Frost2D Framework 1.0 Documentation PDF

GameFuse, LLC

Frost2D Framework 1.0 Reference

Class References

Generated by appledoc 2.1 (build 858).

Frost2D Framework 1.0 Hierarchy

Class Hierarchy

- CAEmitterCell
  - F2DParticles
- F2DConstants
- F2DFunctions
- NSObject
  - F2DAudio
  - F2DConstraint
  - F2DMessage
  - Frost2D
- UIImageView
  - F2DBody
- UIScrollView
  - F2DWorld

Generated by appledoc 2.1 (build 858).

F2DAudio Class Reference

Inherits from	NSObject
Conforms to	AVAudioPlayerDelegate
Declared in	F2DAudio.h

Overview

This class plays and controls loaded music and sound files.

The Frost2D Framework breaks audio down into two categories: sound (e.g. sound effects) and music (e.g. background music). Sound and music files are first both loaded into Frost2D (see the Frost2D class for the proper way of loading sound and music files) then played using their assigned key values. Below is a description of the differences in the way the Frost2D Framework handles sound and music.

- Sound: All sound files are played through Open-AL for optimal performance. Multiple sounds can be played simultaneously (across 32 different sources) and quickly. Sound files should be fairly short and are not meant to be large audio files. Note that sound files have very limited controls, they can only be played. However they are highly optimized for sound effect playback. See Frost2D class for information about loading sound files and proper formatting.
- Music: All music files are played through the AVAudioPlayer. Only one music file should be played at a time. Music files are of high quality and can be large audio files. Music files can notify you when they finish playing through the Frost2D delegate. Music files do not play well quickly and simultaneously so they should not be used as repetitive "sound effects." Music files however offer a lot more control. They can be played, stopped, paused, looped etc. Even the entire AVAudioPlayer instance of the music object can be returned for extreme customization. See Frost2D class for information about loading music files and proper formatting.

Note that the Frost2D Framework takes care of all music and sound interruptions and cleanup for you. Audio files are simply loaded once and can be played at any time using the F2DAudio class methods.

Note that both sound and/or music can be turned off by simply adjusting the F2DSoundOn and F2DMusicOn global variables in the F2DConstants. This is often useful to provide audio options for your App. Note that checking whether or not sound/music is enabled before playing an audio file is not needed. By simply toggling these global variables on and off, sound/music simply will or will not be played. Do not subclass.

Tasks

Sounds

`+ playSoundWithKey:gain:pitch:`

Music

`+ playMusicWithKey:repeatCount:volume:`

+ stopMusicWithKey:
+ getAVAudioPlayerWithKey:

Class Methods

getAVAudioPlayerWithKey:

Retrieves the AVAudioPlayer of the music key.

```
+ (AVAudioPlayer *)getAVAudioPlayerWithKey:(NSString *)key
```

Parameters

key

The associated music file key.

Return Value

The AVAudioPlayer object representing the music key.

Discussion

The AVAudioPlayer can be used to further customize music.

See Also

AVAudioPlayer for more information.

Declared In

F2DAudio.h

pauseMusicWithKey:

Pauses a music file using its associated key.

```
+ (void)pauseMusicWithKey:(NSString *)key
```

Parameters

key

The associated key used to pause the music file.

Discussion

Pausing does not reset the music track.

Declared In

F2DAudio.h

playMusicWithKey:repeatCount:volume:

Plays a loaded music file using its associated key.

```
+ (void)playMusicWithKey:(NSString *)key repeatCount:(NSInteger)repeatCount
volume:(CGFloat)volume
```

key

The associated key used to play the music file.

repeatCount

The amount of times the music file will loop. Only when it stops looping does the music file officially end and will notify the Frost2D delegate through the audioPlayerDidFinishPlaying method. To loop a music file forever, pass the constant F2DRepeatAlways or a negative number.

volume

The volume level of the music ranging from 0 to INFINITY. 0 being no volume, 1 being the volume of the original music file.

Discussion

This is the only method for playing music.

See Also

`Frost2D for information regarding loading music files.`

Declared In

`F2DAudio.h`

playSoundWithKey:gain:pitch:

Plays a loaded sound file using its associated key.

```
+ (void)playSoundWithKey:(NSString *)key gain:(ALfloat)gain pitch:
(ALfloat)pitch
```

Parameters

key

The associated key used to play the sound file.

gain

The volume of the sound ranging from 0 to 1. 0 being no volume and 1 being the original volume of the sound.

pitch

The pitch of the sound. Value cannot be 0. Pass 1 to keep the pitch the same as the original sound file.

Discussion

This is the only method for playing sound. Below is an example of loading and playing a sound file.

```
[frost2D loadSoundFile:@"soundFile" withKey:@"myKey"];
[F2DAudio playSoundWithKey:@"myKey" gain:1 pitch:1];
```

See Also

`Frost2D for information regarding loading sound files.`

Declared In

`F2DAudio.h`

stopMusicWithKey:

```
+ (void)stopMusicWithKey:(NSString *)key
```

Parameters

key

The associated key used to stop the music file.

Discussion

Stopping does reset the music track.

Declared In

`F2DAudio.h`

Generated by appledoc 2.1 (build 858).

F2DBody Class Reference

Inherits from	UIImageView
Declared in	F2DBody.h

Overview

The primary physics object.

The F2DBody class is solved, integrated and drawn by Frost2D based on its set properties and type. An F2DBody can be one of three types. Its type is set using the constants below.

- F2DBodyTypeCircle: This type will make the F2DBody a circle. All contacts with this type will use this object's contactRadius property. Collisions with this type can be static or dynamic.
- F2DBodyTypePolyline: This type will make the F2DBody a polyline. All contacts with this type will use this object's lineSegments property. Collisions with this type can only be static.
- F2DBodyTypeFluid: This type will make the F2DBody a fluid. All contacts with this type will use this object's size property. Collisions with this type can only be static.

Note that the F2DBodyTypeCircle is the only dynamic object. The other body types are just "environments" for the F2DBodyTypeCircle to react in.

The F2DBody class is primarily responsible for all physics (with the exception of constraints) as well as solving gesture recognizers and the device accelerometer. The more properties that are configured on the F2DBody, the more overhead your App will encounter, especially when contactEnabled is set to YES. So be careful what physics each F2DBody needs and optimize how it should function in relation to other F2DBody objects. Also note that every F2DBody has a size property which determines the bounds and frame of the F2DBody; the contactRadius property is derived from the size property, but can also be customized (e.g. the contact radius can be set larger than the size property). Every F2DBody also has a centroid property and rotation property. Other properties such as mass can also be set to further customize the F2DBody.

Because the F2DBody comes from the UIKit, it can actually be used as an IBOutlet and placed on an F2DWorld all through interface builder. Runtime attributes can be set to allow for customizing F2DBody properties in interface builder. Note that when subclassing an F2DBody, if created in interface builder, you will need to override the awakeFromNib method and message the super class, but if the F2DBody is made programmatically, you must override the initWithType method and message the super class.

Below is an example of programmatically initializing an F2DBody and adding it to the F2DWorld.

```
F2DBody* someBody = [[F2DBody alloc]initWithType:F2DBodyTypeCircle centroid:F2DVectorMake(1024/2, 768/2) size:F2DVectorMake(200, 200)];
[world addBody:someBody];
```

Tasks

Other

`type` *property*

`frost2D` *property*

Material

`restitution` *property*

`friction` *property*

`mass` *property*

`density` *property*

`viscosity` *property*

`sleep` *property*

Dynamics

`worldPhysics` *property*

`isStatic` *property*

`isOnSurface` *property*

`lineSegments` *property*

Electrostatics

`electrostaticsEnabled` *property*

`fieldIntensity` *property*

`fieldRange` *property*

`intensityLimits` *property*

`resistivity` *property*

`electricCharge` *property*

Point Motion

`pointMotionEnabled` *property*

- travelPoint *property*
- travelSpeed *property*
- travelDistance *property*

Centripetal Motion

- centripetalMotionEnabled *property*
- centripetalPoint *property*
- centripetalRadius *property*
- period *property*
- angularDistance *property*

Contacts/Collisions

- contactFilterGroup *property*
- contactFilterLayer *property*
- contactEnabled *property*
- collisionEnabled *property*
- identifier *property*
- contactFilterList *property*
- contactFilterIdentifier *property*

Callbacks

- cbCentripetalMotion *property*
- cbIntegration *property*
- cbContact *property*
- cbBarrier *property*
- cbPointMotion *property*
- cbConstraint *property*
- – constraintCallback
- – centripetalMotionCallback
- – pointMotionCallback
- – integrationCallback
- – contactCallback:
- – barrierCallback

Angular Velocity/Rotation

- angularVelocity *property*
- simulateAngularVelocity *property*
- rotation *property*
- rotationalRange *property*
- rotationalDamping *property*
- snappingSpeed *property*

Accelerometer

- accelerometerEnabled *property*
- sensitivity *property*
- inverse *property*
- calibration *property*
- accelerometerLimits *property*
- lag *property*
- detectionLimits *property*

Kinematics

- centroid *property*
- contactRadius *property*
- lastCentroid *property*
- size *property*
- velocity *property*
- speed *property*
- acceleration *property*
- direction *property*
- speedLimits *property*
- barrier *property*
- force *property*
- impulse *property*

Gestures

- panGestureEnabled *property*
- panGestureOptions *property*
- panGestureDamping *property*
- pinchGestureEnabled *property*

pinchGestureSizeLimits *property*

rotationGestureEnabled *property*

panGesture *property*

pinchGesture *property*

rotationGesture *property*

– didPan:

– didPinch:

– didRotation:

Constructors

– initWithType:centroid:size:

Properties

acceleration

The amount of velocity an F2DBody will gain or lose per second.

`@property (nonatomic, assign) f2dVector acceleration`

Discussion

This property can be used to accelerate objects in certain directions. Note that unlike velocity, the acceleration does not constantly change to reflect the acceleration of the F2DBody. This value is strictly yours to set and will never change (because Frost2D uses impulsevelocity integrations) unless the force property is set. Also note that this property has nothing to do with the UIAccelerometer or the UIAcceleration properties. Acceleration results in velocity.

Declared In

`F2DBody.h`

accelerometerEnabled

Whether or not an F2DBody will be moved by the accelerometer.

`@property (nonatomic, assign) BOOL accelerometerEnabled`

Discussion

When enabled, the F2DBody will read the output of the device accelerometer and accelerate the F2DBody based on the acceleration values of the device. For the F2Body to be moved by the accelerometer, this property must be enabled, and the device accelerometer must be running.

See Also

`Frost2D for information regarding the accelerometer.`

Declared In

`F2DBody.h`

accelerometerLimits

The max and min velocity an F2DBody can experience from the device accelerometer.

`@property (nonatomic, assign) f2dVectorLimits accelerometerLimits`

Discussion

This value can be used to limit the velocity components derived from the accelerometer. However, it will not limit the velocity property or net velocity of the F2DBody, that is what the speedLimits property is for. By default there are no limits.

Declared In

`F2DBody.h`

angularDistance

The distance traveled around the circular path.

`@property (nonatomic, assign) CGFloat angularDistance`

Discussion

This value is in radians and represents the distance traveled around the arc of the circular motion path. Upon one complete revolution (2PI) the angular distance is reset to 0. This value can be changed to alter the position of the F2DBody on the motion path.

> **Warning:** *Domain:* [0,2PI]. 0 being no distance around the centripetal motion path.

Declared In

`F2DBody.h`

angularVelocity

The rate of change of a rotating F2DBody.

Discussion

This value is in radians and represents the rotational change of the F2DBody per second (e.g. 4PI would be 2 complete rotations per second). Positive values represent clockwise rotation, negative values represent counterclockwise rotation. Angular velocity directly changes the rotation property of an F2DBody. Angular velocity can be simulated through the simulateAngularVelocity property.

Declared In

`F2DBody.h`

axis

Whether or not an F2DBody can travel in the x or y direction through point motion.

```
@property (nonatomic, assign) f2dVectorOptions axis
```

Discussion

By limiting the axis an F2DBody can travel in to get to a point, the point motion velocity component for that F2DBody is not calculated. Note that because this property works on a component level, F2DBody objects may appear to decelerate while reaching their destination. To prevent this, simply set the travel point on the same axis as the F2DBody.

