"
#include "ResourceTexture.h"

namespace DsdlEngine {
    /***
    *   Base layer class for the engine
    */
    class Layer {
    public:
        //Add Gui Class As friend class
        friend class Gui;

        Layer();

        virtual ~Layer();

        virtual void destroy();

        void loadNodes(SDL_Renderer* r);

        void drawNodes(SDL_Renderer* r);

        void addNodeToLayer(EngineBaseNode* node);

        void removeNodeFromLayer(EngineBaseNode* node);

        std::vector<EngineBaseNode*> layerNodes;
    private:

    };
}
#endif //!_LAYER_
```

Particles.cpp File Reference

`#include "Particles.h"`

Namespaces

- DsdlEngine

Particles.cpp

```

#include "Particles.h"

namespace DsdlEngine {

    Particles::Particles(int x, int y) {
        //Set offsets
        mPosX = x - 5 + (rand() % 25);
        mPosY = y - 5 + (rand() % 25);

        //Initialize animation
        m_frame = rand() % 5;

    }

    Particles::~Particles() {}


    bool Particles::isDead(Particles *p)
    {
        if ((p->life < 0) || (p->size < 1))
            return true;
        return false;
    }


}
```

Particles.h File Reference

```
#include "DsdlEngine.h"
```

Classes

- class DsdlEngine::Particles

Namespaces

- DsdlEngine

Macros

- #define MAL_PARTICLE_LIFE   500

Macro Definition Documentation

#define MAL_PARTICLE_LIFE   500

Definition at line 7 of file Particles.h.

Particles.h

```
 1 #ifndef __Particles__
 2 #define __Particles__
 3 #include "DsdlEngine.h"
 4
 5 namespace DsdlEngine {
 6
 7 #define MAL_PARTICLE_LIFE 500
 8
 9
10     class Particles : public EngineBaseNode{
11     public:
12         Particles(int x, int y);
13         ~Particles();
14
15
16         bool isDead(Particles *p);
17
18         static inline float torad(float angle){
19             return (angle * M_PI) / 180;
20         }
21
22
23     private:
24
25         int life;
26         float mPosX, mPosY, xvel, yvel, angle, size;
27         Uint32 endtime;
28
29     };
30 }
31 #endif // !__Particles__
```

ResourceTexture.cpp File Reference

```
#include "ResourceTexture.h"
#include "EngineError.h"
#include "FileIO.h"
```

Namespaces

- DsdlEngine

ResourceTexture.cpp

```

/*
    Engine Texture
    Handles all the loading for different textures in the engine
    Handles main call to render texture

    Link between ( Engine / Game ) and SDL Textures
*/


#include "ResourceTexture.h"
#include "EngineError.h"
#include "FileIO.h"

/*
    FIle : ResourcTexture
    Author: Derek O Brien
    Description: Load & Render Image and TTf media into Sdl Texture
*/

namespace DsdlEngine{

    //Constructor
    ResourceTexture::ResourceTexture(){
        m_Texture = NULL;
        m_iHeight = 0;
        m_iWidth = 0;
    }

    //Deconstructor
    ResourceTexture::~ResourceTexture(){
        destroy();
    }


    //Load Sprite from file
    bool ResourceTexture::loadTexture(std::string texturePath, SDL_Renderer* r){

        std::string temp = FileIO::getInstance()->getWritablePath() + texturePath;
        //Store in map for loading
        auto it = m_TextureMap.find(temp);

        SDL_Texture* newTexture = NULL;
        SDL_Surface* loadedSurface = NULL;

        if (it == m_TextureMap.end()){
            //Load image at specified path
            loadedSurface = IMG_Load(temp.c_str());


            if (loadedSurface == NULL)
                SDL_Log("SDL_image Error : %s ", IMG_GetError());
            else{
                //Create texture from surface pixels
                newTexture = SDL_CreateTextureFromSurface(r, loadedSurface);
                if (newTexture == NULL){
                    SDL_Log("SDL_CreateTextureFromSurface Error : %s ",IMG_GetError());
                }
                else{
                    //Get image dimensions
                    m_iWidth = loadedSurface->w;
                    m_iHeight = loadedSurface->h;
                }
                //Get rid of old loaded surface
                SDL_FreeSurface(loadedSurface);
            }
             //Add to map
            m_TextureMap[temp] = newTexture;
        }
        else{
            newTexture = it->second;
        }
        //Return success
        m_Texture = newTexture;
        return m_Texture != NULL;
    }