Declared In

`F2DBody.h`

barrier

A rectangular barrier surrounding the F2DBody which prevents further motion of the F2DBody.

```
@property (nonatomic, assign) f2dVectorLimits barrier
```

Discussion

The barrier can react to physics just like an F2DBodyTypePolyline can. It will follow the laws of restitution and friction etc. The barrier is independent of other F2DBody objects; every F2DBody has its own barrier. When the F2DBody comes in contact with the barrier, the barrierCallback method can be messaged if cbBarrier is enabled. The barriers values represent max and min coordinate values. By default there are no limits. The following code would not allow an F2DBody to leave the standard screen space of an iPhone 5 view.

```
someBody.barrier = F2DVectorLimitsMake(568, 0, 320, 0);
```

Declared In

`F2DBody.h`

calibration

A decimal value representing the acceleration offset of the device accelerometer.

```
@property (nonatomic, assign) f2dVector calibration
```

Discussion

By adjusting either the x or y component, the raw UIAcceleration values from the device accelerometer will be offset.

> **Warning:** *Domain:* [-1,1].

See Also

`UIAcceleration for more information.`

Declared In

`F2DBody.h`

cbBarrier

Whether or not an F2DBody can have the barrierCallback messaged.

```
@property (nonatomic, assign) BOOL cbBarrier
```

Discussion

The callback will get messaged after the F2DBody comes in contact with its barrier.

Declared In

`F2DBody.h`

cbCentripetalMotion

Whether or not an F2DBody can have the centripetalMotionCallback messaged.

```
@property (nonatomic, assign) BOOL cbCentripetalMotion
```

Discussion

The callback will get messaged when an F2DBody makes one complete revolution around a centripetalPoint.

Declared In

`F2DBody.h`

cbConstraint

Whether or not an F2DBody can have the constraintCallback messaged.

```
@property (nonatomic, assign) BOOL cbConstraint
```

Discussion

The callback will get messaged when the constraint an F2DBody is attached to reaches its fracture value. When a constraint fractures, the constraint is automatically removed from the F2DWorld.

See Also

`F2DConstraint for more information regarding constraint physics.`

Declared In

`F2DBody.h`

cbContact

Whether or not an F2DBody can have the contactCallback: messaged.

```
@property (nonatomic, assign) BOOL cbContact
```

Discussion

The callback will get messaged after each contact between two F2DBody objects. The F2DBody that the main F2DBody collides with gets passed as an argument.

Declared In

`F2DBody.h`

cbIntegration

Whether or not an F2DBody can have the integrationCallback messaged.

```
@property (nonatomic, assign) BOOL cbIntegration
```

Discussion

The callback will get messaged after each integration of the F2DBody. Should be used as the “game loop.”

See Also

`Frost2D for information regarding the integrator.`

Declared In

`F2DBody.h`

cbPointMotion

Whether or not an F2DBody can have the pointMotionCallback messaged.

```
@property (nonatomic, assign) BOOL cbPointMotion
```

Discussion

The callback will get messaged when the F2DBody comes within range of the travelDistance during point motion.

Declared In

`F2DBody.h`

centripetalMotionEnabled

Whether or not an F2DBody can react to centripetal motion.

```
@property (nonatomic, assign) BOOL centripetalMotionEnabled
```

Discussion

Centripetal motion allows an F2DBody to move in a circular motion around a set point based on a set radius and period.

Declared In

`F2DBody.h`

centripetalPoint

The point of which the F2DBody revolves around.

```
@property (nonatomic, assign) f2dVector centripetalPoint
```

Discussion

Note that when changing the centripetalPoint property, the position of the F2DBody is immediately affected. The centripetal point can constantly be changed to revolve around a moving point.

Declared In

`F2DBody.h`

The distance from the centripetalPoint that the F2DBody will keep when revolving around a point.

```
@property (nonatomic, assign) f2dVector centripetalRadius
```

Discussion

This value is a vector representing both the horizontal and vertical radius of the motion path, allowing for a variety of ellipses, or even a straight line.

> **Warning:** *Domain:* [0,INFINITY]. 0 being no distance, so the centripetal motion will never occur.

Declared In

`F2DBody.h`

centroid

The center position of an F2DBody, representing both the x and y coordinates.

```
@property (nonatomic, assign) f2dVector centroid
```

Discussion

This position is based off of the F2DWorld coordinate system. This property is derived from the size property of the F2DBody. This is a value that must be set upon the initialization of an F2DBody. Note that this property should be used instead of the center property. If created through interface builder, its current center becomes its centroid.

Declared In

`F2DBody.h`

collisionEnabled

Whether or not an F2DBody can react to a contact.

```
@property (nonatomic, assign) BOOL collisionEnabled
```

Discussion

Collisions occur when a contact occurs. The following are the three cases for collisions.

- Circle-Circle
- Circle-Fluid
- Circle-Polyline

Note that reactions with the barrier is not a contact and thus is not a collision. Note that contacts and collisions are different than other physics reactions such as electrostatics.

Declared In

`F2DBody.h`

contactEnabled

Whether or not an F2DBody will check for contacts with other F2DBody objects.

```
@property (nonatomic, assign) BOOL contactEnabled
```

Discussion

This is the primary indicator for all F2DBody contacts. This must be enabled for contacts between all F2DBodyTypes. This property is independent of the actual collision (reaction), but it is the source for all reactions. This property should only be enabled when contacts, and by extension collisions, are necessary, as high overhead will occur.

Declared In

`F2DBody.h`

contactFilterGroup

The contact group an F2DBody belongs to.

```
@property (nonatomic, assign) NSInteger contactFilterGroup
```

Discussion

Only F2DBody objects in different groups can have contacts (e.g. an object in group 1 cannot contact with other objects also in group 1). However, an object in group 0/F2DFilterGroupNone, will contact with every F2DBody, regardless of what group it is in.

Declared In

`F2DBody.h`

contactFilterIdentifier

Used for identifying an F2DBody to be filtered during contacts by other F2DBody objects.

```
@property (nonatomic, assign) NSInteger contactFilterIdentifier
```

Note that this value has nothing to do with the identifier property. This value is only used by the contactFilterList to filter out unwanted contacts, and by extension, unwanted collisions.

Declared In
`F2DBody.h`

contactFilterLayer

The contact layer an F2DBody belongs to.

```
@property (nonatomic, assign) NSInteger contactFilterLayer
```

Discussion
Only F2DBody objects in the same layer can collide with each other, regardless of what groups they are in.

Declared In
`F2DBody.h`

contactFilterList

The list of contactFilterIdentifier values that determine who an F2DBody shouldn't contact with.

```
@property (nonatomic, copy) NSArray *contactFilterList
```

Discussion
By adding a contactFilterIdentifier to this array, the F2DBody will not contact with any F2DBody that shares a contactFilterIdentifier in that list. has However it is recommended to use contactFilterGroup and contactFilterLayer properties as they will provide much less overhead.

Declared In
`F2DBody.h`

contactRadius

The radius of the F2DBody used during contacts.

```
@property (nonatomic, assign) CGFloat contactRadius
```

Discussion
This property represents the mathematical radius around the F2DBody centroid. It is used in all contacts and collisions and is only used with the F2DBodyTypeCircle. Note that by default, this property is derived from the size of the F2DBody by adding the width and height and dividing by 4. Each time the size property of the F2DBody is set, the contactRadius will once again be derived. Note that the contactRadius can be set to other values and should be to accurately match the artwork on the F2DBody, just remember that if you change the size of the F2DBody at all, its contact radius will be derived again.

> **Warning:** *Domain:* (0,INFINITY]. 0 is not a possible value because the F2DBody must always retain a size.

Declared In
`F2DBody.h`

density

The buoyancy factor for an F2DBody.

```
@property (nonatomic, assign) CGFloat density
```

Discussion
In circle-fluid collisions, the buoyancy of the F2DBodyTypeCircle is derived from the ratio of the density of the fluid to the density of the circle. If the circle object's density is greater than the fluid object's density, the circle object will sink. If it is less than, it will float and if it is equal it will have a net buoyancy force of 0. All density defaults to 1.

> **Warning:** *Domain:* (0,INFINITY].

Declared In
`F2DBody.h`

detectionLimits

The range of speeds that the device accelerometer must produce to accelerate the F2DBody.

```
@property (nonatomic, assign) f2dVectorLimits detectionLimits
```

Discussion
If the accelerometer produces a speed not within this range, the acceleration will be ignored and thus no motion will occur. Note that this value uses the speed property. By default there are no limits.

The following example code would only make the accelerometer accelerate the F2DBody if it produces a horizontal speed of at least 100 px/s, no other limitations are applied.

```
someBody.detectionLimits = F2DVectorLimitsMake(F2D_HUGE, 100, F2D_HUGE, 0);
```

Declared In

`F2DBody.h`

direction

The angle that the F2DBody is traveling at in radians.

```
@property (nonatomic, assign, readonly) CGFloat direction
```

Discussion

The angle value is relative to the F2DWorld coordinate system. This value is useful in determining what direction the F2DBody is exactly moving in. Note that 2PI is used instead of 0 radians for motion in the right direction. Also note that this value may be slightly off due to floating point precision. If the F2DBody is not moving in any direction, then -1/F2DDirectionNone is returned.

Declared In

`F2DBody.h`

electricCharge

The kind of charge the F2DBody has.

```
@property (nonatomic, assign) NSInteger electricCharge
```

Discussion

Depending on the charge, F2DBody objects within each other's electric fields will experience a repulsion, attraction or neutral reaction. The charge is set by using one of the F2DBodyCharge constants. Below is a list and description of each charge constant.

- F2DBodyChargePositive: F2DBody will repel other positives, and attract other negatives.
- F2DBodyChargeNegative: F2DBody will repel other negatives, and attract other positives.
- F2DBodyChargeNeutral: F2DBody will neither repel nor attract positives or negatives.
- F2DBodyChargeOpposite: F2DBody will attract all objects, regardless of charge (e.g. gravitational fields).
- F2DBodyChargeLike: F2DBody will repel all objects, regardless of charge.

See Also

`F2DConstants for information regarding the charge constants.`

Declared In

`F2DBody.h`

electrostaticsEnabled

Whether or not an F2DBody can react to electrostatics.

```
@property (nonatomic, assign) BOOL electrostaticsEnabled
```

Discussion

Electrostatics can be used for both attraction and repulsion, depending on the electricCharge property. Electrostatics allow for electric field attraction or repulsions around a set radius. Electrostatics can be useful for simulating "electric contacts."

Declared In

`F2DBody.h`

fieldIntensity

The strength of the electric field surrounding the F2DBody.

```
@property (nonatomic, assign) CGFloat fieldIntensity
```

Discussion

This value, combined with the resistivity of the other F2DBody and the fieldRange, determines the net electric force that an F2DBody will experience when under the influence of an electric field. The closer the F2DBody objects are to each other, the greater the electric force.

> **Warning:** *Domain:* [0,INFINITY]. 0 being no strength of the electric field.

Declared In

`F2DBody.h`

fieldRange

The range of the electric field.

```
@property (nonatomic, assign) f2dVectorRange fieldRange
```

Discussion

F2DBody objects outside the electric field range will not experience any electric force. The distance from the center of the field is exponentially related to the force

> **Warning:** *Domain:* [0,INFINITY]. 0 being no distance between the center of the field and its reacting F2DBody.

Declared In
`F2DBody.h`

force

Applies a force on the F2DBody, directly changing its acceleration.

```
@property (nonatomic, assign) f2dVector force
```

Discussion
The acceleration of the F2DBody will change based on the mass of the F2DBody. Note that the acceleration property of an F2DBody will not be distorted through changing motion (the set force is permanent), but the resultant velocity will. Force motion is gradual.

Declared In
`F2DBody.h`

friction

The resistance of the F2DBody while sliding against the barrier or polylines.

```
@property (nonatomic, assign) CGFloat friction
```

Discussion
If friction is desired, it is recommended to have a value great enough to prevent negligible movement. Friction is exponential.

> **Warning:** *Domain:* [0,INFINITY]. 0 being no friction.

See Also
`@property sleep for information regarding negligible motion.`

Declared In
`F2DBody.h`

frost2D

A reference to Frost2D.

```
@property (nonatomic, weak) Frost2D *frost2D
```

Discussion
Note that this reference to Frost2D is already shared. Simply access this property to quickly access Frost2D, typically from a subclassed F2DBody.

Declared In
`F2DBody.h`

identifier

A way to identify or name an F2DBody.

```
@property (nonatomic, assign) NSInteger identifier
```

Discussion
This property is useful when retrieving contacts between F2DBody objects using the contactCallback:. It can help differentiate F2DBody objects of the same class. Below is an example of using the identifier property in a contact callback.

```
-(void)contactCallback:(F2DBody *)otherBody
{
if (otherBody.identifier==kBallIdentifier)
{
//Contact occurred
}
}
```

Declared In
`F2DBody.h`

impulse

Applies an impulse on the F2DBody, directly changing its velocity.

```
@property (nonatomic, assign) f2dVector impulse
```

Discussion
The velocity of the F2DBody will change based on the mass of the F2DBody. This property has no effect on the F2DBody acceleration property. Impulse motion is instantaneous.