    //Load ttf to sdl texture
    bool ResourceTexture::loadTTF(std::string text, SDL_Color color, TTF_Font* myfont, SDL_Renderer* r){

        this->destroy();
        //Create font as surface
        SDL_Surface* textSurface = TTF_RenderText_Blended(myfont, text.c_str(), color);

        if (textSurface == NULL){
            SDL_Log("TTF_RenderText_Blended Error : %s", TTF_GetError());
        }
        else{//COnvert Surface to the Texture
            m_Texture = SDL_CreateTextureFromSurface(r, textSurface);
            if (m_Texture == NULL){
                SDL_Log("TTF_RenderText_Blended Error : %s" , TTF_GetError());
            }
            else{
                m_iWidth = textSurface->w;
                m_iHeight = textSurface->h;
            }
            //Free surface as no longer needed
            SDL_FreeSurface(textSurface);
        }
        //Return it
        return m_Texture != NULL;
    }


    //Basic render
    void ResourceTexture::render(Vec2 p, Vec2 s, SDL_Renderer* r, SDL_Rect* clip){
        //Set rendering space and render to screen
        SDL_Rect renderQuad;
        if (s.x_ != NULL && s.y_ != NULL) {//For Sprites
            renderQuad = { p.x_, p.y_, s.x_, s.y_ };
        }
        else {  //For TTf Labels
            renderQuad = { p.x_, p.y_, m_iWidth, m_iHeight };
        }

        //Set clip rendering dimensions
        if (clip != NULL){
            renderQuad.w = clip->w;
            renderQuad.h = clip->h;
        }

        //Render to screen
        SDL_RenderCopy(r, m_Texture, clip, &renderQuad);
    }


    //Clean up
    void ResourceTexture::destroy(){
        if (m_Texture != NULL) {
            SDL_DestroyTexture(m_Texture);
            m_Texture = NULL;
            m_iWidth = 0;
            m_iHeight = 0;
        }
    }

    /*
        Set Texture Blend Mode
    */
    void ResourceTexture::setBlendMode(SDL_BlendMode blend){
        SDL_SetTextureBlendMode(m_Texture, blend);
    }


    /*
        Set Alpha value of texture for transperence
        @parma alpha value of texture alpha 0 to 255
    */
    void ResourceTexture::setAlpha(Uint8 alpha){
        SDL_SetTextureAlphaMod(m_Texture, alpha);
    }

}
```

ResourceTexture.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::ResourceTexture

Namespaces

- DsdlEngine

ResourceTexture.h

```
#ifndef _RESOURCETEXTURE_
#define _RESOURCETEXTURE_

#include "EngineDefines.h"

namespace DsdlEngine{

    class ResourceTexture{
    public:
        ResourceTexture();

        ~ResourceTexture();

        bool loadTexture(std::string texturePath, SDL_Renderer* r);

        bool loadTTF(std::string text, SDL_Color color, TTF_Font* myFont, SDL_Renderer* r);

        void render(Vec2 p, Vec2 s, SDL_Renderer* r, SDL_Rect* clip = NULL);

        void setBlendMode(SDL_BlendMode blending);

        void setAlpha(Uint8 alpha);

        void destroy();
    private:

        SDL_Texture* m_Texture;
        std::map<std::string, SDL_Texture*> m_TextureMap;
        int m_iWidth, m_iHeight;
    };
}

#endif // !_RESOURCETEXTURE_
```

Scene.cpp File Reference

```
#include "IScene.h"
```

Namespaces

- DsdlEngine

Scene.cpp

```
#include "IScene.h"
/*
    Base Scene Class
    author: @Derek O Brien
    Description: Interface for base scene in game.
*/


namespace DsdlEngine {


    void IScene::onInput() {
        SDL_Event evnt;
        m_inputManager.update();