F2DBody.h

intensityLimits

The limits of the velocity obtained by an F2DBody while inside an electric field.

```
@property (nonatomic, assign) f2dVectorLimits intensityLimits
```

Discussion
The limits apply to the components of the velocity derived from the electric force experienced while inside an electric field, and thus represent both negative and positive values. By default there are no limits.

Declared In
F2DBody.h

inverse

Whether or not the acceleration by the accelerometer will be inverted.

```
@property (nonatomic, assign) f2dVectorOptions inverse
```

Discussion
This value will reverse the direction of the accelerometer for the F2DBody. This value will need to be adjusted based on the orientation of the device (e.g. landscape-right, portrait, landscape-left, etc).

Declared In
F2DBody.h

isOnSurface

Whether or not an F2DBody is on a barrier or on an F2DBodyTypePolyline object.

```
@property (nonatomic, assign, readonly) BOOL isOnSurface
```

Discussion
This, in combination with other factors like velocity, can be useful in determining whether or not an F2DBody can execute certain motion behavior, such as a "jump."

Declared In
F2DBody.h

isStatic

Whether or not an F2DBody will be integrated.

```
@property (nonatomic, assign) BOOL isStatic
```

Discussion
While set to YES, the F2DBody will not move. This value should be set to YES when an F2DBody is not expected to move. All polyline and fluid body types have their isStatic property set to YES by default.

Declared In
F2DBody.h

lag

The time it takes the accelerometer to apply the derived velocity to the F2DBody. Decrease to make the F2DBody more responsive.

```
@property (nonatomic, assign) CGFloat lag
```

Discussion
This value controls how responsive the F2DBody is to the accelerometer. By default this value is 1, so it would take 1 second for the F2DBody to completely apply the acceleration from the accelerometer. Increasing this value would produce more "lag" and would thus make the F2DBody less responsive to the accelerometer, but its motion would become more dynamic (there would be remove for distortions in the motion of the F2DBody). The smallest this value can be is the time step of the engine/F2DDeltaTime, as that would produce complete responsiveness but no room for motion distortions.

> **Warning:** *Domain:* [F2DDeltaTime,INFINITY].

Declared In
F2DBody.h

lastCentroid

The last centroid of an F2DBody.

```
@property (nonatomic, assign, readonly) f2dVector lastCentroid
```

Discussion

Declared In
F2DBody.h

lineSegments

The line segments that an F2DBody will place, based on its size and rotation. Gives shape to an F2DBody.

```
@property (nonatomic, copy) NSArray *lineSegments
```

Discussion

Can be used to give F2DBody objects of the polyline type a mathematical shape other than a circle. Passing an array of F2DVectorLines is used to place the line segments on the F2DBody. The lines do not need to form a closed figure (hence the polyline type). The line segment positions, when placed on an F2DBody, do not represent world coordinates, but rather relative coordinates to the size and rotation of the F2DBody. The coordinates range from 0 to 1 in both the x and y direction, such that line position 0,0 is at the top left and 1,1 is at the bottom right. Note that the segments, when placed on an F2DBody, will automatically be scaled and rotated as the F2DBody is scaled and rotated, making shapes extremely simplified and easy to work with. The following code would make an F2DBody an equilateral triangle.

```
f2dVectorLine line1= F2DVectorLineMake( F2DVectorMake(0,1) , F2DVectorMake(.5 ,0) ,1);
f2dVectorLine line2= F2DVectorLineMake( F2DVectorMake(.5,0) , F2DVectorMake(1 ,1) ,1);
f2dVectorLine line3= F2DVectorLineMake( F2DVectorMake(1,1) , F2DVectorMake(0 ,1) ,1);
NSArray *array = [NSArray arrayWithObjects:
F2DEncode(line1),
F2DEncode(line2),
F2DEncode(line3),
nil];
someBody.lineSegments=array;
```

Note that when placing lines, left to right is considered north and right to left is considered south. The line's face always goes in the direction of its normal. Note that because the lineSegments property requires an F2DBody to be a polyline type, there are no dynamic collisions between polyline-polyline contacts. See the overview for more information.

There is also a shape method for easily making template shapes. The following method would make that same equilateral triangle.

```
someBody.lineSegments=F2DShapeBody(F2DBodyShapeTriangle, 1);
```

Note that these line segments do not actually draw a line on the screen. They are mathematical representations that artwork should "fit" over to simulate reactions. There is some sample artwork included with the Frost2D Framework. F2DConstraints can be used to draw static lines on the screen, but they will not scale nor rotate with the F2DBody. To draw dynamically the lines on an F2DBody, the draw rect in the F2DWorld can be overridden to calculate and draw the lines by uncommenting the F2D_DRAW_STATE_WORLD preprocessor definition, however it is highly recommended to use artwork to represent these mathematical lines rather than use the draw rect to prevent drastic overhead.

See Also

F2DFunctions for information regarding the f2dVectorLine and its parameters.

F2DConstraint for information regarding drawing static lines on the screen.

F2DConstants for information regarding the draw constants.

F2DWorld for information regarding the preprocessor drawRect definitions.

Declared In
F2DBody.h

mass

The inertia of an F2DBody.

```
@property (nonatomic, assign) CGFloat mass
```

Discussion

The greater the mass of the object, the more resistance it has to external forces and impulses, resulting in a smaller velocity and acceleration. It is recommended to keep the mass of an object at 1, as that value completely prevents object overlap during collisions. Any value greater than 1 may result in some overlap, depending on the circumstance of the collision. If the mass of an object is great enough, it will be immovable, unless another object of great enough mass collides with it. In the Frost2D Framework, mass has no relation to the size of the F2Body nor its density, thus providing more customization.

Warning: *Domain:* [1,INFINITY]. 1 being no tendency to resist motion and complete overlap prevention during collisions.

Declared In
F2DBody.h

panGesture

The pan gesture recognizer on the F2DBody.

```
@property (nonatomic, strong) UIPanGestureRecognizer *panGesture
```

Discussion

This gesture recognizer can be enabled to allow for its gesture to operate. This gesture is initialized by the F2DBody, it only needs to be enabled or disabled through the panGestureEnabled property.

Declared In

panGestureDamping

A factor for reducing the effect panning has on the velocity of the F2DBody. Increase to reduce responsiveness.

```
@property (nonatomic, assign) f2dVector panGestureDamping
```

Discussion

This value can be useful in reducing the velocity obtained by the F2DBody during panning.

> **Warning:** *Domain:* [0,INFINITY].

Declared In

`F2DBody.h`

panGestureEnabled

Whether or not an F2DBody can be panned (dragged by touch input) by the UIPanGestureRecognizer.

```
@property (nonatomic, assign) BOOL panGestureEnabled
```

Discussion

The pan gesture recognizer on the F2DBody is solved in a dynamic way. By panning the F2DBody through touch input, the F2DBody will obtain a velocity both during and after the touch input comes to an end. This property also allows for multitouch (e.g. multiple F2DBody objects can be panned at the same time). Note that the touchesBegan property on an F2DBody attempts to get messaged to stop all motion on the F2DBody when touch input begins. This provides more responsiveness to touch input when a user touches an F2DBody but does not pan it. Note that user interaction for the F2DBody must be enabled for gesture recognizers and touch input to work. Also note that gestures recognizers are extremely customizable (e.g. support for simultaneous gestures, add additional gestures etc).

See Also

`UIPanGestureRecognizer for more information.`

Declared In

`F2DBody.h`

panGestureOptions

Options for whether or not an F2DBody can be panned in the x or y directions.

```
@property (nonatomic, assign) f2dVectorOptions panGestureOptions
```

Discussion

By default both x and y panning is set to YES.

Declared In

`F2DBody.h`

period

The time it takes to complete on revolution around a centripetal point, in seconds.

```
@property (nonatomic, assign) CGFloat period
```

Discussion

The time it takes to revolve is independent of the centripetalRadius property. The speed of the F2DBody revolving is constant and is directly dependent on the period. When an F2DBody is undergoing centripetal motion, all external forces are ignored and the F2DBody becomes “locked” in a motion path until disabled. Note that sleep and speedLimits override this behavior.

> **Warning:** *Domain:* (0,INFINITY].

Declared In

`F2DBody.h`

pinchGesture

The pinch gesture recognizer on the F2DBody.

```
@property (nonatomic, strong) UIPinchGestureRecognizer *pinchGesture
```

Discussion

This gesture recognizer can be enabled to allow for its gesture to operate. This gesture is initialized by the F2DBody, it only needs to be enabled or disabled through the pinchGestureEnabled property.

Declared In

`F2DBody.h`

pinchGestureEnabled

```
@property (nonatomic, assign) BOOL pinchGestureEnabled
```

Discussion

The pinch gesture recognizer on the F2DBody is just static, it simply scales the size of the F2DBody; it does support multitouch (e.g. multiple F2DBody objects can be pinched at the same time). Note that user interaction for the F2DBody must be enabled for gesture recognizers and touch input to work. Also note that gestures recognizers are extremely customizable (e.g. support for simultaneous gestures, add additional gestures etc).

See Also

`UIPinchGestureRecognizer for more information.`

Declared In

`F2DBody.h`

pinchGestureOptions

Options for whether or not an F2DBody can be pinched in the x or y directions.

```
@property (nonatomic, assign) f2dVectorOptions pinchGestureOptions
```

Discussion

By default both x and y pinching is set to YES.

> **Warning:** *Domain:* (0,INFINITY].

Declared In

`F2DBody.h`

pinchGestureSizeLimits

Limits the size the F2DBody can obtain when pinched.

```
@property (nonatomic, assign) f2dVectorLimits pinchGestureSizeLimits
```

Discussion

When an F2DBody is pinched, its x and/or y components of its size will be scaled. This value places limits on the size of the F2DBody during pinching. By default there are no limits.

> **Warning:** *Domain:* (0,INFINITY].

Declared In

`F2DBody.h`

pointMotionEnabled

Whether or not an F2DBody can react to point motion.

```
@property (nonatomic, assign) BOOL pointMotionEnabled
```

Discussion

Point motion allows an F2DBody to move to a desired coordinate using a desired speed and axis of travel. A callback can be messaged when the F2DBody reaches the desired point a certain distance away.

Declared In

`F2DBody.h`

resistivity

The resistance an F2DBody has to the fieldIntensity property while inside an electric field.

```
@property (nonatomic, assign) CGFloat resistivity
```

Discussion

The greater this value, the less the net electric force will act on the F2DBody.

> **Warning:** *Domain:* [0,INFINITY]. 0 being no resistance to electric field strength.

Declared In

`F2DBody.h`

restitution

The amount of velocity (energy) retained after impacting a surface. The elasticity of an object.

```
@property (nonatomic, assign) CGFloat restitution
```

Discussion

This property can be used to simulate the “bounciness” of an F2DBody. It is calculated in all collisions with polylines and in collisions with the barrier. Note that the

Warning: *Domain:* [0,1]. 0 being completely inelastic (no velocity retained) and 1 being completely elastic (all velocity retained).

Declared In
`F2DBody.h`

rotation

The current rotation of the F2DBody.

```
@property (nonatomic, assign) CGFloat rotation
```

Discussion
Rotation represents the current relative angular displacement from the F2DBody object's original angular position in radians. Note that because rotation is always converted to a value within the −2PI to 2PI range, any value greater or less than this value will be converted. By setting the rotation of an F2DBody, the F2DBody will immediately rotate to that position (e.g. setting the rotation to PI/2 or 90 degrees will immediately make the F2DBody rotated 90 degrees). Positive values represent clockwise rotation, while negative values represent counterclockwise rotation.

Warning: *Domain:* [−2PI,2PI].

Declared In
`F2DBody.h`

rotationGesture

The rotation gesture recognizer on the F2DBody.

```
@property (nonatomic, strong) UIRotationGestureRecognizer *rotationGesture
```

Discussion
This gesture recognizer can be enabled to allow for its gesture to operate. This gesture is initialized by the F2DBody, it only needs to be enabled or disabled through the rotationGestureEnabled property.

Declared In
`F2DBody.h`

rotationGestureEnabled

Whether or not an F2DBody can be rotated (rotated by touch input) by the UIRotationGestureRecognizer.

```
@property (nonatomic, assign) BOOL rotationGestureEnabled
```

Discussion
The rotation gesture recognizer on the F2DBody is solved in a dynamic way. By rotating the F2DBody through touch input, the F2DBody will obtain an angular velocity both during and after the touch input comes to an end. This property also allows for multitouch (e.g. multiple F2DBody objects can be rotated at the same time). Note that the touchesBegan property on an F2DBody attempts to get messaged to stop all motion on the F2DBody when touch input begins. This provides more responsiveness to touch input when a user touches an F2DBody but does not rotate it. Note that user interaction for the F2DBody must be enabled for gesture recognizers and touch input to work. Also note that gestures recognizers are extremely customizable (e.g. support for simultaneous gestures, add additional gestures etc).