        while (SDL_PollEvent(&evnt)) {

            if (evnt.key.repeat == 0) {


                switch (evnt.type) {
                case SDL_QUIT:
                    exit(1);
                    break;
                case SDL_MOUSEMOTION:
                    m_inputManager.setMouseCoords(evnt.motion.x, evnt.motion.y);
                    break;
                case SDL_KEYDOWN:
                    m_inputManager.pressKey(evnt.key.keysym.sym);
                    break;
                case SDL_KEYUP:
                    m_inputManager.releaseKey(evnt.key.keysym.sym);
                    break;
                case SDL_MOUSEBUTTONDOWN:
                    m_inputManager.pressKey(evnt.button.button);
                    break;
                case SDL_MOUSEBUTTONUP:
                    m_inputManager.releaseKey(evnt.button.button);
                    break;
                    //Touch down
                case SDL_FINGERDOWN:
                    m_inputManager.pressKey(evnt.button.button);
                    break;
                case SDL_FINGERMOTION:
                    m_inputManager.setMouseCoords((float)evnt.motion.x,
(float)evnt.motion.y);
                    break;
                case SDL_FINGERUP:
                    m_inputManager.releaseKey(evnt.button.button);
                    break;
                default:
                    break;
                }
            }
        }
    }
}
```

SceneManager.cpp File Reference

```
#include "SceneManager.h"
#include "IScene.h"
```

Namespaces

- DsdlEngine

SceneManager.cpp

```
#include "SceneManager.h"
#include "IScene.h"
namespace DsdlEngine{

    //Constructor sets links scene manager and ImainGame
    SceneManager::SceneManager(IMainGame* game) :
        m_pGame(game){
        //Empty
    }

    //Move to next scene
    IScene* SceneManager::moveNext(){
        IScene* currentScene = getCurrentScene();
        if (currentScene->getNextSceneIndex() != SCENE_INDEX_NO_SCENE){
            m_iCurrentSceneIndex = currentScene->getNextSceneIndex();
        }
        return getCurrentScene();
    }

    //Move to Previous Scene
    IScene* SceneManager::movePrevious(){
        IScene* currentScene = getCurrentScene();
        if (currentScene->getPreviousSceneIndex() != SCENE_INDEX_NO_SCENE){
            m_iCurrentSceneIndex = currentScene->getPreviousSceneIndex();
        }
        return getCurrentScene();
    }

    //Set Current scene
    void SceneManager::setScene(int nextScene){
        m_iCurrentSceneIndex = nextScene;
    }

    //Add a scene to game
    void SceneManager::addScene(IScene* newScene){
        newScene->m_iSceneIndex = m_pScenes.size();
        m_pScenes.push_back(newScene);
        newScene->setParentGame(m_pGame);
    }

    //Clean up scenes
    void SceneManager::destroy(){
        for (size_t i = 0; i < m_pScenes.size(); i++){
            m_pScenes[i]->destroyScene();
            //delete m_pScenes[i];
        }
        m_pScenes.clear();
        m_iCurrentSceneIndex = SCENE_INDEX_NO_SCENE;
    }

    //Return the current scene
    IScene* SceneManager::getCurrentScene(){
        if (m_iCurrentSceneIndex == SCENE_INDEX_NO_SCENE) {
            return nullptr;
        }
        return m_pScenes[m_iCurrentSceneIndex];
    }

}
```

SceneManager.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::SceneManager

Namespaces

- DsdlEngine

SceneManager.h

```
#ifndef _SCREENMANAGER_
#define _SCREENMANAGER_

#include "EngineDefines.h"

namespace DsdlEngine{

    //Forward declearation of classes
    class IMainGame;
    class IScene;

    class SceneManager{
    public:

        SceneManager(IMainGame* game);

        ~SceneManager(){ destroy(); };

        IScene* moveNext();

        IScene* movePrevious();

        void setScene(int nextScene);

        void addScene(IScene* newScene);

        void destroy();

        IScene* getCurrentScene();

    protected:

        IMainGame* m_pGame;
        std::vector<IScene*> m_pScenes;
        int m_iCurrentSceneIndex;
    };
}
#endif  //!_SCREENMANAGER_
```

Sprite.cpp File Reference

```
#include "Sprite.h"
```

Namespaces

- DsdlEngine

Sprite.cpp

```
#include "Sprite.h"

/*
    File: Sprite.h
    Author: Derek O Brien
    Description: Sprite file for creating in game sprites. Inherits from engine base node
*/
namespace DsdlEngine{

    //Constructor
    Sprite::Sprite(){
        setEngineNodeType(NodeType::SPRITE);
        m_frame = 0;
        m_numFrames = 1;
        m_opacity = 255;
        m_objectBoundingBox = new SDL_Rect();
        m_engineTexture = new ResourceTexture();
    }

    //DeConstructor
    Sprite::~Sprite() {
        destroy();
    }


    //Create basic sprite with one frame
    void Sprite::create(Vec2 spriteSize, Vec2 position, std::string path){
        setAssetPath(path);

        m_size.x_ = spriteSize.x_;
        m_size.y_ = spriteSize.y_;

        m_position.x_ = position.x_;
        m_position.y_ = position.y_;
        setPosition(position);

        m_numFrames = 1;
        setBoundingBox(position, spriteSize);
    }


    //Create basic sprite with more than one frame
    void Sprite::create(Vec2 spriteSize, Vec2 position, std::string path, int nf){
        setAssetPath(path);

        m_size.x_ = spriteSize.x_;
        m_size.y_ = spriteSize.y_;

        m_position.x_ = position.x_;
        m_position.y_ = position.y_;
        setPosition(position);


        setBoundingBox(position, spriteSize);
        m_numFrames = nf;
    }

```

```cpp
    //Create basic sprite with more than one frame and physics body attached
    void Sprite::createWithPhysics(b2World* world, Vec2 spriteSize, Vec2 position, std::string path, int numFrames, float den, float fri, bool FixedRotation) {
        setAssetPath(path);

        m_size.x_ = spriteSize.x_;
        m_size.y_ = spriteSize.y_;

        m_position.x_ = position.x_;
        m_position.y_ = position.y_;

        setPosition(position);

        setBoundingBox(position, spriteSize);
        m_numFrames = numFrames;

        //Create new collision shape
        m_CollisionShape = new CollisionShape();
        m_CollisionShape->init(world, position, spriteSize, den, fri, FixedRotation);
    }

    //Destroy Sprite
    void Sprite::destroy(){
        EngineBaseNode::destroy();
    }

    //Change sprite texture after sprite is loaded
    void Sprite::updateTexure(Vec2 spriteSize, Vec2 position, std::string path, int numFrames) {

        setUpdateTextureTrue(true);

        setAssetPath(path);

        m_size.x_ = spriteSize.x_;
        m_size.y_ = spriteSize.y_;

        m_position.x_ = position.x_;
        m_position.y_ = position.y_;
        setPosition(position);


        setBoundingBox(position, spriteSize);
        m_numFrames = numFrames;

    }
}
```

Sprite.h File Reference

`#include "EngineBaseNode.h"`

Classes

- class DsdlEngine::Sprite

Namespaces

- DsdlEngine

Sprite.h

```
#ifndef _SPRITE_
#define _SPRITE_

#include "EngineBaseNode.h"
namespace DsdlEngine{

    class Sprite : public EngineBaseNode{
    public:
        Sprite();

        virtual ~Sprite();

        void create(Vec2 spriteSize, Vec2 position, std::string path);

        void create(Vec2 spriteSize, Vec2 position, std::string path, int numFrames);

        void createWithPhysics(b2World* world, Vec2 spriteSize, Vec2 position, std::string
path, int numFrames, float den, float fri, bool FixedRotation);

        void updateTexure(Vec2 spriteSize, Vec2 position, std::string path, int numFrames);

        b2Body* getCollisionBody() { return m_CollisionShape->getBody(); }

        void destroy();

    private:

    };
}

#endif //!_SPRITE_
```

Timing.cpp File Reference

```
#include "EngineDefines.h"
#include "Timing.h"
```

Namespaces

- DsdlEngine

Timing.cpp

```
#include "EngineDefines.h"
#include "Timing.h"

/*
    File: timimg.cpp
    Author: Derek O Brien
*/
namespace DsdlEngine{

    FpsLimiter::FpsLimiter() {
        //Empty
    }
    FpsLimiter::~FpsLimiter() {
        //Empty
    }

    //Initilaze frame rate
    void FpsLimiter::init(float maxFPS) {
        setMaxFPS(maxFPS);
    }

    //Set Max Frame Rate
    void FpsLimiter::setMaxFPS(float maxFPS) {
        m_fMaxFPS = maxFPS;
    }

    //Get start timer
    void FpsLimiter::begin() {
        m_iStartTicks = SDL_GetTicks();
    }

    //End timer
    float FpsLimiter::end() {
        calculateFPS();

        float frameTicks = (float)(SDL_GetTicks() - m_iStartTicks);
        //Limit the FPS to the max FPS
        if (1000.0f / m_fMaxFPS > frameTicks) {
            SDL_Delay((Uint32)(1000.0f / m_fMaxFPS - frameTicks));
        }

        return m_fFps;
    }

    void FpsLimiter::calculateFPS() {
        //The number of frames to average
        static const int NUM_SAMPLES = 10;
        //Stores all the frametimes for each frame that we will average
        static float frameTimes[NUM_SAMPLES];
        //The current frame we are on
        static int currentFrame = 0;
        //the ticks of the previous frame
        static Uint32 prevTicks = SDL_GetTicks();