See Also
`UIRotationGestureRecognizer for more information.`

Declared In
`F2DBody.h`

rotationalDamping

A factor for reducing the effect angular velocity has on the rotation of an F2DBody.

```
@property (nonatomic, assign) CGFloat rotationalDamping
```

Discussion
By increasing this value, angular velocity will no longer have a direct effect on rotation. Increasing this value will make F2DBody objets rotate slower than they normally would given a certain velocity. This value is useful in more accurately matching the artwork of an F2DBody to its angular velocity. For example, an F2DBody with a square shaped image would normally not rotate at the same angular velocity as a circle would. This value, combined with the snappingSpeed and rotationalRange property can provide more realism to artwork that may not math the default properties of an F2DBodyType circle.

Warning: *Domain:* [0,INFINITY]. 0 being no damping.

Declared In
`F2DBody.h`

rotationalRange

The range an F2DBody can rotate in.

```
@property (nonatomic, assign) f2dVectorRange rotationalRange
```

By adjusting this value, the relative rotation of an F2DBody can be limited (for example, a range from –PI/2 to PI/2 would only allow an F2DBody to be rotated 90 degrees both right and left of its initial angular position. All rotations, even through the simulateAngularVelocity property will conform to this range. By default, rotationalRange.max is 2PI and rotationalRange.min is –2PI. Note that because rotation is always converted to values within the –2PI to 2PI range, making the range any greater would have no effect.

> **Warning:** *Domain:* [–2PI,2PI].

Declared In
`F2DBody.h`

sensitivity

A multiplier for the acceleration of the F2DBody by the device accelerometer.

```
@property (nonatomic, assign) f2dVector sensitivity
```

Discussion

This value increases the motion experienced by the F2DBody. This value is a vector representing the sensitivity in both the x and y directions. Increasing it will result in a more sensitive acceleration by the device accelerometer. This value must be greater than 0 for the F2DBody to move. Defaults to 0.

> **Warning:** *Domain:* [0,INFINITY]. 0 being no sensitivity and thus no motion.

Declared In
`F2DBody.h`

simulateAngularVelocity

Whether or not the angular velocity of an F2DBody will dynamically react to other F2Dbody objects and the barrier.

```
@property (nonatomic, assign) BOOL simulateAngularVelocity
```

Discussion

By simulating the angular velocity, the F2DBody will rotate based on collisions with other objects. For example, if this property is enabled, and the F2DBody is "rolling" down an F2DBody polyline type, then the F2DBody will have its angular velocity simulated to accurately reflect the "rolling" motion behavior. If this property is disabled, the F2DBody would not rotate in this case and would appear very static. The following are cases where angular velocity would be simulated.

- While in a fluid.
- When colliding with other F2DBody circle types.
- When colliding with the barrier.
- When colliding with other F2DBody polyline types.

Note that angular velocity can only be simulated on F2DBody circle type objects, however, other F2DBody types can still have angular velocity, just not simulated. All simulated angular velocity uses the x component of its velocity in combination with its contact radius.

Declared In
`F2DBody.h`

size

The size of the F2DBody, representing the width (x) and height (y).

```
@property (nonatomic, assign) f2dVector size
```

Discussion

This property adjusts the outer rectangular bounds of the F2DBody and is used in some physics calculations like fluid, line segment and radius derivations. This property can be used to scale the F2DBody. Note that this property must be used to adjust the size of the F2DBody. Do not access the frame property of an F2DBody as it is frequently undefined due to CGAffineTransforms. This is a value that must be set upon the initialization of an F2DBody. If created through interface builder, its bounds becomes its size.

> **Warning:** *Domain:* (0,INFINITY]. 0 is not a possible value because the F2DBody must always retain a size.

Declared In
`F2DBody.h`

sleep

The speed at which the velocity of an F2Body will be set to 0.

```
@property (nonatomic, assign) CGFloat sleep
```

Discussion

Often, the speed of an F2DBody will reach values extremely close to 0 (e.g. .03). To prevent such negligible values, the physics engine will check if the speed of an object falls below the sleep value. If it does, then the velocity of the F2DBody will be set to 0, thus preventing negligible velocity such in the case of friction. By default the sleep value is set to 1 (e.g. this means that if an F2DBody is traveling at .9 pixels per second its velocity will be set to 0). The sleep value applies to both the x and y speed components, rather than the magnitude. It is recommended, if possible, to increase this value to prevent awkwardly slow motion by F2DBody objects. It will also make it easier to check if an F2DBody is no longer moving. The downside of increasing the sleep value is that in certain cases, like in electrostatics or buoyancy motion, desired small velocity values will not occur because they may fall below the sleep value and thus be set to 0 (e.g. if you set the

Warning: *Domain:* [0,INFINITY]. 0 being no sleep.

Declared In
F2DBody.h

snappingSpeed

The speed at which an F2DBody will snap to the rotation of the surface it is currently on.

```
@property (nonatomic, assign) CGFloat snappingSpeed
```

Discussion

This value will snap, or set the rotation of an F2DBody to match the angle of the surface when at a certain speed. It is used to provide more realism to artwork that may not be curved in shape. For example, artwork with a square shape would not rotate as a circle would, in that at a certain speed it would “snap” to one of its faces. Through this property, F2DBody objects can snap to the angle of the surface (e.g. an F2DBody going down a PI/2 or 90 degrees polyline will have its rotation set to PI/2 when at or below the snappingSpeed). This property, in combination with rotationalRange and rotationalDamping can provide a greater element of realism to artwork that may not “fit” the circle type of an F2DBody. Note that when an F2DBody is “snapped” it is animated over time based on the F2DSnappingAnimationFactor. Also note that all snapping is currently based off of the initial angular position of F2DBody objects.

This is one of the Frost2D Framework’s limitations, as all dynamic F2DBody objects must fit within a “circle” type. However, but adjusting the rotational properties correctly, it is possible to accurately simulate the angular velocity of non-cyclic objects.

Warning: *Domain:* [0,INFINITY]. 0 being no snapping.

Bug: *Development:* More customization for snapping is planned, for example, possibly snapping an F2DBody to custom angles while it is on a surface.

See Also
F2DConstants for information regarding the F2DSnappingAnimationFactor.

Declared In
F2DBody.h

speed

The magnitude of the components of the velocity property.

```
@property (nonatomic, assign) f2dVector speed
```

Discussion

This property represents the absolute value of the velocity components of the F2DBody and will constantly change to reflect the motion of an F2DBody. This property ignores the sign or direction of the F2DBody object’s velocity, thus it is always positive. When setting the speed of an F2DBody, its velocity will remain in its current direction, just the magnitude will change. Speed results in velocity.

Warning: *Domain:* [0,INFINITY]. 0 being no speed.

Declared In
F2DBody.h

speedLimits

Limits the velocity of the F2DBody.

```
@property (nonatomic, assign) f2dVectorLimits speedLimits
```

Discussion

This value uses the speed or magnitude of the velocity components to limit the F2DBody object’s velocity. This value applies to both the positive and negative directions. By default there are no limits. The following would never allow an F2DBody to exceed 100 px/s regardless of direction.

```
someBody.speedLimits = F2DVectorLimitsMake(100,0,100,0);
```

Warning: *Domain:* [0,INFINITY].

Declared In
F2DBody.h

travelDistance

The distance an F2DBody must be from a point to receive the pointMotionCallback.

```
@property (nonatomic, assign) CGFloat travelDistance
```

Discussion

This value allows the F2DBody to receive a callback through the cbPointMotion property when it is within the specified distance from a point. This property is necessary as an F2DBody may never reach the desired point perfectly. To check if the desired point was reached, simply check during each callback to see where the F2DBody object’s centroid is. Note that as long as an F2DBody is within this distance from the travelPoint, the pointMotionCallback method will continuously be messaged.

Declared In
F2DBody.h

travelPoint

The point an F2DBody will attempt to travel to, based on its speed.

```
@property (nonatomic, assign) f2dVector travelPoint
```

Discussion
Note that for an F2DBody to reach its desired location, the speed must be great enough to overcome external forces. The F2DBody will always attempt to reach this point. To “follow” objects, constantly update the travel point.

Declared In
F2DBody.h

travelSpeed

The speed an object will try to maintain when traveling to the travelPoint.

```
@property (nonatomic, assign) CGFloat travelSpeed
```

Discussion
The speed is a scaler quantity in pixels per second.

> **Warning:** *Domain:* Value ranges from 0 to +infinity. 0 being no speed, so the F2DBody would not travel anywhere.

Declared In
F2DBody.h

type

The type of an F2DBody.

```
@property (nonatomic, assign) NSInteger type
```

Discussion
The type determines how an F2DBody will react to other F2DBody objects. Use the constants below to set the F2DBody type.

```
F2DBodyTypeCircle
F2DBodyTypePolyline
F2DBodyTypeFluid
```

All F2DBody objects default to the F2DBodyTypeCircle. See the F2DBody class overview for more information regarding how F2DBodyTypes work. This is a value that must be set upon the initialization of an F2DBody. If created through interface builder, its type defaults to F2DBodyTypeCircle unless a user defined runtime attribute is specified.

See Also
F2DConstants for information regarding the body @property type constants.

Declared In
F2DBody.h

velocity

The pixels per second that an F2DBody will move at, representing the x and y directions.

```
@property (nonatomic, assign) f2dVector velocity
```

Discussion
The velocity property is the exact and definite way of finding and setting the motion of an F2DBody. All motion is through the velocity property. All calculations including forces and impulses and accelerations lead to the velocity property being directly affected. This property is also based off of the F2DWorld coordinate system, and can accept decimal values. Note that negative x values move the F2DBody left and negative y values move the F2DBody up. Note that Frost2D primarily uses impulse-velocity calculations to integrate the F2DBody objects.

Declared In
F2DBody.h

viscosity

The fluid friction of an F2DBody.

```
@property (nonatomic, assign) CGFloat viscosity
```

Discussion
This value represents the internal fluid tension factor of an F2DBodyTypeFluid object. When an F2DBodyTypeCircle object floats in and F2DBodyTypeFluid object, it will face friction.

Declared In
`F2DBody.h`

worldPhysics

Whether or not an F2DBody will be affected by certain physics properties of the F2DWorld, such as gravity.

```
@property (nonatomic, assign) BOOL worldPhysics
```

Discussion
Currently the acceleration due to gravity of the F2DWorld is the only property that depends on this property being enabled. This property is useful for making certain F2DBody objects unaffected by gravity while still allowing them to be dynamic.

Declared In
`F2DBody.h`

Instance Methods

barrierCallback

The callback method messaged when the F2DBody comes in contact with its barrier.

```
- (void)barrierCallback
```

Discussion
Override and set cbBarrier to YES to get messaged. Messaging the super class is not required.

Declared In
`F2DBody.h`

centripetalMotionCallback

The callback method messaged when the F2DBody completes one revolution around a centripetalPoint.

```
- (void)centripetalMotionCallback
```

Discussion
Override and set cbCentripetalMotion to YES to get messaged. Messaging the super class is not required.

Declared In
`F2DBody.h`

constraintCallback

The callback method messaged when a constraint fractures.

```
- (void)constraintCallback
```

Discussion
Override and set cbConstraint to YES to get messaged. Messaging the super class is not required.

Declared In
`F2DBody.h`

contactCallback:

The callback method messaged when the F2DBody comes in contact with another F2DBody.

```
- (void)contactCallback:(F2DBody *)otherBody
```

Parameters
otherBody
The other F2DBody that the primary F2DBody contacts with.

Discussion
Override and set cbContact to YES to get messaged. Messaging the super class is not required.

Declared In
`F2DBody.h`

didPan:

A callback for the UIPanGestureRecognizer that receives a message when a pan occurs on the F2DBody.

```
- (void)didPan:(UIPanGestureRecognizer *)recognizer
```

recognizer

The gesture recognizer that detected the pan.

Discussion

This method can be overridden to customize the way the F2DBody handles this gesture, just remember to message the super class.

Declared In

`F2DBody.h`

didPinch:

A callback for the UIPinchGestureRecognizer that receives a message when a pinch occurs on the F2DBody.

```
- (void)didPinch:(UIPinchGestureRecognizer *)recognizer
```

Parameters

recognizer

The gesture recognizer that detected the pinch.

Discussion

This method can be overridden to customize the way the F2DBody handles this gesture, just remember to message the super class.

Declared In

`F2DBody.h`

didRotation:

A callback for the UIRotationGestureRecognizer that receives a message when a rotation occurs on the F2DBody.

```
- (void)didRotation:(UIRotationGestureRecognizer *)recognizer
```

Parameters

recognizer

The gesture recognizer that detected the rotation.