        //Ticks for the current frame
        Uint32 currentTicks = SDL_GetTicks();

        //Calculate the number of ticks (ms) for this frame
        m_fFrameTime = (float)(currentTicks - prevTicks);
        frameTimes[currentFrame % NUM_SAMPLES] = m_fFrameTime;

        //current ticks is now previous ticks
        prevTicks = currentTicks;

        //The number of frames to average
        int count;

        currentFrame++;
        if (currentFrame < NUM_SAMPLES) {
            count = currentFrame;
        }
        else {
            count = NUM_SAMPLES;
        }

        //Average all the frame times
        float frameTimeAverage = 0;
        for (int i = 0; i < count; i++) {
            frameTimeAverage += frameTimes[i];
        }
        frameTimeAverage /= count;

        //Calculate FPS
        if (frameTimeAverage > 0) {
            m_fFps = 1000.0f / frameTimeAverage;
        }
        else {
            m_fFps = 120.0f;    //MAX ALLOWED
        }
    }
}
```

Timing.h File Reference

Classes

- class DsdlEngine::FpsLimiter

Namespaces

- DsdlEngine

Timing.h

```
#ifndef _TIMING_
#define _TIMING_

namespace DsdlEngine{

    class FpsLimiter {
    public:
        FpsLimiter();

        ~FpsLimiter();

        void init(float maxFPS);

        void setMaxFPS(float maxFPS);

        void begin();

        float end();

    private:
        void calculateFPS();

        float m_fFps, m_fMaxFPS, m_fFrameTime;
        unsigned int m_iStartTicks;
    };
}

#endif
```

Window.cpp File Reference

```
#include "Window.h"
#include "EngineError.h"
```

Namespaces

- DsdlEngine

Window.cpp

```
#include "Window.h"
#include "EngineError.h"
/*
    File : Window.h
    Author: Derek O Brien
    Description: set up and create Window and render for sdl window
*/
namespace DsdlEngine{

    Window::Window(){
        //Empty
    }

    Window::~Window() {
        destroy();
    }

    //Create Sdl Window
    int Window::createWindow(std::string windowName, int screenWidth, int screenHeight, 
unsigned int flag){

        m_screenHeight = screenHeight;
        m_screenWidth = screenWidth;

        //Screen dimensions
        SDL_Rect gScreenRect = { 0, 0, 320, 240 };
        SDL_DisplayMode displayMode;
        if (SDL_GetCurrentDisplayMode(0, &displayMode) == 0)
        {
            gScreenRect.w = displayMode.w;
            gScreenRect.h = displayMode.h;
        }

        //Load Window for windows using size passed in
#ifdef __WIN32__
        SDL_Log("Windows Created for Windows Platform");
        m_pSdlWindow = SDL_CreateWindow(windowName.c_str(), 0, 0, screenWidth, screenHeight, 
flag);
        m_pSdlRenderer = SDL_CreateRenderer(m_pSdlWindow, -1, SDL_RENDERER_TARGETTEXTURE | 
SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
        SDL_SetRenderDrawColor(m_pSdlRenderer, 0, 0, 0, 120);
#endif

        //Load Window for Android using device screen Size
#ifdef __ANDROID__
        SDL_Log("Windows Created for Android Platform");
        m_pSdlWindow = SDL_CreateWindow(windowName.c_str(), SDL_WINDOWPOS_UNDEFINED, 
SDL_WINDOWPOS_UNDEFINED, gScreenRect.w, gScreenRect.h, SDL_WINDOW_ALLOW_HIGHDPI);
        m_pSdlRenderer = SDL_CreateRenderer(m_pSdlWindow, -1, SDL_RENDERER_TARGETTEXTURE | 
SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
        SDL_SetRenderDrawColor(m_pSdlRenderer, 0, 0, 0, 120);
#endif


        if (m_pSdlWindow == nullptr){
            SDL_Log("Window could not be created! SDL Error: %s\n", SDL_GetError());
            SDL_Quit();
        }