Discussion

This method can be overridden to customize the way the F2DBody handles this gesture, just remember to message the super class.

Declared In

`F2DBody.h`

initWithType:centroid:size:

The primary and currently only way to properly initialize an F2DBody programmatically.

```
- (id)initWithType:(NSInteger)type centroid:(f2dVector)centroid size:(f2dVector)size
```

Parameters

type

The type of the F2DBody. The type determines how the F2DBody reacts with other objects. See the type property for more information.

centroid

The initial centroid of the F2DBody. The centroid determines the center of the F2DBody in the coordinate system. See the centroid property for more information.

size

The initial size of the F2DBody. The size determines the outer rectangular bounds of the F2DBody. See the size property for more information.

Return Value

An initialized object.

Discussion

This method passes the fundamental properties of the F2DBody as arguments. Note that F2DBody creation through interface builder messages the awakeFromNib method. See the class overview for more information.

Declared In

`F2DBody.h`

integrationCallback

The callback method messaged when the F2DBody is integrated.

```
- (void)integrationCallback
```

Discussion

Override and set cbIntegration to YES to get messaged. Messaging the super class is not required.

Declared In

`F2DBody.h`

The callback method messaged when the F2DBody is within a specified range from the travelPoint.

```
- (void)pointMotionCallback
```

Discussion

Override and set cbPointMotion to YES to get messaged. Messaging the super class is not required.

Declared In

`F2DBody.h`

Generated by appledoc 2.1 (build 858).

F2DConstants Class Reference

Declared in	F2DConstants.h

Overview

All constants, definitions, macros and global variables.

A huge number (one billion) used for extending limits and other types to unreachable values. Can also be used to represent an infinite value such as in the case of mass, however values may need to be rounded.

```
#define F2D_HUGE 1000000000
```

A macro that converts degrees to radians.

```
#define F2D_DEGREES_TO_RADIANS(__ANGLE__) ((__ANGLE__) / 180.0 * M_PI)
```

A macro that converts radians to degrees.

```
#define F2D_RADIANS_TO_DEGREES(__ANGLE__) ((__ANGLE__) / M_PI * 180.0)
```

A preprocessor definition that, once uncommented, will enable the drawRect of the F2DWorld. This definition must be uncommented for any other F2D_DRAW_STATE to take effect. **See Also** F2DWorld for information regarding the drawRect.

```
#define F2D_DRAW_STATE_BASE
```

A preprocessor definition that enables the drawing of F2DBody line segments. **See Also** F2DWorld for information regarding the drawRect. **See Also** F2DBody for information regarding the line segments property.

```
#define F2D_DRAW_STATE_BODY
```

A preprocessor definition that enables the drawing of F2DWorld line segments. **See Also** F2DWorld for information regarding the drawRect and line segments property.

```
#define F2D_DRAW_STATE_WORLD
```

A preprocessor definition that enables the drawing of F2DConstraint joint lines. **See Also** F2DWorld for information regarding the drawRect. **See Also** F2DConstraint for information regarding the joint line.

```
#define F2D_DRAW_STATE_CONSTRAINT
```

The time value associated with the current refresh rate of the display. Used as a time derivative for physics calculations. The time stamp difference of the integrator. The reciprocal of the display fps. Note that this value should only be read, use the F2DFrameInterval constant to set the fps. **See Also** Frost2D for information regarding the integrator.

The update interval associated with the refresh rate of the display. If 1, integrations will occur at the same rate as the display refresh rate (typically around 60 fps). If 2, integrations will occur every other frame (typically resulting in 30 fps). And so forth. **See Also** Frost2D for information regarding the integrator.

```
NSInteger const F2DFrameInterval = 2
```

The update interval of the device accelerometer. **See Also** Frost2D for information regarding the device accelerometer.

```
NSTimeInterval const F2DAccelerometerInterval = 0.1
```

The circle body type for an F2DBody. Dynamic. **See Also** F2DBody for information regarding the type property.

```
NSInteger const F2DBodyTypeCircle = 0
```

The polyline body type for an F2DBody. Static. **See Also** F2DBody for information regarding the type property

```
NSInteger const F2DBodyTypePolyline = 1
```

The fluid body type for an F2DBody. Static. **See Also** F2DBody for information regarding the type property

```
NSInteger const F2DBodyTypeFluid = 2
```

The spring constraint type for an F2DConstraint, used for retaining a joint distance. **See Also** F2DConstraint for information regarding the type property.

```
NSInteger const F2DConstraintTypeSpring = 0
```

The rope constraint type for an F2DConstraint, used for retaining a joint distance only when that distance is exceeded. **See Also** F2DConstraint for information regarding the type property.

```
NSInteger const F2DConstraintTypeRope = 1
```

The cosmetic constraint type for an F2DConstraint. Typically only used for drawing. No physics. **See Also** F2DConstraint for information regarding the type property.

```
NSInteger const F2DConstraintTypeCosmetic = 2
```

The line constraint type for an F2DConstraint. Typically only used for static drawing. No solving. No physics. Static **See Also** F2DConstraint for information regarding the type property.

```
NSInteger const F2DConstraintTypeLine = 3
```

The electric charge for an F2DBody, used in electrostatics. **See Also** F2DBody for information regarding electrostatics.

```
NSInteger const F2DBodyChargePositive = 1
```

regarding electrostatics.

```
NSInteger const F2DBodyChargeNegative = -1
```

The electric charge for an F2DBody, used in electrostatics. **See Also** F2DBody for information regarding electrostatics.

```
NSInteger const F2DBodyChargeNeutral = 0
```

The electric charge for an F2DBody, used in electrostatics. **See Also** F2DBody for information regarding electrostatics.

```
NSInteger const F2DBodyChargeOpposite = 2
```

The electric charge for an F2DBody, used in electrostatics. **See Also** F2DBody for information regarding electrostatics.

```
NSInteger const F2DBodyChargeLike = 3
```

The rectangle shape formed by placing line segments on an F2DBody through the F2DShapeBody function. **See Also** F2DBody for information regarding line segments.

```
NSInteger const F2DBodyShapeRectangle = 0
```

The triangle shape formed by placing line segments on an F2DBody through the F2DShapeBody function. **See Also** F2DBody for information regarding line segments.

```
NSInteger const F2DBodyShapeTriangle = 1
```

The right triangle shape formed by placing line segments on an F2DBody through the F2DShapeBody function. **See Also** F2DBody for information regarding line segments.

```
NSInteger const F2DBodyShapeRightTriangle = 2
```

The octagon shape formed by placing line segments on an F2DBody through the F2DShapeBody function. **See Also** F2DBody for information regarding line segments.

```
NSInteger const F2DBodyShapeOctagon = 3
```

The combined component speed at which the F2DBody will have no restitution. Prevents sporadic motion while F2DBody objects are on polylines. Increase to reduce sporadic motion at the expense of restitution accuracy.

```
NSInteger const F2DRestitutionTerminationFactor = 50
```

The animation time it takes for an F2DBody to snap to its rotation. **See Also** F2DBody for information regarding rotation and angular velocity.

```
CGFloat const F2DSnappingAnimationFactor = 0.5
```

The value used to repeat a music file indefinitely. **See Also** F2DAudio for more information.

```
NSInteger const F2DRepeatAlways = -1
```

See Also F2DBody for more information regarding contacts and collisions.

```
NSInteger const F2DFilterGroupNone = 0
```

The value used to represent no direction, when an F2DBody stops moving. **See Also** F2DBody for more information regarding the direction property.

```
NSInteger const F2DDirectionNone = -1
```

Whether or not the F2DAudio class will play sound when messaged to do so. This can be changed to provide audio options for the user. **See Also** F2DAudio for more information.

```
BOOL F2DSoundOn = YES
```

Whether or not the F2DAudio class will play music when messaged to do so. This can be changed to provide audio options for the user. **See Also** F2DAudio for more information.

```
BOOL F2DMusicOn = YES
```

Generated by appledoc 2.1 (build 858).

F2DConstraint Class Reference

Inherits from	NSObject
Declared in	F2DConstraint.h

Overview

This class can joint two F2DBody objects to each other, and actively draw lines between them or two static points.

The F2DConstraint, just like the F2DBody, serves a variety of roles depending on the type property. The F2DConstraint class is primarily concerned with solving the physics between two jointed F2DBody objects. But the F2DConstraint class also has a drawing role. Colored lines with varying strokes can be drawn between F2DBody objects, or even between static points on the F2DWorld. This can be useful to “paint in” the mathematical constraint between two F2DBody objects. Note that only two F2DBody objects can be jointed per F2DConstraint, but each F2DBody can belong to multiple F2DConstraint objects, resulting in a string of joints. In order for F2DConstraint joint lines to be drawn, the F2D_DRAW_STATE_BASE and F2D_DRAW_STATE_CONSTRAINT definitions must be uncommented. See the F2DWorld class overview for information regarding the drawRect. Below is a description of each type an F2DConstraint can become as listed in the F2DConstants class.

- F2DConstraintTypeSpring: The default type for the F2DConstraint. The jointed F2DBody objects will attempt to retain a specified distance from each other by applying impulses to alter their velocity values.

- F2DConstraintTypeRope: The jointed F2DBody objects will attempt to retain a specified distance from each other only when that distance is exceeded (larger than the specified distance) by applying impulses to alter their velocity values.

- F2DConstraintTypeCosmetic: The jointed F2DBody objects will not be affected at all by the F2DConstraint physics. Use this type to still draw a line between the two jointed F2DBody objects, but no physics will be solved.

- F2DConstraintTypeLine: Use this type to draw a static line between two points on the F2DWorld. No F2DBody objects are jointed through this type and no physics will be solved at all. This type should literally just be used for drawing static lines between two points. It is typically used to “draw” line segments specified on the F2DWorld and maybe the F2DBody. See the F2DBody and F2DWorld lineSegments property for more information. Do not subclass.

Tasks

Constraint

`type` *property*
`body` *property*
`otherBody` *property*
`distance` *property*

`impulseLimits` *property*
`fracture` *property*
`stroke` *property*
`color` *property*
`drawLine` *property*

Constructors

– `initWithType:distance:body:otherBody:`
– `initWithType:drawLine:stroke:color:`

Properties

body

The first F2DBody jointed to the second F2DBody.

```
@property (nonatomic, weak) F2DBody *body
```

Discussion

This is one of the two F2DBody objects that are jointed together through the F2DConstraint. Physics are applied to the jointed F2DBody objects based on the type of F2DConstraint.

Declared In

`F2DConstraint.h`

color

The color of the drawn joint line.

```
@property (nonatomic, copy) UIColor *color
```

Discussion

By default this value is nil. No line will be drawn if the color is nil.

Declared In

`F2DConstraint.h`

distance

The length of the F2DConstraint.

```
@property (nonatomic, assign) CGFloat distance
```

Discussion

If the type of the F2DConstraint is a spring, then the joint will attempt to retain this distance between the body and otherBody. The the type of the F2DConstraint is a rope, then the joint will only attempt to retain this distance when the distance between the body and otherBody is greater than this amount, but not less.

Warning: *Domain:* (0,INFINITY]. Distance cannot be 0.

Declared In

F2DConstraint.h

drawLine

The f2dVectorLine representing the joint line to be drawn.

```
@property (nonatomic, assign) f2dVectorLine drawLine
```

Discussion

Regardless of the F2DConstraint type, this value directly controls where the joint line will be drawn. It will either be dynamic (line positions will be equal to the F2DBody objects' centroids) or static (line positions will be equal to the static F2DWorld points specified) depending on the type of the F2DConstraint. Note that the drawLine does not in any way use the f2dVectorLine bounds value, just the coordinates. See the class overview for more information.

See Also

F2DBody for information regarding drawing and the lineSegments property.

F2DWorld for information regarding drawing and the lineSegments property.

Declared In

F2DConstraint.h

elastic

Whether or not mass is factored into the F2DConstraint physics.

```
@property (nonatomic, assign) BOOL elastic
```

Discussion

If YES, then the F2DConstraint between the two F2DBody objects will account for the mass of the F2DBody objects. A greater mass object will take more effort to retain its F2DConstraint distance. If NO, mass will be ignored and the F2DBody objects will optimally attempt to retain their joint distance. Defaults to YES.

Declared In

F2DConstraint.h

fracture

The value at which the F2DConstraint will fracture.

```
@property (nonatomic, assign) CGFloat fracture
```

Discussion

An F2DConstraint will fracture if the distance between the body and otherBody becomes greater than the fracture value. If this happens, the F2DConstraint is removed from the F2DWorld, and both F2DBody objects will attempt to get notified if there constraintCallback is implemented and their cbConstraint property is set to YES. By default the fracture is not set within a reasonable range (there is no fracture).

Warning: *Domain:* (0,INFINITY]. Fracture cannot be 0.

See Also

F2DBody for information regarding the constraintCallback method.

Declared In

F2DConstraint.h

impulseLimits

The limits of the impulse that are applied to preserve the joint distance between the two F2DBody objects.

```
@property (nonatomic, assign) f2dVectorLimits impulseLimits
```

Discussion

Limiting these value can make the F2DConstraint weaker when applying impulses to retain the joint distance. By default there are no limits.

Declared In

F2DConstraint.h

otherBody

The second F2DBody jointed to the first F2DBody.

```
@property (nonatomic, weak) F2DBody *otherBody
```

Discussion

This is one of the two F2DBody objects that are jointed together through the F2DConstraint. Physics are applied to the jointed F2DBody objects based on the type of F2DConstraint.

Declared In

F2DConstraint.h

stroke

The thickness of the drawn joint line.

```
@property (nonatomic, assign) CGFloat stroke
```

Discussion

By default this value is 1. This property is strictly for appearance.

Warning: *Domain:* (0,INFINITY]. Stroke cannot be 0.

Declared In

F2DConstraint.h

type

```
@property (nonatomic, assign) NSInteger type
```

Discussion

The type determines the role of the F2DConstraint with respect to its properties. See the overview for more information about the type values for an F2DConstraint.

See Also

`F2DConstants for information regarding the F2DConstraintType constants.`

Declared In

`F2DConstraint.h`

Instance Methods

initWithType:distance:body:otherBody:

This initialization is for all types except the F2DConstraintTypeLine, as it provides physics based parameters.

```
- (id)initWithType:(NSInteger)type distance:(CGFloat)distance body:(F2DBody
*)body otherBody:(F2DBody *)otherBody
```

Parameters

type

The type/role of the F2DConstraint.

distance

The length of the F2DConstraint.

body

The first F2DBody in the joint.

otherBody

The second F2DBody in the joint.

Return Value

An initialized object.

Discussion

One of the currently two ways to initialize an F2DConstraint. Note that drawing joint lines is possible regardless of the F2DConstraintType. See the class overview.

Declared In

`F2DConstraint.h`

initWithType:drawLine:stroke:color:

This initialization is only for the F2DConstraintTypeLine, as it provides drawing based parameters.

```
- (id)initWithType:(NSInteger)type drawLine:(f2dVectorLine)drawLine stroke:
```

Parameters

type

The type/role of the F2DConstraint.

drawLine

The static line to be drawn.

stroke

The thickness of the joint line to be drawn.

color

The color of the joint line to be drawn.

Return Value

An initialized object.

Discussion

One of the currently two ways to initialize an F2DConstraint. Note that drawing joint lines is possible regardless of the F2DConstraintType. See the class overview.

Declared In

`F2DConstraint.h`

Generated by appledoc 2.1 (build 858).

F2DFunctions Class Reference

Declared in	F2DFunctions.h

Overview

All structs and functions.

A structure that contains both an x an y coordinative component of a two dimensional vector.

```
typedef struct  {
CGFloat x;
CGFloat y;
} f2dVector;
```

A structure that contains two vectors for the start and end points of a line segment and a bounds for the contact distance opposite the line normal.

```
typedef struct  {
f2dVector pos1;
f2dVector pos2;
CGFloat bounds;
} f2dVectorLine;
```

A structure that contains a 4-way limit system for both the max and min of x and y components.

```
typedef struct  {
CGFloat maxX;
CGFloat minX;
CGFloat maxY;
CGFloat minY;
} f2dVectorLimits;
```

A structure that contains two BOOLs, typically for options relating the the particular component.

```
typedef struct  {
BOOL x;
BOOL y;
} f2dVectorOptions;
```

A structure that contains a range using a max and min.

```
typedef struct  {
CGFloat max;
CGFloat min;
} f2dVectorRange;
```

Returns an f2dVector using the specified components.

```
static inline f2dVector F2DVectorMake(CGFloat x, CGFloat y)
```

Returns an f2dVector using the specified f2dVector length and quadrant sensitive angle in radians.

```
static inline f2dVector F2DVectorMakePolar(CGFloat magnitude, CGFloat angle)
```

Returns an f2dVector that results from adding two specified f2dVectors.

```
static inline f2dVector F2DVectorAdd(f2dVector vector1, f2dVector vector2)
```

Returns an f2dVector that results from subtracting two specified f2dVectors.

```
static inline f2dVector F2DVectorSubtract(f2dVector vector1, f2dVector vector2)
```

Returns an f2dVector that results from multiplying two specified f2dVectors.

```
static inline f2dVector F2DVectorScale(f2dVector vector1, f2dVector vector2)
```

Returns an f2dVector that results from dividing two specified f2dVectors.

```
static inline f2dVector F2DVectorDivide(f2dVector vector1, f2dVector vector2)
```

Returns a BOOL if the two specified f2dVectors are equal.

```
static inline BOOL F2DVectorEqual(f2dVector vector1, f2dVector vector2)
```

Returns the length of a specified f2dVector.

```
static inline CGFloat F2DVectorGetMagnitude(f2dVector vector)
```

Returns an f2dVector that results from changing the length of a specified f2dVector.

```
static inline f2dVector F2DVectorSetMagnitude(f2dVector vector, CGFloat magnitude)
```

Returns the distance between two specified f2dVector coordinative components.

```
static inline CGFloat F2DVectorGetDistance(f2dVector vector1, f2dVector vector2)
```

Returns the quadrant sensitive angle in radians of a specified f2dVector.

```
static inline CGFloat F2DVectorGetAngle(f2dVector vector)
```

Returns an f2dVector that results from rotating a specified f2dVector a specified number of radians about a specified origin.

```
static inline f2dVector F2DVectorRotateWithOrigin(f2dVector vector, f2dVector origin, CGFloat angle)
```

Returns an f2dVector that results from rotating a specified f2dVector a specified number of radians.

```
static inline f2dVector F2DVectorRotate(f2dVector vector, CGFloat angle)
```

Returns the dot product of two specified f2dVectors.

```
static inline CGFloat F2DVectorDotProduct(f2dVector vector1, f2dVector vector2)
```

Returns an f2dVector that results from normalizing a specified f2dVector.

```
static inline f2dVector F2DVectorNormalize(f2dVector vector)
```

Returns an f2dVector that results from the projection length of a specified f2dVector on a specified axis.

```
static inline CGFloat F2DVectorProject(f2dVector vector, f2dVector axis)
```

Returns a string formatted to contain the data from a specified f2dVector.

```
static inline NSString* NSStringFromF2DVector(f2dVector vector)
```

Returns an f2dVector converted from a specified CGPoint.

```
static inline f2dVector F2DVectorConvertCGPoint(CGPoint point)
```

Returns an f2dVector converted from a specified CGRect using the origin.

```
static inline f2dVector F2DVectorConvertCGRectOrigin(CGRect rect)
static inline f2dVector F2DVectorConvertCGRectSize(CGRect rect)
```

Returns a CGRect converted from a specified centroid and size.

```
static inline CGRect CGRectConvertF2DVectors(f2dVector centroid, f2dVector size)
```

Returns a CGPoint converted from a specified f2dVector.

```
static inline CGPoint CGPointConvertF2DVector(f2dVector vector)
```

Returns an f2dVectorLine using the specified line start and end positions and a bounds.

```
static inline f2dVectorLine F2DVectorLineMake(f2dVector pos1, f2dVector pos2, CGFloat bounds)
```

Returns a string formatted to contain the data from a specified f2dVectorLine.

```
static inline NSString* NSStringFromF2DVectorLine(f2dVectorLine vectorLine)
```

Returns an f2dVectorLimits using the specified component ranges.

```
static inline f2dVectorLimits F2DVectorLimitsMake(CGFloat maxX, CGFloat minX, CGFloat maxY, CGFloat minY)
```

Returns a string formatted to contain the data from a specified f2dVectorLimits.

```
static inline NSString* NSStringFromF2DVectorLimits(f2dVectorLimits vectorLimits)
```

Returns an f2dVectorOptions using the specified BOOL components.

```
static inline f2dVectorOptions F2DVectorOptionsMake(BOOL x, BOOL y)
```

Returns a string formatted to contain the data from a specified f2dVectorOptions.

```
static inline NSString* NSStringFromF2DVectorOptions(f2dVectorOptions vectorOptions)
```

Returns an f2dVectorRange using the specified range.

```
static inline f2dVectorRange F2DVectorRangeMake(CGFloat max, CGFloat min)
```

Returns a string formatted to contain the data from a specified f2dVectorRange.

```
static inline NSString* NSStringFromF2DVectorRange(f2dVectorRange vectorRange)
```

Returns an array of f2dVectorLines corresponding to the specified body shape and line bounds.

```
static NSArray* F2DShapeBody(NSInteger bodyShape, CGFloat bounds)
```

Returns an NSValue from a specified f2dVectorLine structure.

```
static NSValue* F2DEncode(f2dVectorLine vectorLine)
```

Generated by appledoc 2.1 (build 858).

F2DMessage Class Reference

Inherits from	NSObject
Declared in	F2DMessage.h

Overview

This class can send a message to a method after a delay using true game time.

F2DMessage objects use the integration loop or “game loop” of the engine to send messages after a specified delay. The message can also contain an object as an argument. Because the F2DMessage runs off the integrator, it will only solve the delay while the integrator is running; so when the game is “paused” all F2DMessage objects will be paused as well. This can be very useful for timing game events only while the game is actually running. This class should not be used to send messages after a delay when true game time is not needed, as it would provide unnecessary overhead. When an F2DMessage is running, it must be stopped, either by forcing it to stop, or waiting until the message is sent, in which case it would automatically stop itself. F2DMessage objects should not be used to loop events. F2DMessage objects can send a message with no delay (but there would be no purpose to do so). There are two ways to send a message to a method through the F2DMessage class.

- 1 You can send a message by initializing an F2DMessage object using the initWithDelay method, then telling the F2DMessage to run. This approach provides an instance of the F2DMessage object, so that its run and stop methods can be messaged.
- 2 You can send a message by messaging the class method messageWithDelay. This method does not return the instance of the F2DMessage class, so it runs automatically and can’t be stopped once ran until the message is sent. This approach provides an easier way of sending an F2DMessage, but provides less customization in the running and stopping of the F2DMessage.

Note that F2DMessage objects must be stopped once ran. All F2DMessage objects will stop themselves when the message is sent (even if it fails to send). Note that when an F2DWorld is destroyed, all F2DMessage objects are stopped. Do not subclass.

Tasks

Message

`+ messageWithDelay:object:target:selector:`
`– initWithDelay:object:target:selector:`
`– run`
`– stop`

Class Methods

messageWithDelay:object:target:selector:

Creates and sends a message to a method after a delay using true game time.

```
+ (void)messageWithDelay:(CGFloat)delay object:(id)object target:(id)target selector:
(SEL)selector
```

delay

The time it takes for the message to send in seconds.

object

An optional object that can be sent with the message. Pass nil if there is no object.

target

The class receiving the message.

selector

The method to be messaged.

Discussion

This method automatically initializes the F2DMessage object and runs it. It will be stopped once the message attempts to send. Below is an example of sending a message after a delay through this class method.

```
[F2DMessage messageWithDelay:3 object:someBody target:self selector:@selector(methodWithDelay:)];
```

Declared In

`F2DMessage.h`

Instance Methods

initWithDelay:object:target:selector:

Creates and prepares a message to be sent to a method after a delay using true game time.

```
- (id)initWithDelay:(CGFloat)delay object:(id)object target:(id)target selector:
(SEL)selector
```

Parameters

delay

The time it takes for the message to send in seconds.

object

An optional object that can be sent with the message. Pass nil if there is no object.

target

The class receiving the message.

selector

The method to be messaged.

Return Value

An initialized object.

Discussion

This method just initializes the F2DMessage object. It then must be messaged to run. It will be stopped once the message attempts to send, or you can message the stop method to stop it.

Declared In

`F2DMessage.h`

run

Runs the F2DMessage.

Discussion

Once the F2DMessage is running, it will attempt to message the selector when the specified delay is over. This method can only be messaged if the F2DMessage was created through the initWithDelay method.

Declared In

`F2DMessage.h`

stop

Stops the F2DMessage.

```
- (void)stop
```

Discussion

This method can be messaged to stop running an F2DMessage already in progress. This method can only be messaged if the F2DMessage was created through the initWithDelay method.

Declared In

`F2DMessage.h`

Generated by appledoc 2.1 (build 858).

F2DParticles Class Reference

Inherits from	CAEmitterCell
Declared in	F2DParticles.h

Overview

The object used for generating particle systems.