        //Initialize PNG loading
        int imgFlags = IMG_INIT_PNG;
        if (!(IMG_Init(imgFlags) & imgFlags)){
            SDL_Log("SDL_image could not initialize! SDL_image Error %s\n", IMG_GetError());
            SDL_Quit();
        }

        SDL_Log("Window Created");

        return 0;
    }

    //Swap Window Buffer
    void Window::swapBuffer(){
        SDL_GL_SwapWindow(m_pSdlWindow);
    }

    //Destroy Window and Renderer
    void Window::destroy() {
        SDL_DestroyRenderer(m_pSdlRenderer);
        SDL_DestroyWindow(m_pSdlWindow);
        m_pScreenSurface = nullptr;
        SDL_Quit();
    }

}
```

Window.h File Reference

`#include "EngineDefines.h"`

Classes

- class DsdlEngine::Window

Namespaces

- DsdlEngine

Window.h

```
#ifndef _WINDOW_
#define _WINDOW_

#include "EngineDefines.h"

//Wrap Frame Work Code in Namespace
namespace DsdlEngine{

    class Window{
    public:
        Window();

        virtual ~Window();


        int createWindow(std::string windowNname, int screenWidth, int screenHeight, unsigned int flag);

        void swapBuffer();

        int getScreenWidth() { return m_screenWidth; }

        int getScreenHeight() { return m_screenHeight; }

        SDL_Renderer* getRenderer(){ return m_pSdlRenderer; }

        void destroy();
    private:

        SDL_Window* m_pSdlWindow;
        SDL_Renderer* m_pSdlRenderer;
        SDL_Surface* m_pScreenSurface;
        int m_screenHeight, m_screenWidth;
    };
}
#endif  //!_WINDOW_
```

XmlLocalStorage.cpp File Reference

```
#include "XmlLocalStorage.h"
#include "FileIO.h"
```

Namespaces

- DsdlEngine

Variables

- static XmlLocalStorage * DsdlEngine::Instance = nullptr

XmlLocalStorage.cpp

```
#include "XmlLocalStorage.h"
#include "FileIO.h"

/*
    File: XmlLocalStorage
    Author: Derek O Brien
    Description: For loading and saving values to XML file
*/

using namespace tinyxml2;
using namespace std;

namespace DsdlEngine {

    //Create As Singleton static instance
    static XmlLocalStorage* Instance = nullptr;
    XmlLocalStorage* XmlLocalStorage::getInstance() {
        if (!Instance) {
            Instance = new (std::nothrow) XmlLocalStorage();
        }
        return Instance;
    }



    //get integet value for key passed in
    int XmlLocalStorage::getIntegerForKey(const char* key) {

        const char* value = nullptr;
        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;

        //Get node from xml file
        node = FileIO::getInstance()->getXMLNodeForKey(key, &rootNode, &doc);

        //Get the value from the node
        if (node && node->FirstChild()) {
            value = (const char*)(node->FirstChild()->Value());
        }

        //Convert value to type needed
        int temp = 0;
        if (value) {
            temp = SDL_atoi(value);
        }

        if (doc) delete doc;

        return temp;
    }


    //Get bool Value for key
    bool XmlLocalStorage::getBoolForKey(const char* key) {

        const char* value = nullptr;
        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;

        //Get node from xml file
        node = FileIO::getInstance()->getXMLNodeForKey(key, &rootNode, &doc);

        //Get the value from the node
        if (node && node->FirstChild()) {
            value = (const char*)(node->FirstChild()->Value());
        }

        //Convert value to type needed

        bool temp = true;
        if (value) {
            temp = (!strcmp(value, "true"));
        }

        if (doc) delete doc;

        return temp;
    }


    //Get Double  Value for key passed in
    double XmlLocalStorage::getDoubleForKey(const char* key) {

        const char* value = nullptr;
        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;

        //Get node from xml file
        node = FileIO::getInstance()->getXMLNodeForKey(key, &rootNode, &doc);

        //Get the value from the node
        if (node && node->FirstChild()) {
            value = (const char*)(node->FirstChild()->Value());
        }

        //Convert value to type needed
        double temp = 0.0;

        if (value) {
            temp = SDL_atof(value);
        }

        if (doc) delete doc;

        return temp;
    };

    //Get float value for key passed in
    float XmlLocalStorage::getFloatForKey(const char* key) {
        float temp = (float)getDoubleForKey(key);
        return temp;
    };

    //Get String value for key passed in
    std::string XmlLocalStorage::getStringForKey(const char* key) {

        const char* value = nullptr;
        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;