In the Frost2D Framework, very little is added to the existing CAEmitterCell because of the plethora of customization it already contains. What does primarily change, however, is the organization and layout of the CAEmitterCell with respect to its CAEmitterLayer. The F2DParticles class contains a CAEmitterLayer that is already setup. Upon the initialization of the F2DParticles class, the CAEmitterCell is automatically generated and placed on the CAEmitterLayer, seamlessly combining the two into one class, while retaining their properties. This approach allows F2DParticles to simply be added to and removed from the F2DWorld. Note that what is really being added to the F2DWorld is the CAEmitterLayer which contains the CAEmitterCell. It is highly recommended to read the documentation on the CAEmitterCell and CAEmitterLayer if not already familiar with their functions and properties. Aside from merging the CAEmitterCell and CAEmitterLayer into one class, the F2DParticles class also adds some new controls and easier ways to modify preexisting particles. Note that F2DParticles is not part of the F2DWorld drawRect in anyway, and is handled through core animation.

The F2DParticles class allows particles to be attached to an F2DBody through the appendingBody property which can easily “fake” the F2DParticles object’s physics. Methods for starting and stopping the F2DParticles and changing its particleFile also make working with particles a lot easier. Even the render mode of the CAEmitterLayer is automatically established (kCAEmitterLayerAdditive), and the emitterPosition property on the F2DParticles object simplifies particle coordinates. Finally, the changeValue method allows F2DParticles that have been already created to have their properties altered (e.g. birthRate etc). The reason for this is because when a CAEmitterCell is implemented, it cannot have its properties directly accessed unless a key value is utilized. The F2DParticles greatly simplifies this process through the changeValue method. Also note that F2DParticles will cause tremendous overhead when multiple F2DParticles are being displayed. Also note that the greater the amount of particles within the particle system, the more overhead your App will encounter. Also note that F2DParticles are primarily for visual effects, their appendingBody should handle the game logic/events. Also note that there are some Apps you can download to facilitate the process of making particles. Do not subclass.

Below is an example of creating one simple particle system.

```
F2DParticles* particles = [[F2DParticles alloc] initWithParticleFile:@"F2DSampleParticle2.png" emitterPosition:F2DVectorMake(568/2, 320/2)];
particles.scale = 1;
particles.color = [[UIColor colorWithRed:0.8 green:0.4 blue:0.2 alpha:0.1] CGColor];
particles.lifetime = 1;
particles.birthRate = 200;
particles.velocity = 100;
particles.emissionRange = 2*M_PI;
[world addParticles:particles];
```

Tasks

Particles

`emitterPosition` *property*
`particleFile` *property*
`emitterLayer` *property*
`active` *property*
`appendingBody` *property*
`appendingBodyOffset` *property*

– `changeValue:forKey:`
– `startEmitting`
– `stopEmitting`

Constructors

– `initWithParticleFile:emitterPosition:`

Properties

active

Whether or not the F2DParticles object is active.

```
@property (nonatomic, assign, readonly) BOOL active
```

Discussion
The F2DParticles object becomes active once added to an F2DWorld.

Declared In
`F2DParticles.h`

appendingBody

The F2DBody that the F2DParticles emitter position will attach to. The emitter position will follow the F2DBody object’s centroid.

Discussion

Use this property to make an F2DParticles object “follow” an F2DBody. This can be utilized to simulate physics with the F2DParticles object. If nil the F2DParticles will not attach to any F2DBody. Note that the integrator must be running for the particle system to “follow” the appending F2DBody.

See Also

`Frost2D for information regarding the integrator.`

Declared In

`F2DParticles.h`

appendingBodyOffset

An offset for the emitterPosition relative to the F2DBody object’s centroid.

```
@property (nonatomic, assign) f2dVector appendingBodyOffset
```

Discussion

Use this property to offset the coordinates of the emitter position from the appendingBody centroid.

Declared In

`F2DParticles.h`

emitterLayer

The layer which holds the emitter cell.

```
@property (nonatomic, strong) CAEmitterLayer *emitterLayer
```

Discussion

Access this property to change the emitter layer properties of the particle system such as its render mode.

See Also

`CAEmitterLayer for more information.`

Declared In

`F2DParticles.h`

emitterPosition

The center position where the particle system emits from.

```
@property (nonatomic, assign) f2dVector emitterPosition
```

Discussion

Note that the layer the particle system is on is not moving, but the particle emitter zone is. This provides a very dynamic feel. Also note that changing this value is actually changing the emitter layer’s emitter position.

Declared In

`F2DParticles.h`

particleFile

The particle file used as a source for the particle system.

```
@property (nonatomic, assign) NSString *particleFile
```

Discussion

This file will be blended with the particle system, depending on the rendering mode. By default the emitter layer rendering mode is kCAEmitterLayerAdditive. Note that the Frost2D Framework includes sample particles.

Declared In

`F2DParticles.h`

Instance Methods

changeValue:forKey:

Changes the property of an already existing particle system using a key path.

```
- (void)changeValue:(id)value forKey:(NSString *)key
```

Parameters

value

The value to pass to the property key path.

key

The name of the property to apply the value to.

Discussion

Use this property quickly alter the property of a preexisting F2DParticles. Note that once a particle system is created, the only way to change its properties is through this

```
[particles setValue:@0.0 forKey:@"birthRate"];
```

Declared In
`F2DParticles.h`

initWithParticleFile:emitterPosition:

Currently the only way to initialize an F2DParticles object.

`- (id)initWithParticleFile:(NSString *)file emitterPosition:(f2dVector)position`

Parameters
file

The name of the particle file to be used as a source for the particle system. Include the extension.

position

The center position where the particle system emits from.

Return Value
An initialized object.

Discussion
This method contains essential arguments.

Declared In
`F2DParticles.h`

startEmitting

Sets the F2DParticles object's birthRate to its last birth rate before stopEmitting was messaged.

`- (void)startEmitting`

Discussion
This method will only work if stopEmitting was called beforehand.

Declared In
`F2DParticles.h`

stopEmitting

Sets the F2DParticles object's birthRate to 0 and records its last birth rate so it can be reapplied when startEmitting is messaged.

`- (void)stopEmitting`

Discussion
This method is used in conjunction with the startEmitting method.

Declared In
`F2DParticles.h`

F2DWorld Class Reference

Inherits from	UIScrollView
Declared in	F2DWorld.h

Overview

The scene that stores all physics objects and particles.

The F2DWorld class is responsible for adding and removing F2DBody, F2DConstraint and F2DParticles objects from and to the scene or game view. Objects must be added to the F2DWorld in order for them to completely function. All objects added to the F2DWorld must be removed at some point and some objects like the F2DConstraint may remove itself (if it fractured). All objects added to the F2DWorld will be removed when the F2DWorld is destroyed. Only one F2DWorld can be created at a time. F2DWorld objects are created through the Frost2D class. The current active F2DWorld is stored as a property in Frost2D. See the Frost2D class for more information regarding F2DWorld creation and destruction.

Because the F2DWorld, like the F2DBody, comes from the UIKit, the F2DWorld can be handled all through interface builder. Simply drag in a custom UIScrollView and set its custom class to the F2DWorld. From here, F2DBody objects (UIImageView objects) can be dragged into the F2DWorld. Once the F2DWorld is linked, simply create it through Frost2D and the F2DWorld is ready. Note that unlike the F2DBody, the F2DWorld should be created through interface builder, although it can be programmatically created as well. If created through interface builder, the awakeFromNib method gets messaged. If programmatically created, the initWithFrame method gets messaged. Both of which should message the super class.

The F2DWorld also takes on a greater role through the drawRect method. By default, the F2DWorld will have its drawRect disabled. The drawRect provides significantly more overhead, especially in a UIScrollView. There are 4 changes that can be made to enable the drawing needs of your App. These changes are made by uncommenting preprocessor definitions related to the drawRect function. These definitions are found in the F2DConstants class an are described below.

- Uncomment the F2D_DRAW_STATE_BODY definition for the F2DWorld to actively draw all mathematical line segments attached to all F2DBody objects (dynamic drawing). This will greatly increase App overhead (ignoring the fact that drawRect itself produces a huge overhead). See the F2DBody lineSegments property for more information. Note that custom colors/strokes are not supported and must be set in the drawRect implementation. Enabling this drawing definition is typically used for debugging.

- Uncomment the F2D_DRAW_STATE_WORLD definition for the F2DWorld to actively draw all mathematical line segments attached to the F2DWorld (static drawing). This will slightly increase App overhead (ignoring the fact that drawRect itself produces a huge overhead). See the F2DWorld lineSegments property for more information. Note that custom colors/strokes are not supported and must be set in the drawRect implementation. Enabling this drawing definition is typically used for debugging.

- Uncomment the F2D_DRAW_STATE_CONSTRAINT definition for the F2DWorld to actively draw all mathematical joint lines of the F2DConstraint objects. This will slightly increase App overhead (ignoring the fact that drawRect itself produces a huge overhead). This is typically the drawing definition that would be uncommented, as it provides a lot of line drawing customization. See the F2DConstraint class for more information regarding line drawing.

- Uncomment the F2D_DRAW_STATE_BASE definition for line drawing to be able to occur. This would vastly increase App overhead as constant setNeedsDisplay messages would be sent and the drawRect apply, as this definition enables the drawRect itself (thus it makes no sense to have this uncommented but all other definitions commented). Keep this commented unless you need to draw lines on the F2DWorld, typically F2DConstraint lines. This must be uncommented for the other uncommented definitions to even do anything.

Note that for every active definition, App overhead is greatly increased. By commenting/removing these definitions, App overhead will be greatly reduced but the F2DWorld will not draw whatever is commented/removed. If the F2D_DRAW_STATE_BASE is commented/removed, no drawing will occur whatsoever regardless of the other definitions and the App will have no overhead from the drawRect. This should be done if line drawing by the F2DWorld is not needed at all. Remember that artwork inserted in a UIImageView should always take precedence over using the drawRect. The drawRect however is useful when debugging the App or when lines between points need to be drawn (usually in the case of the F2DConstraint). Also note that only the F2D_DRAW_STATE_CONSTRAINT provides line color and stroke customization through the F2DConstraint class; this can be used to "draw over" static F2DWorld line segments if more customization is needed. Note that multiple definitions can be uncommented (e.g. draw F2DConstraint lines and F2DWorld line segments). Do not subclass unless you really know what you're doing.

Tasks

Other

`lineSegments` *property*
`responder` *property*
`gravity` *property*
– `scrollToPosition:animated:`

Bodies

– `addBody:`
– `addBody:aboveBody:`
– `addBody:belowBody:`
– `removeBody:`

Particles

– `addParticles:`
– `addParticles:aboveBody:`
– `addParticles:belowBody:`
– `removeParticles:`

Constraints

– `addConstraint:`
– `removeConstraint:`

Properties

gravity

The acceleration due to gravity of the F2DWorld.

Discussion

All F2DBody objects will be accelerated by this value. Note that the F2DBody must have the worldPhysics property enabled.

See Also

`F2DBody for information regarding the worldPhysics property.`

Declared In

`F2DWorld.h`

lineSegments

The line segments that are mathematically placed in the F2DWorld in which F2DBody objects can interact with.

```
@property (nonatomic, copy) NSArray *lineSegments
```

Discussion

The line segments property of the F2DWorld, unlike the F2DBody, is completely static. Lines are specified by exact coordinates on the F2DWorld. Line segments on the F2DWorld provide significantly less overhead compared to line segments on the F2DBody (because the F2DBody line segments are dynamic and constantly calculated). Line segments on the F2DWorld should be used as much as possible and only in the case of a dynamic shape or line segment should the F2DBody line segments by used. F2DWorld line segments can be drawn by uncommenting the F2D_DRAW_STATE_WORLD preprocessor definition. See the class overview for information regarding the drawRect. Below is an example of placing one f2dVectorLine on the F2DWorld. See F2DFunctions for information regarding the f2dVectorLine and drawRect definitions.

```
f2dVectorLine line = F2DVectorLineMake(F2DVectorMake(0, 100), F2DVectorMake(568, 100), 10);
NSArray* array = @[F2DEncode(line)];
self.world.lineSegments = array;
```

Declared In

`F2DWorld.h`

responder

A UIViewController reference that will receive touch input when the F2DWorld receives touches.

```
@property (nonatomic, strong) UIViewController *responder
```

Discussion

Use this method so that the UIViewController which is storing the F2DWorld can receive touch events through the touchesBegan, touchesEnded, touchesCanceled, and touchesMoved methods. Leaving this property nil will not send any messages. Note that UIGestureRecognizers can be used (recommended) on the F2DWorld from the UIViewController instead of this property to detect touch input.

Declared In

`F2DWorld.h`

Instance Methods

Adds an F2DBody to the top of the F2DWorld.

```
- (void)addBody:(F2DBody *)body
```

Parameters

body

The F2DBody to add to the F2DWorld.

Discussion

A way of adding objects to the F2DWorld. This method is not needed if the F2DBody is added through interface builder.

Declared In

F2DWorld.h

addBody:aboveBody:

Adds an F2DBody to the F2DWorld above another F2DBody.

```
- (void)addBody:(F2DBody *)body aboveBody:(F2DBody *)body2
```

Parameters

body

The F2DBody to add to the F2DWorld.

body2

The F2DBody that the F2DBody being added will be above.

Discussion

A way of adding objects to the F2DWorld. This method is not needed if the F2DBody is added through interface builder.

Declared In

F2DWorld.h

addBody:belowBody:

Adds an F2DBody to the F2DWorld below another F2DBody.

```
- (void)addBody:(F2DBody *)body belowBody:(F2DBody *)body2
```

Parameters

body

The F2DBody to add to the F2DWorld.

body2

The F2DBody that the F2DBody being added will be below.

Discussion

A way of adding objects to the F2DWorld. This method is not needed if the F2DBody is added through interface builder.

Declared In

F2DWorld.h

Adds an F2DConstraint to the F2DWorld.

```
- (void)addConstraint:(F2DConstraint *)constraint
```

Parameters

constraint

The F2DConstraint to add to the F2DWorld.

Discussion

The F2DConstraint becomes active once added to the F2DWorld.

Declared In

`F2DWorld.h`

addParticles:

Adds an F2DParticles to the top of the F2DWorld.

```
- (void)addParticles:(F2DParticles *)particles
```

Parameters

particles

The F2DParticles to add to the F2DWorld.

Discussion

Note that the real object being added to the F2DWorld is the F2DParticles CAEmitterLayer property, which contains the F2DParticles system.

Declared In

`F2DWorld.h`

addParticles:aboveBody:

Adds an F2DParticles to the F2DWorld above an F2DBody.

```
- (void)addParticles:(F2DParticles *)particles aboveBody:(F2DBody *)body
```

Parameters

particles

The F2DParticles to add to the F2DWorld

body

The F2DBody that the F2DParticles being added will be above.

Discussion

Note that the real object being added to the F2DWorld is the F2DParticles CAEmitterLayer property, which contains the F2DParticles system.

Declared In

`F2DWorld.h`

addParticles:belowBody:

Adds an F2DParticles to the F2DWorld below an F2DBody.

Parameters

particles

The F2DParticles to add to the F2DWorld

body

The F2DBody that the F2DParticles being added will be below.

Discussion

Note that the real object being added to the F2DWorld is the F2DParticles CAEmitterLayer property, which contains the F2DParticles system.

Declared In

`F2DWorld.h`

removeBody:

Removes an F2DBody from the F2DWorld.

```
- (void)removeBody:(F2DBody *)body
```

Parameters

body

The F2DBody to remove from the F2DWorld.

Discussion

A way of removing objects from the F2DWorld.

Declared In

`F2DWorld.h`

removeConstraint:

Removes an F2DConstraint from the F2DWorld.

```
- (void)removeConstraint:(F2DConstraint *)constraint
```

Parameters

constraint

The F2DConstraint to remove from the F2DWorld.

Discussion

A way of removing objects from the F2DWorld.

Declared In

`F2DWorld.h`

removeParticles:

Removes an F2DParticles from the F2DWorld.

```
- (void)removeParticles:(F2DParticles *)particles
```

Parameters

particles

Discussion

A way of removing objects from the F2DWorld.

Declared In

`F2DWorld.h`

scrollToPosition:animated:

A method for scrolling the view of the F2DWorld to center around the specified position.

```
- (void)scrollToPosition:(f2dVector)position animated:(BOOL)animated
```

Parameters

position

The position to scroll to.

animated

Whether or not the F2DWorld should animate the scroll to the position. Pass NO to reduce overhead if constantly messaged.

Discussion

It will not scroll to the position if there is no room for the view to center around the position. This method can constantly be messaged to “follow” an F2DBody.

Declared In

`F2DWorld.h`

Generated by appledoc 2.1 (build 858).

Frost2D Class Reference

Inherits from	NSObject
Conforms to	UIAccelerometerDelegate
Declared in	Frost2D.h

Overview

The Frost2D class is responsible for handling system tasks associated with the physics, particles and audio engines.

Frost2D is the primary singleton class and must be accessed using the share class method. You will need to access Frost2D when doing any of the following. Do not subclass.

- Creating, destroying or accessing an F2DWorld
- Running, stopping or accessing the device accelerometer
- Running, stopping or accessing the integrator or “game loop”
- Accessing the constraints, messages and particles arrays
- Accessing the Frost2D delegate
- Loading music and sound

Tasks

Integrator/Accelerometer

`accelerometer` *property*

`integrator` *property*

`accelerometerValues` *property*

`accelerometerIsRunning` *property*

`integratorIsRunning` *property*

– `runIntegrator`

– `runAccelerometer`

– `stopAccelerometer`

– `stopIntegrator`

Arrays

`constraints` *property*

`messages` *property*

`particles` *property*

  `world` *property*
– `destroyWorld`
– `createWorld:`

Audio

  `sharedAudio` *property*
– `loadSoundFile:withKey:`
– `loadMusicFile:withKey:`

Delegate/Singleton

  `delegate` *property*
+ `share`

Properties

accelerometer

The device accelerometer.

```
@property (strong, nonatomic) UIAccelerometer *accelerometer
```

Discussion

Frost2D uses the device accelerometer to accelerate F2DBody objects using the device based on the properties defined in the F2DBody class.

See Also

`F2DBody for information regarding how to accelerate objects using the device.`

Declared In

`Frost2D.h`

accelerometerIsRunning

An indicator for whether or not the accelerometer is running.

```
@property (nonatomic, assign, readonly) BOOL accelerometerIsRunning
```

Discussion

The accelerometer is turned on and off by messaging the runAccelerometer and stopAccelerometer methods.

Declared In

`Frost2D.h`

accelerometerValues

```
@property (nonatomic, assign, readonly) f2dVector accelerometerValues
```

Discussion

This vector is used internally by the F2DBody class to accelerate F2DBody objects using the device accelerometer. These values can be read to determine the location of the device with respect to the x and y axis.

See Also

`UIAcceleration for more information.`

Declared In

`Frost2D.h`

constraints

The array of constraints.

```
@property (strong, nonatomic) NSMutableArray *constraints
```

Discussion

Each constraint is solved and/or drawn while the integrator is running.

Declared In

`Frost2D.h`

delegate

The Frost2D delegate.

```
@property (nonatomic, unsafe_unretained) id<Frost2DDelegate> delegate
```

Discussion

By accessing the Frost2D delegate, the following methods optionally become available. Once implemented, they will get messaged.

```
 - (void) engineDidIntegrate;

 - (void) audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:(BOOL)flag;
```

- The “engineDidIntegrate” method will be messaged after each integration.
- The “audioPlayerDidFinishPlaying” method will be messaged after each music file is finished playing and passes an argument for whether or not it successfully ended.

See Also

`AVAudioPlayer for more information regarding how the “audioPlayerDidFinishPlaying” method operates.`

Declared In

`Frost2D.h`

The physics integrator or “game loop.”

```
@property (strong, nonatomic) CADisplayLink *integrator
```

Discussion

Frost2D uses a CADisplayLink to integrate, solve and draw all physics and other system functions. The refresh rate is based on the F2DDeltaTime and F2DFrameInterval constants.

See Also

`CADisplayLink for more information.`

Declared In

`Frost2D.h`

integratorIsRunning

An indicator for whether or not the integrator is running.

```
@property (nonatomic, assign, readonly) BOOL integratorIsRunning
```

Discussion

The integrator is turned on and off by messaging the runIntegrator and stopIntegrator methods.

Declared In

`Frost2D.h`

messages

The array of messages.

```
@property (strong, nonatomic) NSMutableArray *messages
```

Discussion

Each message is solved while the integrator is running.

Declared In

`Frost2D.h`

particles

The array of particles.

```
@property (strong, nonatomic) NSMutableArray *particles
```

Discussion

Each particles' appending body is solved while the integrator is running.

See Also

`F2DParticles for more information about the appending body property.`

Declared In

sharedAudio

The audio engine singleton automatically created by Frost2D.

```
@property (strong, nonatomic, readonly) F2DAudio *sharedAudio
```

Discussion

The engine itself contains class methods for playing sound files (in Open-AL) and music files (using the AVAudioPlayer) that were loaded using Frost2D. The sharedAudio instance is seldom accessed.

See Also

`F2DAudio for more information regarding how to play/control audio.`

Declared In

`Frost2D.h`

world

The current F2DWorld created by Frost2D.

```
@property (weak, nonatomic, readonly) F2DWorld *world
```

Discussion

The current F2DWorld can be easily accessed at any time by simply accessing Frost2D. This is useful when the original F2DWorld cannot be directly accessed. Frost2D only supports one F2DWorld at a time, thus a F2DWorld must be removed before a new one is created.

Declared In

`Frost2D.h`

Class Methods

share

Generates a reference to the shared instance of this class.

```
+ (Frost2D *)share
```

Return Value

A pointer to the shared instance of this class.

Discussion

This is the primary way of accessing the Frost2D singleton. Once accessed, all the instance methods and properties become available. Note that the first time this method is messaged, Frost2D becomes allocated and initialized. So it is necessary to message this method at least once, typically when the App launches for the first time.

Declared In

`Frost2D.h`

Instance Methods

createWorld:

Creates a specified world.

```
- (void)createWorld:(F2DWorld *)world
```

Parameters

world

The F2DWorld to be created.

Discussion

Creating an F2DWorld allows F2DBody objects and other physics objects to be added to the world and integrated, solved and drawn by the Frost2D Framework.

See Also

F2DWorld for information regarding how to add objects to a @property world.

Declared In

Frost2D.h

destroyWorld

Destroys the current F2DWorld.

```
- (void)destroyWorld
```

Discussion

When an F2DWorld is destroyed, all F2DBody objects, F2DConstraint objects, F2DMessage objects and F2DParticles objects are removed and nullified. The F2DWorld itself is also removed. This method should be messaged when the “game is over,” typically when a UIViewController is being popped or dismissed. Note that after the F2DWorld is destroyed, it must be initialized and created again.

Declared In

Frost2D.h

loadMusicFile:withKey:

Loads a music file and assigns it a key.

```
- (void)loadMusicFile:(NSString *)musicFile withKey:(NSString *)key
```

Parameters

musicFile

The name of the imported music file. Do not include the .mp3 extension.

key

The key to give the music file so that it can be played using this key later on by the F2DAudio class.

Discussion

The music file must have a .mp3 extension. The music file only needs to be loaded once in the App life

See Also

`F2DAudio for more information regarding how to play/control audio.`

Declared In

`Frost2D.h`

loadSoundFile:withKey:

Loads a sound file and assigns it a key.

```
- (void)loadSoundFile:(NSString *)soundFile withKey:(NSString *)key
```

Parameters

soundFile

The name of the imported sound file. Do not include the .caf extension.

key

The key to give the sound file so that it can be played using this key later on by the F2DAudio class.

Discussion

The sound file must have a .caf extension, 44100 bit rate, stereo 16 bit, and 2 channels. Use the following terminal command to convert sound files into the correct format. The sound file only needs to be loaded once in the App life cycle. Once loaded, it can be played at any time using its assigned key by the F2DAudio class.

```
/usr/bin/afconvert -f caff -d LEI16@44100 inputSoundFile.ext outputSoundFile.caf
```

See Also

`F2DAudio for more information regarding how to play/control audio.`

Declared In

`Frost2D.h`

runAccelerometer

Turns the accelerometer on.

```
- (void)runAccelerometer
```

Discussion

While on, the F2DBody class, in combination with its properties, will continuously read the values obtained from the device accelerometer to accelerate itself.

Declared In

`Frost2D.h`

runIntegrator

Turns the integrator on.

```
- (void)runIntegrator
```

While on, the integrator will continuously update the Frost2D engine — solving, drawing and integrating all physics and functions. The F2DBody integration callback and the Frost2D engineDidIntegrate delegate methods are messaged after each integration (assuming they are implemented).

Declared In

`Frost2D.h`

stopAccelerometer

Turns the accelerometer off.

```
- (void)stopAccelerometer
```

Discussion

While off, no accelerometer values will be solved. The accelerometer should be kept off when not in use.

Declared In

`Frost2D.h`

stopIntegrator

Turns the integrator off.

```
- (void)stopIntegrator
```

Discussion

While off, the engine will not integrate, draw or solve any physics. Methods such as playing audio will still continue. The integrator should be kept off when not in use.

Declared In

`Frost2D.h`

Generated by appledoc 2.1 (build 858).