        //Get node from xml file
        node = FileIO::getInstance()->getXMLNodeForKey(key, &rootNode, &doc);

        //Get the value from the node
        if (node && node->FirstChild()) {
            value = (const char*)(node->FirstChild()->Value());
        }

        //Convert value to type needed
        string temp = "No Value Found";

        if (value) {
            temp = string(value);
        }

        return temp;
    }

    //Set a string value for the key
    void XmlLocalStorage::setIntegerForKey(int value, const char* key) {
        // check key
        if (!key) {
            return;
        }

        // format the value as char for saving
        char tmp[50];
        memset(tmp, 0, 50);
#ifdef __WIN32__
        sprintf_s(tmp, "%d", value);
#endif

#ifdef __ANDROID__
        sprintf(tmp, "%d", value);
#endif

        //Save the Value and key
        FileIO::getInstance()->setValueForKey(tmp, key);
    }

    //Set bool value for key passed in
    void XmlLocalStorage::setBoolForKey(bool value, const char* key) {
        if (value == true) {
            setStringForKey("true", key);
        }
        else {
            setStringForKey("false", key);
        }
    }

    //Set double value for key passed in
    void XmlLocalStorage::setDoubleForKey(double value, const char* key) {
        // check key
        if (!key) {
            return;
        }

        // format the value as char for saving
        char tmp[50];
        memset(tmp, 0, 50);
#ifdef __WIN32__
        sprintf_s(tmp, "%f", value);
#endif

#ifdef __ANDROID__
        sprintf(tmp, "%f", value);
#endif

        //Save the value and key
        FileIO::getInstance()->setValueForKey(tmp, key);
    }

    //Set float value for key
    void XmlLocalStorage::setFloatForKey(float value, const char* key) {
        setDoubleForKey(value, key);
    }

    //Set String value for key
    void XmlLocalStorage::setStringForKey(std::string value, const char* key) {
        if (!key) return;

        FileIO::getInstance()->setValueForKey(value.c_str(), key);
    }


    //Delete value fo node
    void XmlLocalStorage::deleteValueForKey(const char* key) {

        XMLElement* rootNode;
        XMLDocument* doc;
        XMLElement* node;
        XMLPrinter printer;

        // check the params
        if (!key) {
            return;
        }

        // find the node
        node = FileIO::getInstance()->getXMLNodeForKey(key, &rootNode, &doc);

        // if node not exist, don't need to delete
        if (!node) {
            return;
        }

        if (doc){
            //Delete Node
            doc->DeleteNode(node);
            std::string path;
            path = FileIO::getInstance()->getWritablePath() + "Default.xml";

            // attach printer to the document you want to convert in to a std::string
            doc->Accept(&printer);

            // Create a std::string and copy your document data in to the string
            const char* buffer = printer.CStr();

            //Write back to file and save file
            if (FileIO::getInstance()->writeDocument(path.c_str(), &buffer)) {
                SDL_Log("Key : %s :: deleted", key);
            }
            delete doc;
        }
    }
}
```

XmlLocalStorage.h File Reference

```
#include "DsdlEngine.h"
#include "EngineDefines.h"
#include "../dependencies/tinyxml/tinyxml2.h"
```

Classes

- class DsdlEngine::XmlLocalStorage

Namespaces

- DsdlEngine

XmlLocalStorage.h

```
#ifndef _XMLLOCALSTORAGE_
#define _XMLLOCALSTORAGE_

#include "DsdlEngine.h"
#include "EngineDefines.h"
#include "../dependencies/tinyxml/tinyxml2.h"

namespace DsdlEngine {

    class XmlLocalStorage {
    public:

        static XmlLocalStorage* getInstance();

        void setIntegerForKey(int value, const char* key);

        void setBoolForKey(bool value, const char* key);

        void setDoubleForKey(double value, const char* key);

        void setFloatForKey(float value, const char* key);

        void setStringForKey(std::string value, const char* key);


        int getIntegerForKey(const char* key);

        bool getBoolForKey(const char* key);

        double getDoubleForKey(const char* key);

        float getFloatForKey(const char* key);

        std::string getStringForKey(const char* key);

        void deleteValueForKey(const char* key);

    protected:

        XmlLocalStorage() {};

        virtual ~XmlLocalStorage() {};

    private:

    };
}
#endif // !_XMLLOCALSTORAGE_
```

Index